A Uniform Framework for the Formal Specification and Verification of Information Flow Security

Heiko Mantel

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken, 2003

Tag des Kolloquiums	11. Juli 2003
Dekan	Prof. Dr. Philipp Slusallek
Vorsitzender	Prof. Dr. Gert Smolka
Gutachter	Prof. Dr. Jörg H. Siekmann, Universität des Saarlandes
	Prof. Dr. David Basin, ETH Zürich, Schweiz
	Dr. John D. McLean, Naval Research Laboratory, Washington, D.C., USA

Short Abstract

In this thesis, we elaborate a uniform basis for the systematic investigation of possibilistic information flow properties. These properties are suitable for specifying security requirements formally such that they can be verified with mathematical rigor. We analyze the variety of known properties, propose new ones, and develop techniques that simplify their verification.

To this end, we introduce *MAKS*, a uniform framework for the investigation of information flow properties. The two basic ideas underlying *MAKS* are: firstly, to separate application-specific aspects of an information flow property from more application-independent aspects and, secondly, to express the latter aspects by assembling primitive building blocks. This modular representation provides a basis for reducing complex reasoning about information flow properties to reasoning about conceptually simpler building blocks. Following this approach, we analyze several information flow properties from the literature, elaborate their advantages and disadvantages, and derive a taxonomy of these properties. In this process, we discover several novel information flow properties that constitute improvements of known ones. Moreover, we exploit the modular representation for developing verification techniques for information flow properties. In particular, we derive unwinding results that reduce the verification of information flow properties to the verification of simpler unwinding conditions. We also derive compositionality results that support the verification of complex systems by reducing the overall verification task to the verification of the individual system components.

The applicability of our results is demonstrated by several examples and also by a complex case study from the area of language-based security.

Deutsche Kurzzusammenfassung

Die vorliegende Arbeit präsentiert einen Ansatz zur systematischen Untersuchung possibilistischer Informationsflusseigenschaften. Diese Klasse von Eigenschaften eignet sich für die formale Spezifikation von Sicherheitsanforderungen hinsichtlich Vertraulichkeit und Integrität und ermöglicht es, solche Anforderungen mit mathematischer Genauigkeit zu beweisen. Die aus der Literatur bekannten Informationsflusseigenschaften werden in der Arbeit eingehend analysiert und verglichen, neue Eigenschaften werden synthetisiert, und es werden Verifikationstechniken für Informationsflusseigenschaften vorgeschlagen.

Zu diesem Zweck wird ein einheitlicher Rahmen (*MAKS*) für die Analyse von Informationsflusseigenschaften vorgestellt. Dieser Rahmen basiert auf zwei grundlegenden Ideen: Erstens, die applikationsabhängigen Aspekte einer Informationsflusseigenschaft klar von den applikationsunabhängigen Aspekten zu trennen, und zweitens, die applikationsunabhängigen Aspekte in einer modularen Weise aus Bausteinen zusammenzusetzen. Die modulare Darstellung ermöglicht es, die Untersuchung komplexer Informationsflusseigenschaften auf die Untersuchung der primitiven Bausteine zurückzuführen. Unter Verwendung dieses Ansatzes werden verschiedene Eigenschaften aus der Literatur analysiert, ihre Vor- und Nachteile werden herausgearbeitet, und es wird eine Taxonomie dieser Eigenschaften hergeleitet. Dabei ergeben sich mehrere Verbesserungsmöglichkeiten, die zur Definition neuer Informationsflusseigenschaften führen. Außerdem werden Techniken entwickelt, die die Verifikation einer Informationsflusseigenschaft für ein gegebenes System auf ein konzeptionell einfacheres Problem zurückführen, nämlich auf die Verifikation von einfacheren, lokalen Bedingungen (*unwinding*-Technik) oder auf die Verifikation der Eigenschaft für die einzelnen Komponenten des Systems (kompositionale Verifikation).

Die Verwendung der vorgestellten Ergebnisse und vorgeschlagenen Konzepte wird in vielen kleinen Beispielen illustriert. Außerdem wird die praktische Verwendbarkeit an einer komplexen Fallstudie aus dem Bereich der statischen Programmanalyse gezeigt.

Extended Abstract

The security of information systems is of critical importance. For instance, many customers in electronic banking are concerned about keeping the balance of their accounts confidential. The integrity of bank accounts is important for their owners and also for the banking industry. In electronic commerce, some customers do not want their transactions to become public. For the success of electronic voting, the confidentiality and integrity of electronic votes are obvious prerequisites as they are the basis of democracy. However, the various security needs are opposed by manifold possibilities for attacks, in particular, since the advent of the Internet.

The key question is: How can we, as computer scientists, make sure that the critical systems that we build are indeed secure? To this end, formal methods offer a rigorous solution: By verifying formally that a system meets its requirements, one ensures with mathematical precision that this is indeed the case. Further advantages of formal methods are that requirements can be specified without ambiguities and that specifying a system and its requirements formally leads to a better understanding, which often reveals flaws before the actual verification begins.

In this thesis, we focus on the formal specification and verification of security requirements concerning confidentiality and integrity. The security properties that we investigate for this purpose belong to the class of possibilistic information flow properties. Using these properties, for instance, the confidentiality of data is expressed by the requirement that observations of untrusted users are not influenced by any confidential data. This means, in particular, that untrusted users must not be able to directly read the data. Expressing confidentiality by the lack of dependencies excludes also more subtle attacks that exploit malfunctioning or malicious programs with legitimate access to the confidential data (e.g. that a Trojan horse sends the data in some encoded form to an untrusted user). This is what makes information flow properties superior to access control policies. Another advantage of information flow properties is that they do not constrain the security mechanism (e.g. access control) that is used in a system for enforcing the security requirements. In other words, they capture what the requirements are without prescribing how they should be fulfilled.

To date, there is a (confusing) variety of different information flow properties that have been proposed for various reasons. For some of these properties, verification techniques have been developed already. However, information flow properties differ fundamentally from "usual" system properties like safety or liveness as they are properties of sets of traces.

In this thesis, our main objectives are: to provide a basis for the systematic investigation of information flow properties, to elaborate a better understanding of the properties known from the literature, to improve them where possible, and to develop techniques that simplify their verification. We sketch our main contributions in these respects in the following.

In Chapter 3, we introduce the Modular Assembly Kit for Security Properties (abbreviated *MAKS*), a framework for possibilistic information flow properties. The uniform representation in our framework provides a basis for the systematic investigation of a large number of properties. Formally, an information flow property is represented in *MAKS* by a pair consisting of a set of *views* and a *security predicate*. This representation cleanly separates the application-specific restrictions on the flow of information (expressed by the set of views) from the application-independent semantics of such restrictions (expressed by the security predicate). Security predicates are defined in a modular way by assembling *basic security predicates* (abbreviated BSPs) from a given set. These BSPs are the building blocks of *MAKS*. Based on this modular representation, the investigation of a complex information flow property can

be reduced to the investigation of each individual BSP from which the property is assembled. We exploit this possibility at several points of this thesis to simplify our investigations.

In Chapter 4, we analyze information flow properties from the literature, elaborate their relationship, and point out their advantages and disadvantages. So far, the diversity of information flow properties has been confusing and the relationship between many properties was not well understood. In particular, the selection of a suitable information flow property has been a difficult task during formal developments. Beyond the investigation of known properties, we show how novel properties can be derived in *MAKS*. The possibility to derive a new property in a goal-directed fashion when it is needed is preferable to inventing the property in an ad-hoc manner.

In Chapters 5 and 6, we develop techniques for simplifying the verification of information flow properties. Without such techniques, the verification of these properties is quite involved because they are closure properties of sets of traces. The unwinding approach reduces the verification of information flow properties to the verification of so-called unwinding conditions. These conditions are easier to handle during verification because they involve only individual computation steps rather than complete system behaviors. We show for several information flow properties how they can be proved by unwinding. Moreover, we simplify the verification of unwinding conditions by dropping an assumption made in previous work on unwinding (namely that unwinding relations must be equivalence relations). We show that this assumption is not essential for the soundness of the unwinding technique. For the verification of complex systems, the unwinding technique can be used in combination with a divide-and-conquer approach. To this end, we derive several compositionality results that reduce the verification of the overall system to the (conceptually simpler) verification of individual system components. We also present a classification of compositionality results that provides a more structured perspective on these results. Furthermore, we propose a novel information flow property, *weakened forward correctability*, that is preserved under composition, and we point out the improvement in comparison to previously proposed properties. All of our verification techniques are developed in a uniform way by exploiting the concepts of *MAKS*, i.e. we firstly develop the corresponding technique for the individual BSPs and then lift it over the assembling operation. A side effect of this approach is that verification techniques can be easily derived for an information flow property after this property has been represented in *MAKS*. This facilitates the use of novel information flow properties during formal developments.

We illustrate the applicability of our results in a complex case study from the domain of language-based security in Chapter 7. In the course of the case study, we provide also more general guidelines for such applications.

Ausführliche Zusammenfassung

Für die Akzeptanz vieler Informationssysteme ist deren Sicherheit von entscheidender Bedeutung. So ist es zum Beispiel vielen *e-banking*-Kunden wichtig, dass ihr Kontostand vertraulich bleibt. Die Integrität von Kontoständen ist für deren Inhaber bedeutsam, aber natürlich auch für das Bankgewerbe. Im *e-commerce* möchten manche Kunden nicht, dass ihre Transaktionen öffentlich werden. Für *e-voting* sind die Vertraulichkeit und die Integrität abgegebener Stimmen offensichtlich eine unabdingbare Voraussetzung. Jedoch steht den vielfältigen Sicherheitsbedürfnissen eine Vielzahl an Angriffsmöglichkeiten gegenüber, insbesondere wegen der globalen Vernetzung durch das Internet.

Die zentrale Frage ist, wie die Sicherheit kritischer Systeme bereits bei deren Entwicklung garantiert werden kann. Formale Methoden bieten hierzu eine stringente Antwort, da durch eine formale Verifikation der Sicherheitsanforderungen mit mathematischer Genauigkeit sichergestellt werden kann, dass diese tatsächlich vom untersuchten System erfüllt werden. Weitere Vorteile formaler Methoden sind, dass Anforderungen eindeutig spezifiziert werden können und dass ein verbessertes Verständnis der betrachteten Systeme und ihrer Anforderungen durch die Konstruktion formaler Spezifikationen gefördert wird. Das führt häufig zur Entdeckung von Fehlern bereits bevor die eigentliche Verifikation begonnen hat.

Die vorliegende Arbeit behandelt die formale Spezifikation und Verifikation von Sicherheitsanforderungen hinsichtlich Vertraulichkeit und Integrität. Zu diesem Zweck werden Sicherheitseigenschaften betrachtet, die zur Klasse der possibilistischen Informationsflusseigenschaften gehören. Die Vertraulichkeit von Daten wird mit diesen Eigenschaften zum Beispiel durch die Anforderung ausgedrückt, dass die Beobachtungen von Nutzern, die nicht vertrauenswürdig sind, nicht von diesen Daten beeinflußt werden dürfen. Dadurch wird insbesondere sichergestellt, dass nicht vertrauenswürdige Nutzer die vertraulichen Daten nicht unmittelbar lesen können. Darüberhinaus schließt eine solche Modellierung von Vertraulichkeit auch subtilere Attacken aus, die die Existenz fehlerhafter oder vorsätzlich bösartiger Programme mit erlaubtem Zugriff auf die vertraulichen Daten ausnutzen (z.B. Trojanische Pferde, die die vertraulichen Daten kodiert an nicht vertrauenswürdige Nutzer verschicken). Dieses ist ein wesentlicher Vorteil gegenüber Sicherheitsmodellen, die auf Zugriffskontrolle basieren. Ein weiterer Vorteil von Informationsflusseigenschaften ist, dass sie offenlassen, mit welchen Mechanismen die Sicherheitsanforderungen im System realisiert werden, d.h. sie beschreiben, was die Anforderungen sind, ohne die Realisierungsmöglichkeiten einzuschränken.

Es gibt eine Vielzahl von Informationsflusseigenschaften, die jeweils aus unterschiedlichen Gründen vorgeschlagen wurden, wobei für manche dieser Eigenschaften bereits entsprechende Verifikationstechniken entwickelt wurden. Es ist auch bekannt, dass es zwischen Informationsflusseigenschaften und herkömmlichen Systemeigenschaften, wie z.B. Lebendigkeit oder Sicherheit im Sinne von *safety*, grundsätzliche Unterschiede gibt.

Die wesentlichen Ziele dieser Arbeit sind, eine Grundlage für die systematische Untersuchung von Informationsflusseigenschaften zu schaffen, ein besseres Verständnis der bekannten Eigenschaften herauszuarbeiten, diese weiter zu verbessern und Techniken für ihre Verifikation zu entwickeln. Im folgenden wollen wir unsere Beiträge kurz skizzieren.

In Kapitel 3 wird ein Rahmen *MAKS* (für: *Modular Assembly Kit for Security Properties*) für die einheitliche Darstellung von Informationsflusseigenschaften vorgestellt. Dieser uniforme Rahmen bildet die Grundlage für die systematische Untersuchung einer Vielzahl von Informationsflusseigenschaften. Formal setzt sich die Darstellung einer Informationsflusseigen-

schaft in *MAKS* aus zwei Komponenten zusammen, nämlich einer Menge von sogenannten *views* (zur Spezifikation von Einschränkungen an den Informationsfluss) und einem *security predicate* (zur Definition der Semantik solcher Einschränkungen). Die anwendungsabhängigen Einschränkungen an den Informationsfluss werden also von der anwendungsunabhängigen Semantik klar getrennt. Weiterhin werden die *security predicates* in einer modularen Weise definiert, indem primitive Bausteine (*basic security predicates*, abgekürzt BSPs) zusammengesetzt werden, wofür eine Menge von BSPs zur Verfügung gestellt wird, die die Bausteine von *MAKS* bilden. Diese modulare Darstellung erlaubt es, die Untersuchung komplexer Informationsflusseigenschaften auf die Untersuchung der weniger komplexen BSPs zurückzuführen. Von dieser Technik werden wir im Verlauf der Arbeit häufig Gebrauch machen, um unsere Untersuchungen zu vereinfachen.

In Kapitel 4 werden die verschiedenen aus der Literatur bekannten Informationsflusseigenschaften untersucht, wobei die Vor- und Nachteile der einzelnen Eigenschaften detailiert herausgearbeitet werden. Ein tieferes Verständnis dieser Eigenschaften ist hilfreich, um die Auswahl einer geeigneten Eigenschaft für die formale Spezifikation gegebener Anforderungen zu erleichtern. Bisher war diese Auswahl recht schwierig, zum einen wegen der Vielzahl der bekannten Informationsflusseigenschaften und zum anderen, weil die Zusammenhänge zwischen den einzelnen Eigenschaften in vielen Fällen nicht hinreichend geklärt waren. Über die Untersuchung bereits bekannter Informationsflusseigenschaften hinausgehend wird aufgezeigt, wie neue Eigenschaften in *MAKS* synthetisiert werden können. Werden neue Eigenschaften benötigt, so ist diese zielgerichtete Synthese einem Ad-hoc-Vorgehen bei der Konstruktion vorzuziehen.

In den Kapiteln 5 und 6 werden Techniken zur Erleichterung der Verifikation von Informationsflusseigenschaften vorgeschlagen. Diese Eigenschaften sind Abschlusseigenschaften von Mengen von Systemverhalten und daher ist eine direkte Verifikation sehr aufwendig. Der *unwinding*-Ansatz vereinfacht die Verifikation von Informationsflusseigenschaften, indem er sie auf die Verifikation von sogenannten *unwinding*-Bedingungen reduziert, die einfacher zu handhaben sind, weil sie nur lokale Anforderungen an einzelne Berechnungsschritte stellen (statt an Mengen von kompletten Berechnungen). Diese Verifikationstechnik wird in Kapitel 5 auf Informationsflusseigenschaften erweitert, für die sie bisher nicht anwendbar war. Außerdem wird gezeigt, dass eine Annahme, die in bisherigen Arbeiten gemacht wurde (nämlich, dass *unwinding*-Relationen Äquivalenzrelationen sein müssen), für die Korrektheit der *unwinding*-Technik unwesentlich ist und somit fallengelassen werden kann. Dadurch wird die Verifikation der *unwinding*-Bedingungen vereinfacht. Für die Verifikation komplexer Systeme kann der *unwinding*-Ansatz in Verbindung mit einem *divide-and-conquer*-Ansatz verwendet werden. Zu diesem Zweck werden in Kapitel 6 mehrere Kompositionalitätsresultate hergeleitet, die die Verifikation eines Gesamtsystems auf die (konzeptionell einfachere) Verifikation der einzelnen Systemkomponenten zurückführen. Weiterhin wird eine Taxonomie entwickelt, um eine einheitliche und strukturierte Sicht auf die vorgestellten Kompositionalitätsresultate zu erhalten. Außerdem wird eine neue Informationsflusseigenschaft (*weakened forward correctability*) vorgeschlagen, die generell unter Komposition erhalten bleibt, und die, wie wir zeigen werden, eine Verbesserung gegenüber bestehenden Eigenschaften darstellt. Alle diese Verifikationstechniken werden in einer einheitlichen Weise unter Ausnutzung von *MAKS* hergeleitet, d.h. die jeweilige Technik wird zunächst für die einzelnen BSPs entwickelt und dann wird daraus die entsprechende Verifikationstechnik für die verschiedenen Informationsflusseigenschaften abgeleitet. Ein positiver Nebeneffekt dieser Vorgehensweise ist es, dass die Herleitung von Verifikationstechniken für neue Informationsflusseigenschaften recht einfach wird,

was wiederum die Verwendung neuer Eigenschaften in formalen Entwicklungen wesentlich erleichtert.

Die praktische Verwendung der vorgestellten Resultate und vorgeschlagenen Konzepte wird in vielen Beispielen illustriert und außerdem in Kapitel 7 an einer komplexen Fallstudie aus dem Bereich der statischen Programmanalyse im Detail gezeigt. Im Verlauf der Fallstudie werden auch Leitlinien für eine allgemeinere Verwendung unserer Resultate gegeben.

Publications

We have published excerpts from this thesis as follows:

An early version of *MAKS* was presented in [Man00c]. This version was less expressive and less generic than the one described in Chapter 3. Nevertheless, several information flow properties from the literature could be represented in this version already. These properties were also ordered by implication based on this representation. More recent versions of our framework and of the taxonomy were described as preliminaries in [Man02] and brief overviews on *MAKS* were given in [Man00a, Man00b].

Preliminary versions of our unwinding results were described in [Man00d]. These results were based on more restrictive assumptions than those in Chapter 5. Moreover, the relationship between information flow properties and refinement statements was pointed out, providing a basis for using simulation techniques to verify information flow properties. Appendix C elaborates these aspects in much more detail.

An earlier version of our compositionality results was published in [Man02]. Chapter 6 describes these results in more detail and, in particular, presents the complete proofs.

The case study in Chapter 7 is based on joint work with Andrei Sabelfeld. Previous versions of this case study have been presented in [MS01, MS03a]. These versions differ slightly from the one presented here (see Section 7.7).

Additional publications during my PhD studies have not been incorporated into this thesis for reasons of space and time. They are concerned with intransitive information flow [Man01a], refinement of secure systems [Man01b], language-based security [SM02], other applications of information flow properties [MSK+01, HMS03], tool support for information flow properties and, more generally, for formal methods [HMR+98, AHMS99, AHMS00, AHL+00, MS03b], formal methods for fault tolerance [MG00a, MG00b], and theorem proving [KMOS97, AM98, AMS98, Man98, MK98, MO99, KM03]. The contributions most closely related to this thesis are briefly discussed in Chapter 8.

Acknowledgments

First of all, I would like to thank my supervisor Jörg Siekmann for his support during my PhD studies. He offered critical advice and made many useful suggestions that helped me with my research and with writing this thesis. Moreover, the excellent infrastructure that he provided in his research group has been very beneficial during my work. I also thank him for his encouraging enthusiasm and for his support of my research stays abroad.

I am very grateful that David Basin accepted to serve on my PhD committee. He contributed his time and energy during a period of time when he was very busy with moving from Freiburg to Zürich and with establishing his new chair at ETH Zürich.

I thank John McLean for serving on my PhD committee and for many interesting discussions at various places. He motivated me to investigate the compositionality of information flow properties and made numerous other useful suggestions for improving this thesis. His articles had a major share in attracting me to the area of information flow security.

Beyond my PhD committee, my special thanks go to Dieter Hutter, Michael Kohlhase, and Christoph Kreitz, who also provided advice and guidance during my studies. Their opinions helped me to navigate. Moreover, I wish to thank Felix Gärtner, Alexandra Heidger, Dieter Hutter, and Axel Schairer for very useful comments on earlier versions of this thesis.

For fruitful collaborations, I am indebted to several colleagues. In particular, I thank Andrei Sabelfeld for a very efficient collaboration on language-based security that lead to the case study presented in Chapter 7. Many thanks to Dieter Hutter and Axel Schairer for collaborating on the application of my work to the security of multi-agent systems and mobile devices. For productive cooperations that also led to joint papers, I thank my coauthors Serge Autexier, Michael Balser, Felix Gärtner, Klaus P. Jantke, Matthias Kabatnik, Michael Kreutzer, Christoph Kreitz, Bruno Langenstein, Jens Otten, Wolfgang Reif, Georg Rock, Enno Sandner, Gerhard Schellhorn, Stephan Schmitt, Kurt Stenzel, Werner Stephan, Roland Vogt, Andreas Wolpers, and Alf Zuggenmaier.

I wish to thank all of my colleagues in the formal methods group at DFKI for a productive working atmosphere. In particular, I would like to thank Serge Autexier, Dieter Hutter, Axel Schairer, and Werner Stephan for many interesting discussions. Thanks to Axel also for being a very pleasant office mate. My research stays at Cornell University, Chalmers University, and Carnegie Mellon University contributed much to the success of my research. I would like to thank Bob Constable, Christoph Kreitz, Frank Pfenning, Andrei Sabelfeld, and David Sands for inviting me to these places and for providing an inspiring working environment.

There are numerous other people I am thankful to, e.g., for interesting discussions and for providing useful background information. Unfortunately, I cannot name all of them here. As representatives, I would like to explicitly mention Christoph Benzmüller, Michael Backes, Alexander Buchmann, Frédéric Cuppens, Riccardo Focardi, Harald Görl, Joshua Guttman, Fabio Martinelli, Jonathan Millen, Birgit Pfitzmann, John Rushby, Peter Ryan, Fred Schneider, and Steve Schneider.

Last but not least, I would like to thank my friends and my family for their support and encouragement during the last years. Above all, I thank Alexandra for her support that helped me to get through the rough times as well as for being part of the good times.

Contents

Chapter 1

Introduction

The important and still growing role information systems play in many aspects of todays life requires systems that can be trusted. However, the development of trustworthy systems is not an easy task. There are numerous examples of failures of computer systems and in many situations users have even become used to the fact that their computers do not work properly. Given that failures of computer systems can put life, liberty, and property at tremendous risk (cf. [Neu95]), there is a desperate need for improvements in the current situation.

Malfunctions of information systems can be attributed either to conceptual errors or to flaws in the implementation. In either case, the application of formal methods appears to be most promising for avoiding such errors during system development [Jon01]. The construction of formal models enforces that system requirements have been thoroughly understood when a system is built, formal notions are helpful to avoid ambiguities in requirement descriptions, and, moreover, they provide a basis for the verification of critical requirements.

Of course, the intended application determines which of the requirements are of critical importance for a given system. Besides functional correctness, i.e. proper outputs are produced for all inputs, at least four main classes of critical system properties can be distinguished: safety properties, real-time properties, fault-tolerance properties, and security properties [Rus94]. While safe or real-time systems usually operate in relatively benign environments, fault-tolerant or secure systems have to be reliable even if the environment is hostile. In fault tolerance, this hostility can be attributed to random events like, e.g., a processor crash or the notorious cosmic ray that flips a bit in memory. However, in security, the hostility of the environment is not accidental. Rather, pest programs like, e.g., Trojan horses or computer viruses, are hostile on purpose and countermeasures against attacks are being outmaneuvered in a goal-directed and increasingly sophisticated fashion.

Security engineering is the area of computer science that tackles the challenge to construct information systems that are reliable even if they operate in these hostile environments. When engineering secure systems, difficulties arise not only from possible attacks but also from the complexity of those systems. For example, it must be taken into account that the widespread interconnection of information systems, in particular by the Internet, allows various forms of almost borderless communication. Moreover, the information exchanged increasingly itself consists of software and it is very difficult, often practically impossible, for a user to fully assess what this software will do on their systems [Sch99]. This complexity makes it particularly important to have precise criteria for the security of systems as well as the possibility to rigorously check whether these criteria are fulfilled by a given system.

The aim of this thesis is to improve formal development techniques for secure systems. More precisely, the focus is on the formal specification and verification of a special class of security properties, namely restrictions on the information flow within a system. This class of security properties can express confidentiality as well as integrity requirements. Historically, this approach to specifying and verifying security requirements has evolved from Goguen and Meseguer's well known noninterference [GM82, GM84]. Based on the ideas underlying noninterference, numerous other information flow properties have been proposed in order to cope with issues like, e.g., nondeterministic system behavior or compositional system design. By providing a unified perspective on the various information flow properties, this thesis contributes to a deeper understanding of these properties. Beyond this, the state-of-the-art in the area of information flow security is advanced in several directions. In particular, we propose new information flow properties, derive compositionality results, and present novel unwinding theorems that ease the verification of information flow properties during formal developments in practice.

We propose a modular framework, *MAKS*, for the uniform representation of information flow properties. This framework supports the comparison of already known information flow properties and also the goal-directed development of new ones. This has led us to several novel information flow properties that turn out to be improvements of previously proposed properties. Based on the concepts of *MAKS*, we propose verification techniques that simplify the verification of information flow properties by reducing the overall verification task to the verification of simpler local conditions (unwinding technique) or by reducing the verification of the overall system to the verification of individual system components (compositional verification). The unwinding techniques presented are applicable for every information flow property represented in our framework, i.e. by representing an information flow property in *MAKS*, one obtains unwinding techniques without any further effort. Hence, the tedious derivation of unwinding techniques for every individual information flow property has become obsolete. The derivation of compositionality results is simplified in a similar way. Using our techniques, compositionality results for a given information flow property can be easily derived based on its representation in *MAKS*. The applicability of our results is demonstrated in a case study from the area of language-based security. More precisely, we show that a particular language-based analysis technique is sound wrt. some information flow property in our framework and apply our unwinding techniques and compositionality results in this process. This case study is also interesting in its own right because it establishes a rigorous connection between two different areas of computer security: language-based and specification-based information flow control. Further contributions, including results on refinement, intransitive information flow, language-based security, tool support, and further case studies are briefly summarized at the end of the thesis. We have published these results elsewhere in more detail (see page **x**).

In the remainder of this introduction, we will briefly review the state-of-the-art (in Section 1.1) in order to put our main contributions into perspective (in Sections 1.2–1.5). An overview of the structure of the thesis will be given in Section 1.6.

1.1 A Formal Methods Approach to Security Engineering

In this thesis, we pursue a *formal methods* approach. Using formal methods, the behavior of systems as well as properties of these systems can be modeled in precise terms. This makes it possible to specify requirements without ambiguities and to verify critical requirements with

mathematical rigor. More generally, it has been argued that formal methods can provide the thinking tools for the future of computer science [Jon01].

1.1.1 Formal Security Models

The formal specification of a system's security requirements is referred to as a *formal security model*. As depicted in Figure 1.1, a formal security model usually comprises: a system component, a security component, a satisfaction relation between the first two components, and a proof in some calculus that this satisfaction relation holds (e.g. [GM82]). While the system component specifies *how* the system operates, the security component specifies *what* security properties are required, and the verified satisfaction relation ensures that these security requirements are, indeed, fulfilled by the system.

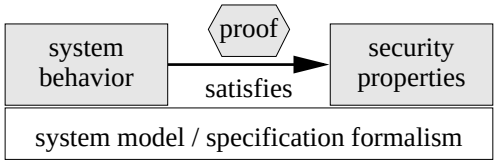


Figure 1.1: Structure of a formal security model

Security models can be constructed at different levels of detail. However, in general, a security model incorporates a "model" of the system, rather than the "real" system, and a "model" of the security properties, rather than the "real" properties. Simplifying from reality is done in model building, in particular, because the objective is to obtain representations that are easier to reason about than the real systems or real properties themselves. In model building, the two fundamental concepts used for simplification are *abstraction* (in order to leave out technical details) and *decomposition* (in order to distinguish different parts and then to focus on the particularly critical ones). The inverse operations for abstraction and decomposition, i.e. *refinement* and *composition*, are necessary in order to map the insights gained from a simplified model back to reality.

From a different perspective, refinement and composition provide a basis for a formal *stepwise development process* for critical systems. Starting from an abstract specification of a system's requirements, one *refines* the specification, e.g. by making design decisions or by adding technical details, and *decomposes* it into simpler specifications such that implementations of the component specifications can be composed to a suitable overall system (divide-and-conquer approach). This stepwise approach allows one to accompany different phases of system development by formal methods like, e.g. the definition phase, the design phase or the implementation phase, and to ensure that critical system properties that have been established in one phase are preserved when moving to the next phase. In particular, this allows one to apply formal methods already *before* system development has been completed. Applying formal methods *during* system development has the advantage that conceptual errors can be detected before they result in costly misdevelopments. It is widely accepted that constructing a formal model of a system's requirements, its architecture, and its behavior results in a deeper understanding of the system at the respective level. A positive side effect of the modeling process is that flaws like, e.g., requirements errors or omissions are often detected at this stage [Som96, p. 160]. Further confidence in the system is gained from a formal verification process, which usually reveals further flaws. To date, formal methods have become an integral part of the software engineering process for very critical systems.

For example, evaluation criteria like ITSEC [ITS91] or CC [CC99] require the construction of formal security models during early development phases (for E4/EAL 5 or higher).

1.1.2 A Trace-Based System Model

Usually, all components of a security model are based on a common (semantic) system model or a common (syntactic) specification formalism. The system model that we employ throughout this thesis is a trace-based model, i.e. the behavior of a system is modeled by a *set of traces* where every trace models a possible execution sequence of the system. Formally, a trace is a sequence of events where every event models an atomic action like, e.g. sending a message or receiving a message. Nondeterministic behavior is modeled by the *possibility* of different execution sequences, i.e. we employ a *possibilistic system model* (rather than a probabilistic one). Probabilities with which the different possible execution sequences will occur need not be specified in a possibilistic specification. This considerably simplifies the construction of system specifications, which, nevertheless, often remains a nontrivial task.

Similarly, system properties can be distinguished depending on whether they are only concerned with possible executions (possibilistic properties) or also with the probability of these executions (probabilistic properties). In this thesis, we focus on *possibilistic properties*, i.e. properties that are expressed in terms of possible executions and that do not depend on probabilities of these executions. For simplicity, we refer by "properties" always to "possibilistic properties" in the following. In a trace-based system model, many system properties, including safety (e.g. invariants) and liveness properties (e.g. termination), can be modeled by *sets of traces*. Properties of this kind are referred to as *properties of traces* because they are satisfied if every possible trace of the given system is contained in the set that specifies the property (cf. e.g. [AS85]). However, not every critical system property can be modeled by a set of traces. For example, many security properties are *properties of sets of traces*, i.e. they need to be modeled by *sets of sets of traces*. In particular, this is true for information flow properties [McL94b], i.e. the security properties investigated in this thesis.

The particular trace-based system model that we employ is that of *event systems*. This system model has been quite popular for the investigation of information flow properties (e.g. [McC87, GN88, JT88, Mil94, ZL97]). Using event systems, a given system is specified not only by a set of traces (modeling the system's behavior) but also by sets of input events and output events (modeling the system's interface). Event systems provide a suitable basis for the specification of complex systems, e.g. by composing a system from components that operate concurrently and communicate with each other via their interfaces. Nevertheless, this system model is conceptually quite simple and easy to apply.

1.1.3 Modeling Security Requirements

It is not trivial to specify the security requirements for a given system correctly. Let us give an example in order to illustrate the kind of mistakes that can easily be made.

The confidentiality of some value within a system might naively be expressed by the requirement that this value must never leave the system. For example, if you have $\$5,342$ in your bank account then confidentiality of your balance would be expressed by the requirement that the number $5,342$ must not leave your PC (e.g. as part of an e-mail). In particular, this implies that a program that helps you to administer your bank account does not send your balance to the supplier of the program. However, this naive approach has serious deficiencies.

Firstly, it is *too restrictive*: e.g. if 5,342 happens to be the registration number of your software license then it would be impossible to e-mail a license agreement (containing the registration number) to the supplier of the program. Nevertheless, it is quite obvious that this would not reveal your balance. Secondly, the naive approach is *too liberal*: e.g. the program might increment the confidential value, then send 5,343 (instead of 5,342) to its supplier, and the supplier would obtain the balance by decrementing the received number.

This illustrates that the critical issue is not whether any output of the system *contains* confidential values but rather whether it *depends* on confidential information. Hence, confidentiality should be expressed as the *lack of dependencies on confidential information*. This approach is taken by the security properties considered in this thesis. More specifically, information flow properties are investigated that are based on the idea of *noninterference*, which is the topic of the following section.

1.1.4 Specifying Security Requirements with Noninterference

Numerous different information flow properties have been proposed, including noninterference [GM82], nondeducibility [Sut86], restrictiveness [McC87], separability [McL94a], and many others. These properties differ in their formal definitions, sometimes in quite subtle ways. Nevertheless, they share the following common intuitive understanding of noninterference:

> *A group of processes is* noninterfering *with another group of processes if the actions of the first group have no effect on the observations of the second group.*

Hence, if a group of processes is noninterfering with another group then there is *no information flow* from the first group to the second group (the observations do not depend on the actions). This means, on the one hand, that processes in the first group cannot reveal any secrets (e.g. passwords, encryption keys, classified data, or private information like your balance) to processes in the second group and, on the other hand, that processes in the second group cannot be corrupted by processes in the first group. These observations provide the basis for modeling *confidentiality* requirements as well as *integrity* requirements between processes.

However, *noninterference is not limited to specifying security requirements between processes*. Namely, restrictions on the flow of information between any *subjects* (e.g. processes, threads, or users) can be specified. Moreover, noninterference can also be used to specify restrictions on the flow of information *within processes* or *within threads*. For example, this is necessary to specify security requirements for a process that accesses confidential information, that also communicates with untrusted users, but that must not communicate any confidential information to these untrusted users. In order to express security requirements with this fine granularity, it must be possible to require noninterference between individual atomic actions. However, it would be quite inconvenient to specify for every pair of events whether they may interfere with each other or not. Therefore, the concept of *security domains* is introduced. Related events are grouped into security domains and noninterference is specified in terms of security domains rather than in terms of individual events. Giving names (D_1, D_2, ...) to security domains allows one to specify security requirements abstractly (e.g. by the statement $D_1 \not\rightsquigarrow D_2$) without having to consider the specific sets of events.

Note that the only difference between confidentiality requirements and integrity requirements (when expressed in terms of noninterference) is the *direction* in which security domains must not interfere. In case of confidentiality, the protected security domains must not interfere with the environment and, in case of integrity, the environment must not interfere with

the protected security domains. Therefore, the integrity problem is the dual of the confidentiality problem. Throughout this thesis, we will take the viewpoint of confidentiality, i.e. we interprete a noninterference statement $D_1 \not\rightsquigarrow D_2$ as "the activities in domain D_1 are confidential for domain D_2". However, most of our explanations can be translated to integrity by exploiting the duality between confidentiality and integrity. Since we focus on confidentiality in the presentation, we will use the term "security" synonymously with "confidentiality".

1.1.5 Noninterference Definitions

In Section 1.1.4, we have informally introduced noninterference but in order to verify that noninterference statements like $D_1 \not\rightsquigarrow D_2$ are actually satisfied by a given system we need a *formal definition.*

The system model underlying Goguen and Meseguer's definition of noninterference is that of deterministic state machines. In this model, a system accepts commands from its various users as input, processes these commands according to a deterministic transition function, and returns outputs to the users according to a deterministic output function. For a given system, noninterference holds between two users U_1 and U_2 (expressed by $U_1 \not\rightsquigarrow U_2$) if the output given to U_2 does not depend on whether U_1 provides input to the system or not. More precisely, $U_1 \not\rightsquigarrow U_2$ holds if for every sequence of commands the output to U_2 after execution of this sequence is identical to the output to U_2 after execution of the purged sequence, i.e. the sequence that results after removing all commands issued by U_1.[1] Intuitively, the idea is: if U_2's output does not depend on whether U_1 inputs any commands then there cannot be any information flow from U_1 to U_2. In particular, it is not possible that confidential information flows from U_1 to U_2.

There are many other formal definitions of noninterference. Even the noninterference definition by Goguen and Meseguer [GM82] is predated by earlier work in this direction [FLR77, Rus81b]. Numerous further variants of noninterference have been proposed over the last 20 years (e.g. [Sut86, McC87, Fol87, Jac88, JT88, GN88, McL90, WJ90, O'H90, Gra91, Rya91, BC92, MC92, Rus92, McL94a, FG95, Ros95, ZL97, RG99, Sch01, FR02]). In order to avoid confusion with Goguen and Meseguer's notion of noninterference, these later variants were given different names like, e.g., nondeducibility, generalized noninterference, restrictiveness, or noninference. Nevertheless, all of these information flow properties intuitively share the same underlying idea, namely the one of noninterference.

1.1.6 Related Work on Noninterference

This section focuses just on the main developments on noninterference that are relevant for the contributions of this thesis. It is not intended as a complete overview on all work in this area. Further discussions of related work are contained in the introductions and summaries of the various chapters in this thesis. For a general overview on noninterference and other security models, we refer to [McL94b]. More specific overviews on intransitive information flow and on refinement of information flow properties can be found in the related work sections of our articles on these topics [Man01a, Man01b].

Noninterference-like Information Flow Properties for Nondeterministic Systems. One major line of research has been concerned with finding formal definitions of noninterfer-

[1] Formally: $\forall t \in C^*.\ output(execute(t), U_2) = output(execute(purge(t, U_1)), U_2)$

ence that are appropriate for deterministic systems as well as for nondeterministic systems.

Goguen and Meseguer's definition of noninterference is only adequate for deterministic systems because the underlying system model, i.e. that of deterministic state machines, is not well suited for modeling nondeterministic systems. Historically, Sutherland's nondeducibility [Sut86] was the first information flow property suitable for nondeterministic systems. Subsequently, many other information flow properties that are also suitable for deterministic as well as for nondeterministic systems followed, e.g., generalized noninterference [McC87], noninference [O'H90], or separability [McL94a]. Typical motivations for proposing a new property were examples of systems that are intuitively insecure although they satisfy a previously proposed information flow property (motivating the definition of a more restrictive property), examples of systems that violate a previously proposed information flow property although they are intuitively secure (motivating the definition of a less restrictive property), or the choice of a system model for which previously no information flow property had been proposed. As a consequence of this line of research, to date, multiple information flow properties co-exist, none of them being superior to all others, in general.

The confusing diversity of information flow properties has made comparisons necessary because it is sometimes hard to see the similarities and differences of these properties. In particular, it is hardly possible to compare two properties directly if the underlying system models differ considerably. System models used in the context of information flow security include event systems [McC87, JT88, Mil94, ZL97], other trace-based models [McL94a], nondeterministic state machines [McC90, WJ90], the process algebra CSP [Hoa85] with different semantics [RS99] (trace semantics [Jac89, Sch01], ready sets semantics [Rya91], and failure divergence semantics [Ros95]), the process algebra SPA [FG95], and probabilistic models [McL90, WJ90, Gra91]. Therefore, before comparing information flow properties that are based on different system models, their definitions need to be translated into a common system model. This means, rather than comparing information flow properties directly, their translations are compared. For example, McLean translated information flow properties into a trace-based model before comparing them to each other [McL94a, McL96]. In contrast to this, Focardi/Gorrieri based their comparison of information flow properties on labeled transition systems and the process algebra SPA [FG95] and Zakinthinos/Lee used event systems [Zak96, ZL97]. Common to all these comparisons is that they focus on possibilistic information flow properties rather than probabilistic ones.[2] As we also focus on possibilistic properties in this thesis, we use the term "information flow property" synonymously with "possibilistic information flow property" in the following.

The previous comparisons of information flow properties have led to a better understanding of these properties. In particular, taxonomies have been derived that show which information flow properties are more restrictive than others. Moreover, uniform representations of information flow properties from the literature revealed similarities between properties that look very different in their original definitions. It has even become possible to derive general

[2]Possibilistic information flow properties are conceptually simpler than probabilistic information flow properties. Moreover, possibilistic properties do not require the construction of a probabilistic system specification (a possibilistic system specification suffices) and, hence, they are easier to apply in formal developments. The price paid for these advantages is that probabilistic covert channels (ones that could be exploited based on Shannon's information theory [Sha48]) cannot be ruled out by possibilistic information flow properties [WJ90]. For a description of probabilistic information flow properties like the (applied) flow model [McL90, Gra91] or probabilistic noninterference [Gra91] and a discussion of the trade-off between possibilistic information flow properties and probabilistic ones, we refer to [WJ90, McL94b].

results concerning compositionality that hold for whole classes of information flow properties rather than only for individual properties. However, none of the previous comparisons covers all the important information flow properties from the literature. Moreover, a clear identification of the kinds of systems for which the various properties are good for is missing. Finally, there seems to be a trade-off between expressiveness and uniformity in the underlying frameworks. While the uniform framework (selective interleaving functions) underlying McLean's comparison [McL94a, McL96] provides a basis for deriving general compositionality results for classes of information flow properties, this framework is not expressive enough to represent all properties that are of interest. In contrast to this, the frameworks underlying the comparisons by Focardi/Gorrieri [FG95], Peri/Wulf/Kienzle [PWK96], and Zakinthinos/Lee [Zak96, ZL97] are, in principle, quite expressive but they lack uniform concepts for the representation of information flow properties that could be used to show general results about classes of properties.

Verification Techniques. Another line of research has been concerned with the development of techniques that simplify the verification of information flow properties. Although a direct verification of information flow properties is also possible (cf. [McL92]), this is usually a complex task. The key observation for the development of verification techniques for information flow properties was that the global requirement imposed by an information flow property can be reduced to more local conditions that involve only individual transitions [GM84] and, hence, are easier to handle during verification. The proof that the local conditions imply the given information flow property involves an inductive argument and, therefore, this approach is known under the name *unwinding*. The local verification conditions are called *unwinding conditions* and the theorem ensuring the implication of the information flow property is called the *unwinding theorem*.

Traditionally, unwinding assumes that system states can be grouped into classes such that states in the same class are indistinguishable from each other for a potential attacker. If actions that involve confidential information (like, e.g., the receipt of a secret message) always cause a transition from a state in one class to another state in the same class then, intuitively, these actions do not interfere with the observations of the attacker. This is because the states before and after these actions are indistinguishable for him. Another typical unwinding condition is that the classification of states must be preserved under stepwise execution, a requirement that bears similarities to the requirement imposed on congruence relations. Formally, the grouping of states into classes is usually given by a relation, the so called *unwinding relation* that must be specified before the unwinding conditions can be proven.

Starting with an unwinding theorem for noninterference [GM84], unwinding theorems for various other information flow properties have been proposed [HY87, Jac90, GCS91, Rya91, MC92, Rus92, Mil94, Pin95, Zak96, RS99]. Each of these unwinding theorems aims at the verification of one particular information flow property. Examples of properties for which unwinding theorems have been developed are noninterference, generalized noninterference, forward correctability, and the perfect security property.

However, unwinding results do not exist for all information flow properties of interest. Moreover, it is somewhat annoying that unwinding theorems for different properties are derived separately although the proofs of different unwinding theorems often have a lot in common. One might hope that these similarities provide a basis for more general results on unwinding. However, no such general results have been proposed so far.

Preservation under Composition. The development of large and complex systems is only feasible if a divide-and-conquer approach is applied: Requirements for the overall system are divided into subtasks, these subtasks are assigned to system components, and, after the specification of component interfaces, components can be developed independently. Upon composing the system, the satisfaction of the overall requirements should follow from the fact that all components fulfill their specifications. Any need to inspect implementation details of system components in this process would not only violate the idea of the divide-and-conquer approach, but it would also lead to an undesired increase of complexity. Moreover, since vendors are often unwilling to reveal details about their products, implementation details about some components might simply not be available.

Unfortunately, information flow properties are not preserved under composition, in general [McC87]. However, certain information flow properties are known to be preserved under composition. For example, McCullough's restrictiveness is, in general, composable [McC87]. In fact, historically, restrictiveness was the first composable information flow property. Other information flow properties that are preserved under arbitrary composition are, e.g., forward correctability [JT88], separability [McL94a], and the perfect security property [ZL97]. For certain information flow properties that are not preserved under composition in general, restrictions are known under which these properties are preserved [McL94a, McL96, ZL96]. For example, generalized noninterference is preserved if components are composed sequentially in the form of a cascade such that communication occurs only in one direction. There are many other studies that have investigated the compositionality of information flow properties (e.g. [GN88, Mil90, O'H92, BCC94, RW95, Jür00, Sch01]).

Nevertheless, a couple of important problems have remained unsolved. In particular, a uniform theory of composition for secure systems is still missing. To date, the various compositionality results are only loosely connected. Deeper insights, on how these results are related, would be desirable. It would also be helpful to more deeply understand why some properties are composable while others are not. Moreover, the derivation of compositionality results is often a quite difficult task and one might wish to have techniques that simplify the derivation of such results. McLean's framework of selective interleaving functions [McL94a, McL96] is certainly a step towards a uniform theory of composition for secure systems. This framework also simplifies the derivation of compositionality results. However, several information flow properties cannot be represented in the framework and, hence, it obviously cannot help in the derivation of compositionality results for such properties. An example for a known result that cannot be derived in that framework is the compositionality of forward correctability.

1.2 *MAKS*: A Framework for Information Flow Properties

In this thesis, we propose the Modular Assembly Kit for Security Properties (abbreviated by *MAKS* in the following), a uniform framework for information flow properties. The main objective of our framework is to serve as a basis for reasoning about information flow properties. This means, *MAKS* is intended as a thinking tool for a particular class of security properties.

The three main features of *MAKS* are:

Expressiveness The well known information flow properties can be represented in our framework including generalized noninterference [McC87, McL94a, ZL97], forward correctability [JT88], nondeducibility for outputs [GN88], noninference [O'H90], generalized noninference [McL94a], separability [McL94a], the perfect security property [ZL97], and

several further novel properties that we propose in this thesis.

Uniformity Information flow properties are represented in a uniform way as they are specified by pairs consisting of a *set of views* and a *security predicate*. The set of views defines the application-specific security requirements where each view roughly corresponds to a noninterference statement. The security predicate provides a formal definition of noninterference. Security predicates are assembled from so-called *basic security predicates* (abbreviated by BSPs in the following), the basic building blocks of *MAKS*. This modular representation of information flow properties is the reason why we call our framework an *assembly kit*. Intuitively, each BSP from which a given information flow property is assembled can be regarded as a "basic ingredient" of that property. Formally, a BSP is a very primitive information flow property.

Simplification The modular representation of information flow properties in *MAKS* allows one to reduce reasoning about complex information flow properties to reasoning about BSPs. For example, information flow properties can be compared to each other by identifying common BSPs in their respective representations and then comparing the BSPs in which they differ. Similarly, an information flow property can be verified by verifying each BSP from which it is assembled. Moreover, proving compositionality results for information flow properties can be reduced to proving compositionality results for BSPs. These reductions simplify reasoning about information flow properties considerably. Moreover, after a result has been proven for a given BSP, this result can be exploited during the investigation of multiple information flow properties.

Other frameworks either emphasize expressiveness (at the cost of uniformity) or uniformity (at the cost of expressiveness). For example, the frameworks by Focardi/Gorrieri [FG95] and by Peri/Wulf/Kienzle [PWK96] are, in principle, expressive enough to represent most information flow properties. However, these frameworks lack concepts that are specifically targeted at a uniform representation of information flow properties. In contrast to this, the framework by McLean [McL94a, McL96] provides concepts for the uniform representation of information flow properties. However, these concepts are not expressive enough to represent all information flow properties that are of interest.

Among the previously proposed frameworks, McLean's framework of selective interleaving functions is the only one that considerably simplifies the investigation of information flow properties. In particular, it provides a basis for deriving general compositionality results for classes of information flow properties. In this respect, it is the only framework that is comparable to *MAKS*. However, the limited expressiveness of McLean's framework implies that the number of properties that can be investigated is quite restricted. Moreover, the derivation of unwinding theorems appears to be outside the scope of that framework. In other words, using *MAKS* more results can be derived for a greater number of properties.

The concepts for the uniform representation of information flow properties in *MAKS* differ considerably from the concepts used in the prior frameworks [McL94a, McL96, Zak96, ZL97]. A key novelty of *MAKS* is that information flow properties are represented in a *modular* way. To find basic concepts that are *sufficiently general* to represent the most important information flow properties as well as *sufficiently specific* to simplify reasoning about the represented properties was one of the main difficulties in the development of *MAKS*. After these concepts had been defined, another difficulty was to identify suitable building blocks that resemble "basic ingredients" of known information flow properties, i.e. the various BSPs.

The benefit of our modular representation based on these concepts is that reasoning about complex information flow properties can be reduced to reasoning about simpler BSPs. Following this approach, we derive several results with the objective of *deepening our understanding* of information flow properties. For example, we present a *taxonomy of known information flow properties*. Based on the insights gained from our investigations, we derive *novel information flow properties* with the objective of overcoming certain deficiencies of known ones. For example, we derive *weakened forward correctability* from forward correctability [JT88] because the latter property is unnecessarily restrictive. Other examples of novel information flow properties that we propose are GNI^*, $IBGNI^*$, NDO^*, and FC^* (see Chapter 4).

1.3 Verification Techniques

The verification techniques that we propose in this thesis follow the unwinding approach [GM84]. We introduce a collection of unwinding conditions and prove corresponding unwinding theorems, each of which states that some combination of unwinding conditions implies some information flow property. Unwinding a given information flow property requires one to choose an unwinding theorem for this property, to prove all unwinding conditions named in this theorem, and then to apply the unwinding theorem in order to complete the proof. In comparison to verifying an information flow property directly, the advantage of unwinding is that unwinding conditions are easier to handle during verification. This is because unwinding conditions are expressed in terms of individual actions and, hence, are conceptually simpler than information flow properties, which are expressed in terms of complete execution sequences. Like in most other approaches to unwinding, we use so called unwinding relations (i.e. binary relations between states) in the definition of our unwinding conditions. Typical requirements are that the states before and after certain actions must be related by the unwinding relation or that the unwinding relation is preserved under stepwise execution.

In unwinding theorems, the unwinding relation is existentially quantified. Hence, an unwinding relation must be chosen before the unwinding conditions can be proven. In practice, constructing an appropriate unwinding relation is often more difficult than the actual proof of the unwinding conditions. Therefore, it is quite unfortunate that prior approaches to unwinding further complicate this construction by imposing additional side conditions on the unwinding relation, namely that it must be an equivalence relation (e.g. in [HY87, Rya91, GCS91, MC92, Rus92, Mil94, Zak96, RS99]). An interesting improvement of our approach is that we do not impose any such side conditions and, in particular, do not require that unwinding relations are reflexive, transitive or symmetric. Having no side conditions is convenient in practice because it permits more freedom in the construction of unwinding relations. Eliminating the need for side conditions has also more fundamental advantages. Namely, there are cases where suitable unwinding relations exist but none of them is an equivalence relation. In these cases, proof attempts with the known unwinding techniques would be doomed to fail while the proof can be completed with our techniques.

Our approach is not targeted at verifying a particular information flow property. Rather, we present unwinding techniques that can be used for *all* information flow properties represented in *MAKS*. This includes not only properties known today but also further properties that might be proposed in the future. This general applicability of our unwinding techniques has become possible by the modular representation of information flow properties. Since BSPs are assembled by conjunction, an information flow property can be verified by verifying

each BSP from which it is assembled. The unwinding conditions and unwinding theorems that we propose aim at the verification of BSPs and, hence, can be exploited during the verification of all information flow properties that are assembled from these BSPs. This means, by representing an information flow property in *MAKS*, one obtains unwinding techniques "for free". In particular, our verification techniques can be applied to nondeducibility for outputs [GN88], separability [McL94a], and many other information flow properties for which previously no unwinding techniques existed. Previous unwinding results do not offer this generality because they aim at the verification of some *specific* information flow property only. For example, the unwinding result in [GM84] is only applicable to the original definition of noninterference [GM82]; the respective unwinding results in [HY87, Rus92, Pin95] each aim at the verification of a particular variant of intransitive noninterference; the unwinding results in [Rya91] aim at verifying CSP-noninterference; and the unwinding result in [Mil94] aims at verifying forward correctability [JT88]. Zakinthinos suggested a more uniform approach and used this approach to derive unwinding theorems for generalized noninterference, forward correctability, the perfect security property, and generalized noninference. However, he still had to derive these theorems separately for each information flow property [Zak96].

Summarizing, our main improvements are:

No side conditions on the unwinding relation We do not require that unwinding relations must be reflexive, transitive, or symmetric. The unwinding techniques in this thesis improve on our earlier attempt, in which we showed that the symmetry requirement for unwinding relations is unnecessary [Man00d].

General applicability We present unwinding theorems (for BSPs) that can be exploited during the verification of all information flow properties represented in *MAKS*, including several properties for which previously no unwinding results were available. That is, we do not have to derive unwinding theorems for each individual property from scratch.

In addition to our results on unwinding, we also show that various forms of simulation techniques can be used in the verification of information flow properties. In other words, we propose verification techniques that provide alternatives to the unwinding approach. Simulation techniques like, e.g., forward simulation or backward simulation were originally developed in order to show that one specification refines another specification rather than for the verification of information flow properties.

1.4 Compositionality Results

In this thesis, we propose several compositionality results that provide a basis for a modular verification of large and complex systems. Rather than verifying directly that a complex system is "secure", one verifies that each of its components is "secure" (e.g. by unwinding) and then applies a compositionality result in order to conclude the "security" of the overall system from the "security" of the components.

The various information flow properties behave quite differently under composition. Some information flow properties are preserved under composition in general while others are only preserved if certain, sometimes quite restrictive, conditions are satisfied. An example of a novel compositionality result that we derive is: if two systems each satisfy weakened forward correctability then their composition also satisfies weakened forward correctability. Weakened forward correctability, i.e. our novel information flow property, is an improvement of

Johnson and Thayer's forward correctability [JT88]. This is very interesting because forward correctability itself is an improvement of McCullough's restrictiveness [McC87] (a quite complicated property), which historically was the first information flow property with a general compositionality result. The possibility to further improve this line of composable information flow properties had remained undetected for the last fifteen years.

However, our objective is not only to present a collection of novel compositionality results but rather to provide a basis for deriving such results in a uniform and systematic way. We reduce the problem of preserving a given property under composition to the problem of preserving all BSPs from which it is assembled. This is possible because BSPs are assembled by conjunction. To this end, we derive compositionality results for various BSPs from which compositionality results for more complex information flow properties can be obtained easily.

Previous work on compositionality of information flow properties often focused on one particular property only (like e.g. in [McC87, JT88, GN88, Mil90, BCC94, ZL97]). Moreover, if different information flow properties were considered then compositionality results were derived separately for each of these properties from scratch (like e.g. in [FG95, Zak96]). Prior to our work, the only positive exception to this was McLean's work on compositionality in the context of his framework of selective interleaving functions [McL94a, McL96]. However, several information flow properties for which we derive compositionality results in this thesis could not be investigated by McLean because of the limited expressiveness of his framework.

We also use *MAKS* to re-justify several already known compositionality results in order to illustrate that our uniform approach works rather generally. As a side effect of these re-justifications, we deepened our understanding of the known compositionality results. The insights that we gained also led to a classification of compositionality results. This classification groups compositionality results that *hold for the same reasons* into the same class. McLean [McL94a] grouped compositionality results depending on whether they *can be applied in similar settings*.[3] These two classifications complement each other: while we put more emphasis on understanding the reasons why the various results are valid, McLean puts more emphasis on explaining in which settings these results can be applied.

Summarizing, two main features of our approach to deriving compositionality results are:

Uniformity We derive compositionality results in a uniform way. Namely, we reduce the problem of preserving a given information flow property under composition to the problem of preserving all BSPs from which this property is assembled.

General applicability We present compositionality results for BSPs that can be used during the derivation of compositionality results for more complex information flow properties. That is, our approach is not targeted to only a single information flow property. Rather, it is applicable to *all* information flow properties that can be represented in *MAKS*. We demonstrate by numerous examples that it, indeed, provides a suitable basis for the derivation of a wide range of compositionality results.

1.5 Case Study

In order to illustrate how our results can be applied in a concrete setting, we perform a case study of considerable size and complexity. This case study is concerned with a rigorous

[3] More specifically, results are grouped together if they impose the same restriction on the composition operation. Possible restrictions are product, cascade, and general composition.

justification of Sabelfeld and Sands's approach to language-based security [SS00].

Sabelfeld and Sands proposed a security type system that can be used to check mechanically whether a given program is "secure". For this type system, they derived a soundness result, stating that every type-correct program satisfies the *strong security condition*. However, the strong security condition is somewhat non-standard and has been derived by Sabelfeld and Sands in an ad hoc fashion. In the case study, we show that every program that is strongly secure also satisfies a particular information flow property in *MAKS*. An immediate consequence of this implication is that the security type system is sound wrt. this information flow property. This is a more convincing soundness result than the original one because it relates the security type system to a security condition from a well established class of information flow properties.

The programming language that we consider is DMWL, an imperative language with features for multi-threaded as well as for distributed programming. This language is an extension of the language originally considered by Sabelfeld and Sands. We start the case study by specifying the behavior of DMWL programs in an event-based formalism. Rather than specifying the behavior of a particular program, we specify the behaviors of all DMWL programs. In other words, we specify the operational semantics of DMWL by an event system. This (generic) specification is the system component of our security model. We proceed in the case study by expressing the security requirements for DMWL programs with an information flow property. In this process, we show how a suitable information flow property can be constructed in a goal-directed and application-driven way with the help of *MAKS*. Interestingly, the information flow property at which we arrive is completely novel. With the help of our taxonomy of information flow properties, we obtain a basic understanding of this novel property and its relation to known properties. The specification of the information flow property constitutes the security component of our security model. Finally, we verify that this information flow property holds for all strongly secure programs. For this purpose, we firstly prove that the property holds for primitive system components (i.e. for individual, strongly secure processes) with the help of our unwinding techniques. Secondly, we prove that the information flow property holds for more complex systems (i.e. for distributed systems consisting of multiple, strongly secure processes) by exploiting a compositionality result that is derived using our uniform approach to deriving compositionality results.

Summarizing, the case study encompasses all areas to which we contribute in this thesis. Namely, we use *MAKS* to derive a suitable information flow property; we use the taxonomy in order to deepen our understanding of this property; we apply our unwinding techniques to verify this property for individual processes; and we exploit our approach to deriving compositionality results for verifying that the property holds for distributed programs.

1.6 Overview

After this introductory chapter and the description of preliminaries in Chapter 2, the thesis is structured as follows:

The presentation of our own contributions begins in Chapter 3 with the description of *MAKS*. We formally define all basic concepts of *MAKS* and propose a collection of BSPs. These BSPs will be used to assemble information flow properties throughout this thesis. We also derive a taxonomy of our BSPs that can be used to compare BSPs with each other.

In Chapter 4, we show how the information flow properties from the literature can be

represented with *MAKS* in a modular way. Thereby, we demonstrate that *MAKS* is expressive enough to serve as a basis for the investigation of information flow properties. Based on the modular representation, we clarify several issues about the represented properties and, in particular, present a taxonomy of known information flow properties.

Unwinding techniques are proposed in Chapter 5. We present unwinding conditions and unwinding theorems for our BSPs. We also show how these unwinding results for BSPs can be exploited during the verification of more complex information flow properties. This chapter is complemented by Appendix C in which we propose further verification techniques.

In Chapter 6, we show how compositionality results can be derived following a uniform approach. This approach builds on the modular representation of information flow properties in *MAKS*. We apply our approach in order to re-justify several already known compositionality results, which culminates in a classification of compositionality results. We also derive several novel compositionality results and propose a novel composable information flow property.

In Chapter 7, the applicability of the results presented in Chapters 3–6 is demonstrated in a concrete case study from the area of language-based security. In the case study, we show how to derive a suitable information flow property in *MAKS*, make use of our taxonomy of information flow properties, apply our unwinding techniques, derive a compositionality result following our uniform approach to derive such results, and then apply the compositionality result in order to verify the information flow property for complex distributed systems. Besides illustrating the applicability of our results, this case study is also interesting from the perspective of language-based security and it already inspired some improvements of the language-based techniques.

A summary of the contributions of this thesis, a brief description of ongoing work, and a list of interesting directions for future research is given in Chapter 8.

Chapter 2

Notions and Notation

In this chapter, we introduce basic notions and formal notation to be used in later chapters. In particular, we introduce a trace-based system model, namely the model of *event systems*, that will provide the common semantic basis of our investigations throughout this thesis. Event systems have been quite popular for the investigation of information flow properties (e.g. [McC87, GN88, JT88, Mil94, ZL97]) because they are suitable for modeling deterministic as well as nondeterministic systems but, nevertheless, are conceptually quite simple and easy to apply. A second system model, namely that of *state-event systems*, is introduced because its use will simplify the presentation in some parts of this thesis. Event systems and state-event systems are closely related. A state-event system can be obtained from an event system by enriching it with a notion of system state.

We will also present abstract definitions of classes of system properties, namely of *properties of traces* and of *properties of sets of traces*. The security properties that we investigate in this thesis, i.e. information flow properties, belong to the second class of properties. More specifically, they are *closure properties of sets of traces*. In order to illustrate the wide spectrum of information flow properties, we briefly review three concrete information flow properties from the literature, further information flow properties follow in later chapters.

Overview. Event systems and state-event systems are defined in Section 2.1. Section 2.2 describes how system properties can be specified. In Section 2.3, it is explained how system models, other than event systems or state-event systems, can be adopted.

2.1 System Model

2.1.1 Traces

The behavior of a system can often be adequately specified by a set of traces where each *trace* in this set models a possible execution sequence of the given system. Formally, traces are defined as sequences of events where an *event* models an atomic action like, e.g., sending a message, receiving a message, or performing a computation step. Every *occurrence of an event* in a trace models the occurrence of the respective atomic action.

Definition 2.1.1 (Trace). Let E be a set of events. A *trace over* E is a (possibly empty) finite sequence of events in E. ◇

In this thesis, we focus on *finite traces*. The set of all finite traces over some set E is denoted by E^*. For simplicity, we use "trace" as synonym for "finite trace" in the following.

As a notational convention, we separate adjacent events in a trace by dots and surround traces by angle brackets. For example, $\langle\rangle$ denotes the empty trace, $\langle e_1 \rangle$ denotes the trace in which a single event e_1 occurs, and $\langle e_1.e_2.e_3 \rangle$ denotes the trace in which e_1, e_2, e_3 occur (in this order). Concatenation of traces is also denoted by dots, i.e. $\langle e_1.e_2 \rangle.\langle e_3 \rangle = \langle e_1.e_2.e_3 \rangle$. We use the letters τ and t (possibly with indices or primes) to denote traces where the Greek letter τ is only used for traces that model possible behaviors of the system under consideration.

Example 2.1.2. Let $E_{\mathtt{RND}} = \{term\} \cup \{out(n) \mid n \in \mathbb{N}\}$ be a set of events where the event *term* models that the system terminates and an event $out(n)$ models that the natural number n is output. The behavior of a random generator that outputs a sequence of random natural numbers and then terminates can be modeled by the smallest set $Tr_{\mathtt{RND}} \subseteq {E_{\mathtt{RND}}}^*$ satisfying:

1. $\langle\rangle \in Tr_{\mathtt{RND}}$,
2. $\langle term \rangle \in Tr_{\mathtt{RND}}$, and
3. if $\tau \in Tr_{\mathtt{RND}}$ then $\langle out(n) \rangle.\tau \in Tr_{\mathtt{RND}}$.

The traces $\langle term \rangle$, $\langle out(42) \rangle$, $\langle out(42).out(13).term \rangle$, and $\langle out(1).out(2).out(3).out(4) \rangle$ are examples of possible traces in $Tr_{\mathtt{RND}}$. The last two of these traces are also illustrated in Figure 2.1 using a graphical notation where events are viewed as arrows.

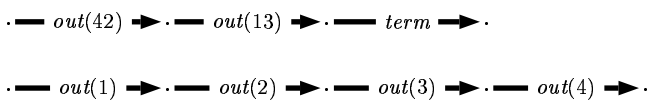


Figure 2.1: Some possible traces for the random generator (cf. Example 2.1.2)

Note that the trace $\langle out(1).out(2).out(3).out(4) \rangle$ models an incomplete run of the system (no occurrence of event *term*) while $\langle out(42).out(13).term \rangle$ models a complete run. Typically, a set of traces that models the possible behaviors of a system contains *all* traces that model possible behaviors (including the ones that model incomplete runs). Formally, this means that the set of traces is closed under prefixes. ◇

Remark 2.1.3. Specifying a system by the set of its possible traces can be seen as a *semantic approach*. There are various *syntactic* specification languages for specifying sets of traces, including process algebras like, e.g., CSP [Hoa85], and temporal logics like, e.g., TLA [Lam94]. The main advantage of basing our investigations on a semantic system model rather than a specific syntactic specification language is that our results become applicable to different specification languages (also cf. Section 2.3). ◇

2.1.2 Event Systems

Using event systems, a system is modeled not only in terms of its behavior (by a set of traces) but also in terms of its interface (i.e. by sets of input events and output events).

Definition 2.1.4 (Event system). An *event system* ES is a tuple (E, I, O, Tr) where E is a *set of events*, $I \subseteq E$, $O \subseteq E$ are the *sets of input events* and *output events*, respectively, and $Tr \subseteq E^*$ is the *set of possible traces*. I and O must be disjoint, i.e. $I \cap O = \emptyset$, and Tr must be closed under prefixes, i.e. every prefix of a trace in Tr must also be in Tr. ◇

Example 2.1.5. Let $E_{\texttt{RND}}$ and $Tr_{\texttt{RND}}$ be defined like in Example 2.1.2. Moreover, let $I_{\texttt{RND}} = \emptyset$ and $O_{\texttt{RND}} = \{out(n) \mid n \in \mathbb{N}\}$. The event system $\texttt{RND} = (E_{\texttt{RND}}, I_{\texttt{RND}}, O_{\texttt{RND}}, Tr_{\texttt{RND}})$ provides a more detailed specification of the random generator than the one in Example 2.1.2. RND makes explicit that *term* is an internal event ($term \notin I \cup O$) and $out(n)$ is an output event. ◇

Example 2.1.5 illustrates how nondeterministic systems can be modeled by event systems. After a trace τ (without occurrences of event *term*) has occurred, any of the events $out(0)$, $out(1)$, $out(2)$, ... can occur. That is, nondeterministic behavior is reflected in the specification by the possibility that different (output) events can occur.

Let us introduce a few more notions for traces that we need at various points of this thesis.

Definition 2.1.6 (Totality in a set of events). $ES = (E, I, O, Tr)$ is *total* in a subset E' of E, denoted by $total(ES, E')$, iff[1] $\tau.\langle e\rangle \in Tr$ holds for all $\tau \in Tr$ and all $e \in E'$. ◇

Definition 2.1.7 (Input totality). $ES = (E, I, O, Tr)$ is *input total* iff $total(ES, I)$. ◇

Definition 2.1.8 (Projection). The *projection* $t|_{E'}$ of a trace $t \in E^*$ to a subset E' of E results from t by deleting all events *not* in E', i.e. $\langle\rangle|_{E'} = \langle\rangle$, if $e \notin E'$ then $(t.\langle e\rangle)|_{E'} = t|_{E'}$, and if $e \in E'$ then $(t.\langle e\rangle)|_{E'} = t|_{E'}.\langle e\rangle$. ◇

Example 2.1.9. The projections of $t = \langle out(1).out(2).out(3).out(4)\rangle$ to output events that involve, respectively, only odd or only even numbers are

$$\begin{aligned} t|_{\{out(n)\,|\,n \bmod 2=1\}} &= \langle out(1).out(3)\rangle \\ t|_{\{out(n)\,|\,n \bmod 2=0\}} &= \langle out(2).out(4)\rangle \end{aligned}$$ ◇

Definition 2.1.10 (Interleaving). The set of all *interleavings* of two traces $t_1, t_2 \in E^*$ is defined inductively by

$$\begin{aligned} interleaving(t_1, \langle\rangle) &= \{t_1\} \\ interleaving(\langle\rangle, t_2) &= \{t_2\} \\ interleaving(\langle e_1\rangle.t_1, \langle e_2\rangle.t_2) &= \{\langle e_1\rangle.t' \mid t' \in interleaving(t_1, \langle e_2\rangle.t_2)\} \\ &\quad \cup \{\langle e_2\rangle.t' \mid t' \in interleaving(\langle e_1\rangle.t_1, t_2)\} \end{aligned}$$ ◇

Example 2.1.11. Let $t_1 = \langle out(1).out(3)\rangle$ and $t_2 = \langle out(2).out(4)\rangle$. There are 6 possible interleavings of t_1 and t_2. In each of these interleavings, $out(1)$ occurs before $out(3)$ and $out(2)$ occurs before $out(4)$. For example, $\langle out(1).out(2).out(3).out(4)\rangle \in interleaving(t_1, t_2)$, but $\langle out(4).out(3).out(2).out(1)\rangle \notin interleaving(t_1, t_2)$. ◇

Remark 2.1.12. Historically, event systems can be regarded as a descendent of the trace-based semantics of Hoare's CSP [Hoa85]. According to these semantics, every process corresponds semantically to a pair consisting of an *alphabet*, i.e. a set of events, and a set of possible *traces*. McCullough further refined this trace-based model by distinguishing *input events* and *output events* from *internal events* [McC87]. The intuition behind this distinction was that *input events* are controlled by the environment while *output events as well as internal events* are controlled by the system. The sets of input events and output events also specify the *interface* that shall be used for interacting with the system.

In many publications, the definition of event systems incorporates the assumption of *input totality* (cf. e.g. [JT88, ZL97]). We refrain from making input totality a *general* assumption

[1] As usual, we abbreviate "if and only if" by "iff".

because this is often an unnecessary restriction. According to Definition 2.1.4, an event system may be, but need not to be, input total. Consequently, all results that we derive for our definition of event systems are also valid for event systems that are input total. ◇

2.1.3 State-Event Systems

State-event systems result from enriching event systems with a notion of *state*. Using states, the set of possible traces can be specified in an inductive manner by a transition relation.

Definition 2.1.13 (State-event system). A *state-event system*[2] SES is a tuple (S, s_0, E, I, O, T) where S is a *set of states*[3], $s_0 \in S$ is the *initial state*, E is a *set of events*, $I, O \subseteq E$ are the *sets of input events* and *output events*, respectively, and $T \subseteq S \times E \times S$ is the *transition relation*. I and O must be disjoint and T must correspond to a (partial) function of type $S \times E \hookrightarrow S$, i.e. for all $s \in S$ and $e \in E$, there is at most one $s' \in S$ with $(s, e, s') \in T$. ◇

Despite our assumptions (i.e. that there is only one initial state s_0 and that for every $s \in S$ and every $e \in E$ there is at most one $s' \in S$ with $(s, e, s') \in T$), state-event systems are suitable for the specification of nondeterministic systems. The reason for this is that the output is *not* completely determined by the state: if a given system is in some state s then different sequences of output events might be possible. Hence, nondeterminism arises from the choice between different enabled events (rather than from a nondeterministic choice of the state after an event has occurred). This is illustrated by the following example.

Example 2.1.14. Let $E_{\mathtt{RND}}$, $I_{\mathtt{RND}}$, $O_{\mathtt{RND}}$ and let $Tr_{\mathtt{RND}}$ be defined like in Example 2.1.5. Moreover, let $S_{\mathtt{RND}} = \{s_0, s_t\}$ be a set of states and $T_{\mathtt{RND}} = \{(s_0, out(n), s_0) \mid n \in \mathbb{N}\} \cup \{(s_0, term, s_t)\}$ be a transition relation. The random generator can be specified by the state-event system $SES_{\mathtt{RND}} = (S_{\mathtt{RND}}, s_0, E_{\mathtt{RND}}, I_{\mathtt{RND}}, O_{\mathtt{RND}}, T_{\mathtt{RND}})$. Note that all events in E are enabled in the initial state s_0. The state remains unaffected if a natural number is output in state s_0. After an occurrence of the event *term*, the system moves to the terminal state s_t, in which no events are enabled. This means that the same traces are possible for $SES_{\mathtt{RND}}$ and for RND. ◇

Let us introduce a few further notions and some notation for state-event systems.

Instead of $(s, e, s') \in T$ we sometimes use the notation $s \xrightarrow{e}_T s'$ where $s, s' \in S$ are states and $e \in E$ is an event. If the transition relation T is obvious from the context then we omit the index and write $s \xrightarrow{e} s'$ instead of $s \xrightarrow{e}_T s'$. To abbreviate sequences of transitions, we use the notation $s \overset{t}{\Longrightarrow}_T s'$ (or $s \overset{t}{\Longrightarrow} s'$ if T is obvious) where $t \in E^*$ is a trace. For a state event system $SES = (S, s_0, E, I, O, T)$, the relation $\overset{t}{\Longrightarrow}$ is defined by:

$$s \overset{\langle\rangle}{\Longrightarrow} s' \quad \text{, if } s = s'$$
$$s \overset{\langle e \rangle . t}{\Longrightarrow} s' \quad \text{, if } \exists s'' \in S.\, (s \xrightarrow{e} s'' \wedge s'' \overset{t}{\Longrightarrow} s')$$

Definition 2.1.15 (Reachable). A state $s \in S$ is *reachable* for $SES = (S, s_0, E, I, O, T)$, denoted by $reachable(SES, s)$, if there is a trace $t \in E^*$ such that $s_0 \overset{t}{\Longrightarrow} s$ holds. ◇

[2] We use the term "state-event system" rather than the term "state machine" (e.g. used in [McC90]). The term "state machine", *in this context*, often leads to misunderstandings because it emphasizes states over events despite the fact that communication with the environment is assumed to occur by synchronization on the occurrence of shared events rather than by sharing part of the state space (cf. Definition 7.3.6).

[3] It is possible to view states as mappings from state variables to values. However, also other notions are possible because the notion of state is left transparent in Definition 2.1.13.

Definition 2.1.16 (Enabled). For $SES = (S, s_0, E, I, O, T)$, a trace $t \in E^*$ is *enabled* in a state $s \in S$, denoted by $enabled(SES, s, t)$, if there is a state $s' \in S$ such that $s \stackrel{t}{\Longrightarrow} s'$ holds. $\diamond$

Definition 2.1.17 (Possible trace). For $SES = (S, s_0, E, I, O, T)$, a trace $t \in E^*$ is *possible* if t is enabled in s_0. The set of all possible traces for SES is denoted by Tr_{SES}. $\diamond$

Definition 2.1.18 (Induced event system). The *event system* ES_{SES} that is *induced* by the *state-event system* $SES = (S, s_0, E, I, O, T)$ is defined by $ES_{SES} = (E, I, O, Tr_{SES})$. $\diamond$

Remark 2.1.19. It is easy to check that the set of traces $Tr_{SES_{\mathtt{RND}}}$ generated by the state-event system $SES_{\mathtt{RND}}$ from Example 2.1.14 is identical to the set $Tr_{\mathtt{RND}}$ in Example 2.1.5. Recall that, in Example 2.1.5, $Tr_{\mathtt{RND}}$ had to be specified by a meta-level induction (in the text). In contrast to this, state-event systems incorporate a transition relation that can be used for an inductive specification of the set of possible traces on the object level. $\diamond$

Remark 2.1.20. Many syntactic specification formalisms, including most programming languages, are based on a state-transition model. Hence, one often obtains the information necessary for defining a state-event system (rather than an event system) for free. $\diamond$

2.2 Specifying System Properties

While a system specification expresses *how* a system behaves, the specification of system properties expresses *what* the system's requirements are. In order to formally verify that a system satisfies its requirements, it is necessary to specify system properties in precise terms and to state formally what it means for a system specification to satisfy a given system property (also cf. Figure 2.2).

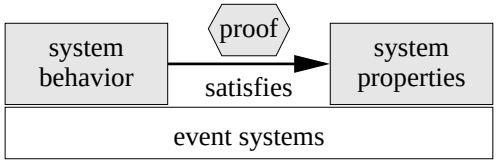


Figure 2.2: Structure of a formal specification

In this section, we present abstract definitions of classes of system properties and define corresponding satisfaction relations. Concrete property specifications are special instances of these abstract definitions, just like concrete system specifications are special instances of the abstract definition of event systems.

2.2.1 Properties of Traces

System requirements can often be specified by properties of traces, i.e. properties that are satisfied by an event system if they are satisfied by every possible trace of the event system. Examples for properties of traces are, e.g., that a particular invariant holds during program execution or that a program will eventually terminate.

Definition 2.2.1 (Property of traces). A *property of traces* over a set E of events is a predicate $P : E^* \to Bool$.[4] An event system $ES = (E, I, O, Tr)$ *satisfies* $P : E^* \to Bool$ iff $P(\tau)$ holds for all $\tau \in Tr$. $\diamond$

Example 2.2.2. For the random generator from Example 2.1.5, no events are enabled after an occurrence of *term*. This statement can be formalized by the property of traces P_{term} defined by: $P_{term}(\tau) \Leftrightarrow \forall \alpha, \beta \in E^*.(\tau = (\beta.\langle term \rangle.\alpha) \Rightarrow \alpha = \langle\rangle)$. The event system $ES_{\texttt{RND}}$ defined in Example 2.1.5 fulfills P_{term}. This is shown by induction on the length of possible traces and a case distinction according to the definition of $Tr_{\texttt{RND}}$: In the base case P_{term} holds for the traces $\langle\rangle$ and $\langle term \rangle$; in the step case, $P_{term}(\tau)$ obviously implies $P_{term}(\langle out(n) \rangle.\tau)$. $\diamond$

Remark 2.2.3. The property P_{term} is an example of a *safety property* because it ensures that *nothing bad happens*, i.e. after an occurrence of the event *term* no other events will occur. This means that the system *has terminated* after an occurrence of *term* rather than that it *will eventually terminate*. Properties of the latter kind are different from safety properties because they ensure that *eventually something good will happen*. Properties of this kind are referred to as *liveness properties*.[5] $\diamond$

2.2.2 Properties of Sets of Traces

There are system requirements that cannot be specified by properties of traces. Let us consider the following simple example: Assume a system that takes a request as input, processes the request, outputs a response, and then terminates. A critical requirement of such a system might be that the *average response time* is below a certain time limit t. Obviously, the validity of this requirement depends on the set of *all* possible system executions. Therefore, this requirement is no property of traces. Rather it is a *property of sets of traces*.

Definition 2.2.4 (Property of sets of traces). A *property of sets of traces* over a set E of events is a predicate $Q : \mathcal{P}(E^*) \to Bool$.[6] An event system $ES = (E, I, O, Tr)$ *satisfies a property* $Q : \mathcal{P}(E^*) \to Bool$ *of sets of traces* iff $Q(Tr)$ holds. $\diamond$

The class of properties of sets of traces subsumes the class of properties of traces in the sense that each property of traces can be expressed by an equivalent property of sets of traces.[7] Moreover, there are properties of sets of traces for which there is no equivalent property of traces. As an interesting side note, the following remark elaborates in more detail which properties of sets of traces can be expressed by equivalent properties of traces.

---

[4] $Bool = \{True, False\}$ is the set of truth values. A predicate $P : E^* \to Bool$ holds for $t \in E^*$ iff $P(t) = True$.

[5] Interestingly, safety and liveness properties are *fundamental* for properties of traces because every property of traces can be expressed as the conjunction of a safety and a liveness property. This was shown in [AS85] for infinite traces that are state-based. The two classes of properties of traces can be characterized as follows: If a safety property holds for every finite prefix of a given infinite trace then it must also hold for the complete trace and for every finite trace t there must be an infinite trace with prefix t for which the liveness property holds. In particular, this means that liveness properties cannot distinguish between different *finite* traces. Hence, they cannot be investigated in a system model that permits only finite traces (like our definition of event systems). However, this does not matter because we are interested in a different class of system properties.

[6] As usual, $\mathcal{P}$ denotes the powerset construction, i.e. $\mathcal{P}(E^*)$ denotes the set of all subsets of E^*.

[7] Every property of traces $P : E^* \to Bool$ induces a property of sets of traces $Q_P : \mathcal{P}(E^*) \to Bool$ defined by $Q_P(Tr) = \forall \tau \in Tr.\, P(\tau)$. The property Q_P is equivalent to P in the sense that Q_P is satisfied for a given event system iff P is satisfied for that event system.

Remark 2.2.5 (Relating properties of sets of traces and properties of traces). Let us investigate more closely, which properties of sets of traces are induced by properties of traces. The characteristic set Γ_P of a property of traces P is defined by $\Gamma_P = \{\tau \in E^* \mid P(\tau)\}$. The characteristic set Γ_Q of a property of sets of traces Q is defined by $\Gamma_Q = \{Tr \subseteq E^* \mid Q(Tr)\}$. For each property of traces P, the characteristic set Γ_{Q_P} of the property of sets of traces Q_P (i.e. the property induced by P) is the set of all subsets of Γ_P, i.e. $\Gamma_{Q_P} = \{Tr \mid Tr \subseteq \Gamma_P\}$. Clearly, Γ_{Q_P} is closed under subsets and has a maximal element (namely Γ_P). Likewise, every property of sets of traces Q with a characteristic set that is closed under subsets and that contains a maximal element induces an equivalent property of traces P_Q. The characteristic set Γ_{P_Q} is simply the maximal element of Γ_Q. Consequently, the set of all properties of traces is isomorphic to the set of all properties of sets of traces with a characteristic set that is closed under subsets and that contains a maximal element. ◇

The following remark illustrates that some properties of sets of traces that cannot be expressed by *equivalent* properties of traces can at least be conservatively approximated.

Remark 2.2.6 (Approximating properties of sets of traces). According to Remark 2.2.5, properties of sets of traces with a characteristic set that is *not* closed under subsets or that has *no* maximal element cannot be expressed by an *equivalent* property of traces. Nevertheless, some of these properties can be *approximated* by properties of traces. For example, the average-response-time property (a property of sets of traces) can be approximated by a worst-response-time property (a property of traces) in the sense that if the worst response time of a system is below a time limit t then the average response time is also below this time limit. Hence, the worst-response-time property is a *conservative* approximation of the average-response-time property. ◇

Almost all properties of sets of traces can be conservatively approximated by properties of traces.[8] However, the approximation often results in such a restrictive requirement that it is not appropriate for replacing the original property.[9] In particular, the *information flow properties* that we investigate in this thesis *are closure properties of sets of traces* [McL94a] and, hence, no good approximations by properties of traces are known for them.[10]

2.2.3 Closure Properties of Sets of Traces

Intuitively, a closure property requires for a given set that whenever certain elements are members of the set then certain other elements also must be members of the set.

Definition 2.2.7 (Closure property of sets of traces). A property of sets of traces $Q : \mathcal{P}(E^*) \to \mathit{Bool}$ is a *closure property of sets of traces* iff for all $Tr \subseteq E^*$ there is a set $\overline{Tr} \subseteq E^*$ with $\overline{Tr} \supseteq Tr$ and $Q(\overline{Tr})$. ◇

[8] The property $\mathit{INCONS} : E \to \mathit{Bool}$ defined by $\forall \tau \in E^*. \mathit{INCONS}(\tau) \Leftrightarrow \mathit{False}$, i.e. the property that does not hold for any trace, is a conservative approximation of all properties of sets of traces except for the property $Q : \mathcal{P}(E^*) \to \mathit{Bool}$ defined by $\forall Tr \in \mathcal{P}(E^*). Q(Tr) \Leftrightarrow \mathit{False}$. The reason why Q cannot be approximated is that $Q(\emptyset)$ does not hold while every property of sets of traces induced by a property of traces holds for $\emptyset$.

[9] The inappropriateness of approximating a given property of sets of traces becomes most obvious if *INCONS* is the only possible conservative approximation. However, even if there are approximations different from *INCONS* then using the approximations is often inappropriate. For example, one usually wants to enforce tighter bounds on the average response time than on the worst response time.

[10] Access control provides a means to *implement* the security requirements expressed by information flow properties. As access control models correspond to safety properties [Sch00a], they are, in general, not suitable alternatives for expressing the *abstract* security requirements.

Note that for each set of traces there is a superset for which a given closure property holds. In other words, a closure property can always be made true by adding elements to a given set of traces. Let us consider a simple example of a closure property of sets of traces.

Example 2.2.8. The random generator behaves chaotically until the event *term* has occurred. This can be formalized by the following closure property of sets of traces:

$$Q_{chaotic}(Tr) \Leftrightarrow \forall \tau \in Tr.\, (\tau|_{term} = \langle\rangle \Rightarrow \forall e \in E.\, \tau.\langle e\rangle \in Tr)$$

The property $Q_{chaotic}$ requires: If τ is in Tr and does not contain occurrences of *term* then for all $e \in E$, the trace $\tau.\langle e\rangle$ must also be in Tr. For instance, $Q_{chaotic}(Tr_{\mathtt{RND}})$ holds (for system RND from Example 2.1.5) because every event in $E_{\mathtt{RND}}$ can occur until *term* has occurred. ◇

2.2.4 Specifying Secure Information Flow

Now we are ready to give a more formal introduction to information flow properties than in Chapter 1. To this end, we present the specification of a simple example system and then illustrate using three different information flow properties from the literature how a formal analysis reveals that this system is insecure.

Example 2.2.9. Consider a system that receives confidential input from some trusted user (the so called *high-level user*) and that also outputs data to some untrusted user (the so called *low-level user*) who should not obtain the confidential information. We model that the high-level user inputs a secret number $n \in \mathbb{N}$ by the event hi_n and model that a number $n' \in \mathbb{N}$ is output to the low-level user by $lo_{n'}$. The sets of all high-level events and all low-level events are denoted by H and L, respectively (i.e. $H = \{hi_n \mid n \in \mathbb{N}\}$ and $L = \{lo_n \mid n \in \mathbb{N}\}$).

Define the event system $\mathtt{LEAK} = (E_{\mathtt{LEAK}}, I_{\mathtt{LEAK}}, O_{\mathtt{LEAK}}, Tr_{\mathtt{LEAK}})$ where $I_{\mathtt{LEAK}} = \{hi_n \mid n \in \mathbb{N}\}$, $O_{\mathtt{LEAK}} = \{lo_n \mid n \in \mathbb{N}\}$, $E_{\mathtt{LEAK}} = I_{\mathtt{LEAK}} \cup O_{\mathtt{LEAK}}$, and $Tr_{\mathtt{LEAK}} \subseteq E_{\mathtt{LEAK}}{}^*$ is the smallest set with:

1. $\langle\rangle \in Tr_{\mathtt{LEAK}}$,
2. if $n \in \mathbb{N}$ then $\langle hi_n\rangle \in Tr_{\mathtt{LEAK}}$, and
3. if $n \in \mathbb{N}$ and $\tau \in Tr_{\mathtt{LEAK}}$ then $\langle hi_n.lo_n\rangle.\tau \in Tr_{\mathtt{LEAK}}$.

According to this specification, LEAK receives a number n as high-level input (event hi_n) and immediately outputs this number to the low-level (event lo_n). This means that if the low-level user receives, e.g., the sequence $\langle 42\rangle.\langle 13\rangle$ from the system then he can deduce that the high-level user has previously input $\langle 42\rangle.\langle 13\rangle$. Clearly, this is a security violation.

More formally, the low-level user observes the sequence $\langle lo_{42}.lo_{13}\rangle$ and deduces from this observation and his knowledge of the system specification[11] that some trace in

$$\{\langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle.\tau \mid \tau = \langle\rangle \vee \exists n \in \mathbb{N}.\, \tau = \langle hi_n\rangle\} \tag{2.1}$$

must have occurred (cf. Figure 2.3). All traces in this set have a common prefix, namely $\langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle$ and, thus, the low-level user can deduce that $\langle 42\rangle.\langle 13\rangle$ has been input by the high-level user. ◇

[11] In investigations of secure information flow, it is commonly assumed that untrusted users might have complete knowledge of the system specification. This is a worst case assumption.

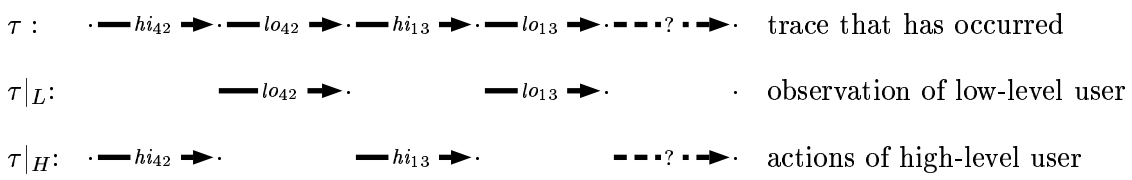


Figure 2.3: A trace and its projections to low-level and high-level events (cf. Example 2.2.9)

The problem with system LEAK is that the low-level user can narrow down the traces that could possibly have generated his observation $\langle lo_{42}.lo_{13}\rangle$ to a set of traces that is small enough such that he can infer confidential information. In other words, the problem with system LEAK is that the set of possible traces $Tr_{\texttt{LEAK}}$ is too small. If, e.g., the trace $\langle lo_{42}.lo_{13}\rangle$ (modeling that $\langle 42\rangle.\langle 13\rangle$ is output to the low-level user although no input has been provided by the high-level user) also would be possible then the low-level user could not tell from his observation $\langle lo_{42}.lo_{13}\rangle$ whether some trace in the set (2.1) or the trace $\langle lo_{42}.lo_{13}\rangle$ has actually occurred. This means, he would not know if the sequence that he obtains corresponds to confidential information from the high-level user or if it is some arbitrary sequence that has been invented by the system. Hence, there is no security violation.

More generally, a given system can be made "secure" by adding possible behaviors. In other words "security" is a closure property of sets of traces (also cf. Definition 2.2.7).

Let us now briefly review three information flow properties from the literature, namely noninference, separability, and generalized noninterference. For these properties, we will illustrate with system LEAK how insecurities can be detected in a formal analysis. Let us first provide the formal definitions of these properties.

Noninference [O'H90, McL94a, ZL97] is formally defined as follows:

$$NF(ES) \quad \equiv \quad \forall \tau \in Tr.\, \tau|_L \in Tr$$

This means, noninference holds for a given event system ES iff the projection of each possible trace to the set of low-level events also is a possible trace.

Separability [McL94a, ZL97] is formally defined as follows:

$$SEP(ES) \quad \equiv \quad \forall \tau_l, \tau_h \in Tr.\, interleaving(\tau_h|_H, \tau_l|_L) \subseteq Tr$$

That is, separability holds iff each interleaving of the high-level sequence in one possible trace with the low-level sequence in another possible trace is also a possible trace.

Generalized noninterference [McC87] is formally defined as follows:

$$\begin{aligned} GNI(ES) \quad \equiv \quad & \forall t_1, t_2, t_3 \in E^*. \\ & ((t_1.t_2 \in Tr \wedge t_3|_{E\setminus(H\cap I)} = t_2|_{E\setminus(H\cap I)}) \\ & \quad \Rightarrow \exists t_4 \in E^*.\, (t_1.t_4 \in Tr \wedge t_4|_{L\cup(H\cap I)} = t_3|_{L\cup(H\cap I)})) \end{aligned}$$

Hence, generalized noninterference holds iff for every trace $t_1.t_3$ that differs from a possible trace $t_1.t_2$ only in occurrences of high-level input events there is a possible trace $t_1.t_4$ that differs from $t_1.t_3$ only in occurrences of high-level internal events and high-level output events.

Let us now illustrate how system LEAK can be analyzed with these properties.

Example 2.2.10. *NF*(LEAK) does not hold. This is because $\tau = \langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle$ is a possible trace but its projection to L, i.e. $\tau|_L = \langle lo_{42}.lo_{13}\rangle$ is not a possible trace. ◇

Example 2.2.11. *SEP*(LEAK) does not hold. This is because $\tau_l = \langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle$ and $\tau_h = \langle\rangle$ are possible traces but $\langle lo_{42}.lo_{13}\rangle \in \mathit{interleaving}(\tau_h|_H, \tau_l|_L)$ is not a possible trace. ◇

Example 2.2.12. Let $t_1 = \langle\rangle$, $t_2 = \langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle$, and $t_3 = \langle lo_{42}.lo_{13}\rangle$. The trace $t_1.t_2$ is a possible trace of LEAK and t_3 differs from t_2 only in occurrences of high-level input events. There are no high-level internal events or high-level output events and, hence, the only trace that differs from t_3 at most in high-level internal events or high-level output events is $t_4 = t_3$. For these choices, generalized noninterference requires that $t_1.t_4$ is a possible trace. However, this is not the case and, consequently, *GNI*(LEAK) does not hold. ◇

Examples 2.2.10–2.2.12 show that noninference, separability, and generalized noninterference all detect the insecurity of the system LEAK in Example 2.2.9, i.e. neither *NF*(LEAK) nor *SEP*(LEAK) nor *GNI*(LEAK) holds. Hence, wrt. this particular example, the three information flow properties are equivalent in the sense that they yield the same result. However, in general, these three information flow properties are *not* equivalent. A deeper understanding of information flow properties (including *NF*, *SEP*, *GNI*, and many others) as well as an identification of their precise differences and similarities is one of the main objectives of this thesis. This objective will be pursued in Chapter 4.

2.3 Using Other Specification Formalisms

The model of event systems provides the common semantic basis of our investigation, however, this does not exclude the specification of a system's behavior with other (semantic) system models or (syntactic) specification formalisms. The only condition is that there must be a mapping from that system model or specification formalism to event systems such that each specification is mapped to a unique event system. We say that a given system property is satisfied by such a specification if it is satisfied by the induced event system.

Example 2.3.1. Let $\mathit{SES} = (S, s_0, E, I, O, T)$ be a state-event system, $P : E^* \to \mathit{Bool}$ be a property of traces, and $Q : \mathcal{P}(E^*) \to \mathit{Bool}$ be a property of sets of traces. The property P is satisfied by *SES* iff P is satisfied by ES_{SES} (formally: $\forall \tau \in Tr_{SES}.\, P(\tau)$). The property Q is satisfied by *SES* iff Q is satisfied by ES_{SES} (formally: $Q(Tr_{SES})$). ◇

In Chapter 5, we shall adopt the model of state-event systems when we elaborate verification techniques for information flow properties. Moreover, we will use state-event systems in our case study in Chapter 7 where we also introduce a convenient syntax for specifying concrete state-event systems.

2.4 Summary

In this chapter, we have introduced basic notions and some notation and we are now ready to move to the first central chapter of this thesis: the introduction of the *MAKS* framework.

Chapter 3

MAKS: A Modular Framework for Information Flow Properties

3.1 Introduction

The overwhelming variety of known information flow properties (e.g. [Sut86, McC87, GN88, JT88, O'H90, WJ90, McL94a, FG95, ZL97, Sch01]) and the rather subtle technical differences in their respective formal definitions makes it difficult to achieve a thorough understanding of all these properties. Hence, there is a need to analyze these properties, to compare them to each other, and to classify them wrt. their applicability.

In this chapter, we propose *MAKS*, the *Modular Assembly Kit for Security Properties*. *MAKS* provides a framework in which information flow properties can be represented in a uniform way, which is helpful in understanding the differences and similarities of the various properties. Besides simplifying comparisons, it also provides a suitable basis for other investigations of information flow properties. Examples are the derivation of unwinding theorems and of compositionality results to be presented in later chapters of this thesis.

In *MAKS*, an information flow property is defined by two elements: a set of views and a security predicate. The set of views defines *where* the flow of information is restricted. This means, a view demands that some set of confidential events (modeling the actions involving confidential information) does not interfere with some other set of visible events (modeling the observations by a potential attacker). The security predicate defines formally *what* noninterference means in this context. This means, an information flow property is satisfied by a given system if the security predicate holds for every view in the set.

The main novelty of our framework is its *modular structure*. Security predicates are assembled from simpler basic security predicates (abbreviated by BSPs in the following). These BSPs are *primitive security predicates* that constitute the building blocks of our assembly kit *MAKS*. A key step in the development of this modular representation was the observation that, for some information flow properties, noninterference means that an observer cannot deduce that confidential events have occurred while, for other properties, it also means that he cannot deduce that confidential events have *not* occurred. Separating these two aspects helped us to elaborate further differences between known information flow properties. Ultimately, our investigation led to the identification of "basic ingredients" of known information flow properties, i.e. the BSPs. These BSPs can be used to assemble known properties. However, they can also be used to assemble new information flow properties.

Overall, this results in security models with the structure depicted in Figure 3.1. Note that this structure is a specialization of the general structure of a security model (cf. Figure 1.1).

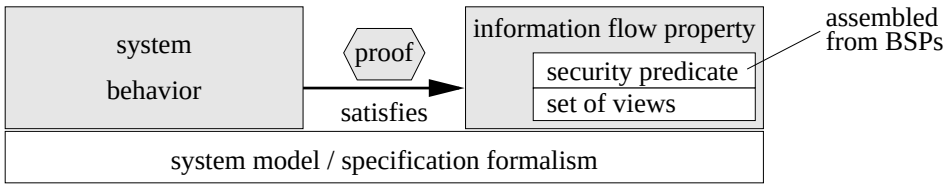


Figure 3.1: Structure of a security model based on *MAKS*

The representation of an information flow property in *MAKS* makes it very clear *where* the property requires noninterference (expressed by the set of views) and *what* the underlying definition of noninterference is (expressed by the security predicate). Moreover, the modular representation of security predicates allows one to reduce reasoning about complex information flow properties to reasoning about the much simpler BSPs.

Overview. In Section 3.2, the concepts for representing information flow properties in *MAKS* are defined, including views and security predicates. In Section 3.3, the notion of flow policies is introduced as a means to specify sets of views in an intuitive way based on a graphical notation. A collection of BSPs from which security predicates can be assembled is proposed in Section 3.4. In Section 3.5, these BSPs are compared to each other, and the main results of this chapter are summarized in Section 3.6.

Notational Conventions. E denotes a set of events, I and O are disjoint subsets of E, and $Tr \subseteq E^*$ is a set of traces. Let $ES = (E, I, O, Tr)$ be the resulting event system.

3.2 Assembling Information Flow Properties

Security requirements like confidentiality or integrity can be expressed as restrictions on the information flow within a system. As stated before, an information flow property is defined in *MAKS* by two elements: a set of views and a security predicate. While the former specifies *where* the flow of information is restricted, the latter defines *what* these restrictions mean.

3.2.1 Views

A *view* specifies concrete restrictions on the permitted flow of information within a system.

Definition 3.2.1 (View). A *view* $\mathcal{V} = (V, N, C)$ *in* E is a triple such that V, N, C forms a disjoint partition of E. ◇

Intuitively, a view describes the perspective of a (potentially malicious) observer of the system. For a given view $\mathcal{V} = (V, N, C)$, the set V specifies the events that are *visible* for the observer. For example, a user can observe the input that he provides to a system (i.e. occurrences of all input events he can engage in) and the output that he obtains from the system (i.e. occurrences of all output events he can engage in). Internal events might also be observable, e.g., if the user is located within the system (as an insider) or if he has some possibility to circumvent the system's interface. This means, all events in which an observer can engage are visible

to him. Events in V can be directly observed when they occur. Occurrences of all other events (i.e. events in $N \cup C$) are *not* directly observable. Hence, if a trace $\tau \in Tr$ occurs then the observer only sees the projection $\tau|_V$. The set C specifies the set of events that are *confidential* for the observer, i.e. the observer must not be able to deduce if events in C have or have not occurred (based on his observations and other knowledge he might have about the system).[1] Occurrences of all other events (i.e. events in $V \cup N$) are *not* confidential. Since the sets V, N, C are a disjoint partition of E, a view is completely determined by the sets V and C. The remaining events are collected in the set N. Intuitively, these events are *neither* visible *nor* confidential for the observer. The sets V, N, C cover all sensible combinations of visibility and confidentiality because, if some event were visible as well as confidential then this would result in an immediate security breach (cf. Figure 3.2).

	e invisible for observer	e visible for observer
e confidential for observer	$e \in C$	—
e not confidential for observer	$e \in N$	$e \in V$

Figure 3.2: Relating events wrt. confidentiality and visibility

Example 3.2.2. Let us investigate a two-level security policy where a low-level user must not be able to obtain high-level information. We divide the events of ES into two disjoint sets $L, H \subseteq E$ (L for low level, H for high level) such that L contains the events that are visible for a given observer when they occur and H contains events that are invisible for him. The view $\mathcal{H}_L = (L, \emptyset, H)$ intuitively specifies the following security requirement: no high-level information must flow to the low-level user under the assumption that occurrences of events in L are visible and occurrences of events in H are invisible for this user. ◇

The security requirements specified by a view are specific for a particular event system (more precisely: for its set of events). A graphical notation for specifying restrictions on the information flow in a more abstract and generic way will be introduced in Section 3.3.

3.2.2 Security Predicates

A *security predicate* defines what it means for the restrictions of a given view $\mathcal{V} = (V, N, C)$ to be satisfied, i.e. under which conditions can we say that no information about occurrences or nonoccurrences of events in C can be deduced from observations of events in V. Hence, a security predicate can be understood as a definition of what critical information flow means: there is critical information flow for a view if the security predicate does not hold for this view and there is no critical information flow otherwise.

In *MAKS*, security predicates are specified in a modular fashion by assembling them from *basic security predicates* where a basic security predicate defines a closure property on sets of traces. This modular representation of security predicates is one of the key novelties of *MAKS* that distinguishes it from all other frameworks for information flow properties.

Definition 3.2.3 (Basic security predicate). A *basic security predicate* BSPs is a property of sets of traces that is parametric in a view. Moreover, for every set E of events and every view $\mathcal{V}$ in E, $\mathrm{BSP}_{\mathcal{V}} : \mathcal{P}(E^*) \to \mathit{Bool}$ must be a closure property. ◇

[1] As usual in investigations of secure information flow, we assume that the observer has complete knowledge of the system specification. This is a worst-case assumption.

This definition of basic security predicates is somewhat abstract. A collection of concrete basic security predicates will be introduced in Section 3.4.

Definition 3.2.4 (Security predicate). A *security predicate SP* is defined by a set $\{\mathrm{BSP}^j \mid j \in J\}$ of basic security predicates where J is a nonempty index set. A security predicate SP holds for a view $\mathcal{V}$ in a set E of events and a set of traces $Tr \subseteq E^*$ (denoted by $SP_\mathcal{V}(Tr)$) iff $\mathrm{BSP}^j_\mathcal{V}(Tr)$ holds for all $j \in J$. ◇

Note that the BSPs from which a security predicate is assembled are conjoined, i.e. a given security predicate holds if all BSPs hold from which it is assembled. Therefore, we also use the notation $\bigwedge_{j \in J} \mathrm{BSP}^j$ instead of $\{\mathrm{BSP}^j \mid j \in J\}$ for security predicates.

Theorem 3.2.5. Let SP be a security predicate. For every view $\mathcal{V}$ in E, $SP_\mathcal{V} : \mathcal{P}(E^*) \to \mathit{Bool}$ is a closure property of sets of traces. ◇

Proof. According to Definition 3.2.3, BSPs are closure properties of sets of traces. From Definition 2.2.7, we obtain that all BSPs hold for the set of all traces over E (i.e. E^*). Since BSPs are conjunctively connected, $SP_\mathcal{V}(E^*)$ holds. From Definitions 2.2.7, we obtain that $SP_\mathcal{V}$ is a closure property (E^* is a superset of any other set of traces over E). □

3.2.3 Information Flow Properties

In *MAKS*, an information flow property consists of a set of views and a security predicate.

Definition 3.2.6 (Information flow property). An *information flow property* is a pair (VS, SP) where VS is a set of views and SP is a security predicate. ◇

An information flow property is satisfied by a system if for all views the security predicate is satisfied by the set of possible traces.

Definition 3.2.7 (Satisfaction). Let (VS, SP) be an information flow property. *ES satisfies* (VS, SP) iff $SP_\mathcal{V}(Tr)$ holds for every view $\mathcal{V} \in VS$. ◇

3.3 From Abstract to Concrete Security Requirements

The restrictions on the permitted flow of information specified by a set of views are rather concrete as they are formulated in terms of the events of a specific system. In this section, we introduce the notion of *flow policies*, which can be used to specify restrictions on the permitted flow of information in a more abstract way. A graphical notation for flow policies provides the basis for an intuitive description of such restrictions. Using flow policies, the concrete set of views of an information flow property need not be specified explicitly but is rather generated from the flow policy using some additional information, the so called *domain assignment*.

3.3.1 Flow Policies

In a flow policy, restrictions on the permitted flow of information are expressed in terms of *security domains*. Typical security domains are, e.g., groups of users, sets of processes, collections of files, or memory sections. The benefit of expressing restrictions in terms of security domains, rather than in terms of the events of a specific event system (like it is done in a view), is that flow policies can be defined independently of the particular event system.

After the set of security domains $\mathcal{D}$ has been determined, the information flow between domains is specified by three binary relations $\rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow$ over $\mathcal{D}$. The *noninterference relation* $\not\rightsquigarrow$ specifies where information flow is forbidden. For two security domains $D_1, D_2 \in \mathcal{D}$, $D_1 \not\rightsquigarrow D_2$ expresses that information must not flow from D_1 to D_2. Rather than leaving it implicit where information flow is not forbidden, allowed information flow is specified explicitly by two relations $\rightsquigarrow_V, \rightsquigarrow_N$. The relations $\rightsquigarrow_V$ and $\rightsquigarrow_N$ differ in whether the activities of one domain are directly visible for another domain or not. The *interference relation* $\rightsquigarrow_V$ specifies that the activities of certain security domains are directly visible for other security domains. This means: $D_1 \rightsquigarrow_V D_2$ expresses that occurrences of events in D_1 *are visible* for D_2. The relation $\rightsquigarrow_N$ specifies between which domains (limited) *indirect* information flow is *permitted* although *direct* information flow is impossible (in contrast to $\rightsquigarrow_V$). In other words, $D_1 \rightsquigarrow_N D_2$ expresses that occurrences of events in D_1 are invisible for D_2 and that we do not care whether these occurrences can be deduced by D_2.

Since activities cannot be confidential and visible at the same time for a security domain, there are no other sensible possibilities to relate two security domains in terms of visibility and confidentiality. Therefore, any two domains should be related by exactly one of $\rightsquigarrow_V$, $\rightsquigarrow_N$, and $\not\rightsquigarrow$. Note the similarity to the classification of events in a view (cf. Figure 3.2).

Definition 3.3.1 (Flow policy). Let $\mathcal{D}$ be a set of security domains. A *flow policy* Pol is a quadruple $(\mathcal{D}, \rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow)$ where $\rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow \subseteq \mathcal{D} \times \mathcal{D}$. $\rightsquigarrow_V$ must be a reflexive relation and $\rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow$ must form a disjoint partition of $\mathcal{D} \times \mathcal{D}$. ◇

Definition 3.3.2. A flow policy $Pol = (\mathcal{D}, \rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow)$ is called *transitive* if $\rightsquigarrow_V$ is a transitive relation and, otherwise, *intransitive*.

Flow policies can be depicted as graphs where every node corresponds to a security domain. The relations $\rightsquigarrow_V$, $\rightsquigarrow_N$, and $\not\rightsquigarrow$ are, respectively, depicted as solid, dashed, and crossed arrows. For the sake of readability, the reflexive sub-relation of $\rightsquigarrow_V$ is usually omitted. This graphical representation is illustrated in Figure 3.3 for the flow policies Pol_H, Pol_{MLS}, Pol_P, and Pol_{HI} that are discussed in the following examples.

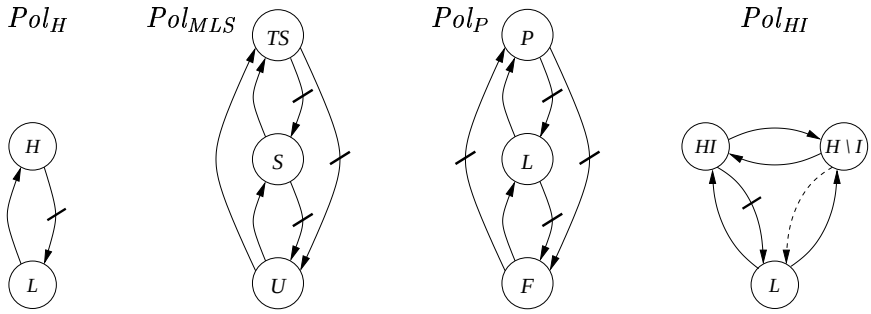


Figure 3.3: Example flow policies

Example 3.3.3. As depicted in Figure 3.3, the flow policy Pol_H consists of two security domains L (low-level events) and H (high-level events). According to this flow policy, occurrences of low-level events are visible for the high-level domain ($L \rightsquigarrow_V H$) and no information about occurrences of high-level events must be deducible for the low-level domain ($H \not\rightsquigarrow L$).

The flow policy Pol_H is a very simple example of a *multi-level security policy* (often abbreviated by *MLS-policy*) that is suitable for systems operating at two levels of confidentiality. For example, Pol_H could be used to specify the confidentiality requirements for a personal computer that on the one hand is used to process private data like, e.g., information about the financial assets of its user, and on the other hand is connected to the Internet. In this example, the main confidentiality requirement is that the private data cannot be communicated into the Internet (neither accidentally nor on purpose by some Trojan horse program). In order to express the informal security requirement correctly, it is necessary that all events concerned with processing of private data are associated with domain H and that all events involving accesses to the Internet are associated with domain L. If a system fulfills Pol_H then every adversary, who observes only occurrences of events in L (e.g. only Internet accesses), cannot deduce any information about the private data of the user. ◇

The 2-level flow policy Pol_H is well known in the literature on secure information flow (e.g. [GN88, McL94a, ZL97]).[2] There are two reasons for the popularity of Pol_H: firstly, it is a very simple flow policy and, secondly, reasoning about arbitrary *transitive* flow policies (with $\rightsquigarrow_N = \emptyset$) can be reduced to reasoning about Pol_H. Let us explain the latter using the example of a three-level security policy.

Example 3.3.4. The flow policy Pol_{MLS} (cf. Figure 3.3) has three domains U ("unclassified"), S ("secret"), and TS ("top secret"). The restrictions are that no information must flow from secret to unclassified ($S \not\rightsquigarrow U$) and no information must flow from top secret to secret or unclassified ($TS \not\rightsquigarrow S$, $TS \not\rightsquigarrow U$).

The requirements described by Pol_{MLS} can, alternatively, also be expressed by multiple instances of the flow policy Pol_H. For this purpose, we put ourselves successively in the perspective of a top secret user, in the perspective of a secret user, and in the perspective of an unclassified user who operate in the security domains TS, S, and U, respectively. For the *top secret user*, no confidentiality requirements exist because this user may observe occurrences of events in all three domains. The *secret user* must not be able to deduce any information about occurrences of events in TS. This user can only observe occurrences of events that are in domain S or U. Consequently, the security requirements for the secret user can be expressed by an instance of the flow policy Pol_H (associating H with all events that are associated with domain TS in Pol_{MLS} and associating L with all events that are associated with one of the domains S or U in Pol_{MLS}). The *unclassified user* must not be able to deduce any information about occurrences of events in S or TS. This user only observes occurrences of events that are in domain U. Consequently, the requirements for the unclassified user can be expressed by another instance of the flow policy Pol_H (associating H with all events that are associated with TS or S in Pol_{MLS} and associating L with all events that are associated with U in Pol_{MLS}).

Since every user operates in one security domain, it is equivalent to enforce either Pol_{MLS} or to enforce the two instances of Pol_H. ◇

The reduction to the two-level flow policy is only possible for flow policies that are transitive. Let us provide a simple example of an *intransitive flow policy*.

Example 3.3.5. The flow policy Pol_P (cf. Figure 3.3) has three domains P (for "printer"), L (for "labeler"), and F (for "file system"). The restrictions on the information flow are

[2] To be precise, in the literature the 2-level flow policy is defined using only two relations $\rightsquigarrow$ and $\not\rightsquigarrow$ where $\not\rightsquigarrow$ roughly corresponds to the union of our relations $\not\rightsquigarrow$ and $\rightsquigarrow_N$. Our definition of flow policies based on three relations $\rightsquigarrow_V$, $\rightsquigarrow_N$, and $\not\rightsquigarrow$ is novel (cf. Remark 3.3.7).

similar to the ones in Pol_{MLS} (if one identifies TS with P, S with L, and U with F). The only important difference between the two flow policies is that no information must flow from F to P according to Pol_P while information may flow from U to TS according to Pol_{MLS}.

Pol_P is an *intransitive flow policy* because $F \rightsquigarrow_V L$ and $L \rightsquigarrow_V P$ but $F \not\rightsquigarrow P$. At first sight, it might appear somewhat counterintuitive to restrict information flow from F to P but to allow information flow from F to P via the communication channel L. However, intransitive policies are often very useful in practice in order to express exceptions to restrictions on the information flow. For example, for a system that consists of a file system with multiple security levels (domain F), a labeler (domain L), and a printing service (domain P), the flow policy Pol_P imposes the restriction that the contents of files must pass the labeler before being printed. If the labeler operates correctly, i.e. if any data that passes the labeler is augmented by a label with security information on the respective data, then Pol_P enforces that all files have been labeled before being printed. ◇

In this thesis, we focus on transitive flow policies. Nevertheless, intransitive flow policies can be integrated into *MAKS*. We refer to [Man01a] for a description of how this can be done.

The following example illustrates the motivation of our distinction between three relations $\rightsquigarrow_V$, $\rightsquigarrow_N$, and $\not\rightsquigarrow$.

Example 3.3.6. In order to ensure confidentiality, we have the following: If a system does not generate any secret data and also does not increase the secrecy of input that it receives then it suffices to rule out that information about occurrences of high-level input events can be deduced [MC92]. Under these conditions, it is not necessary to explicitly prevent deductions about occurrences of high-level internal or output events because if any confidential information is deduced then something about the occurrence of high-level input events is also deduced. The underlying line of thought has been nicely explained by Guttman and Nadel (for systems that do not make any nondeterministic choices that are secret):

> *"High-level output is attributable partly to high-level input, partly to low-level input, and partly to the basic working of the system. A low-level user has access to information of the second and third kind anyway. Thus, if the user could determine something significant about high-level output, he would be able to attribute it to something about high-level input. Consequently, if we know that the user is unable to deduce anything about high-level input, it should follow that he cannot deduce anything* [critical] *about high-level output."* [GN88, page 37]

These assumption might hold, e.g., for a crypto-component that is used to encrypt secret data. The data is secret when it is input into the crypto-component and encrypting it does not make it more secret. Here, we assume that the encrypted data is only send to the high-level users as we are not concerned with issues of downgrading[3] and, therefore, the security requirement can be captured by the flow policy Pol_{HI} (cf. Figure 3.3).[4] Pol_{HI} consists of three domains HI, L, and $H \setminus I$. All high-level input events (events in $H \cap I$) are associated with domain HI (in particular, events that model receiving secret data). All high-level internal and output events (including computations of encryptions and sending of encrypted data) are associated with domain $H \setminus I$. All low-level events (whatever activities of the system that

[3] We refer to [Man01a] for how to integrate downgrading into *MAKS*.

[4] This security requirement could also be expressed by Pol_H. However, Pol_H is more restrictive than necessary and, therefore, Pol_{HI} is more appropriate under our assumptions.

are visible to potential adversaries) are associated with domain L. Consequently, according to Pol_{HI}, occurrences of low-level events are visible for both high-level domains ($L \rightsquigarrow_V HI$, $L \rightsquigarrow_V H \setminus I$). Information about occurrences of high-level input events must not be deducible for the low-level domain ($HI \not\rightsquigarrow L$). Occurrences of other high-level events are invisible for the low-level domain but information about such occurrences may be deduced (due to $H \setminus I \rightsquigarrow_N L$) if they do not reveal any information about occurrences of high-level inputs. ◇

Remark 3.3.7. Traditionally, two relations, $\rightsquigarrow$ and $\not\rightsquigarrow$, have been used to specify information flow policies. However, with only two relations it is not possible to distinguish all three possibilities to relate security domains. Rather, one either has to focus on visibility or on confidentiality. If one focuses on confidentiality then $\rightsquigarrow_V$ and $\rightsquigarrow_N$ fall together. If one focuses on visibility then $\rightsquigarrow_N$ and $\not\rightsquigarrow$ fall together. Note that the latter possibility corresponds to the traditional viewpoint, i.e. the traditional $\rightsquigarrow$ corresponds to our relation $\rightsquigarrow_V$ and the traditional $\not\rightsquigarrow$ corresponds to the union of our relations $\rightsquigarrow_N$ and $\not\rightsquigarrow$. The benefit of our approach with three relations over the traditional approach is a more precise specification of which events are confidential and which events are visible. ◇

3.3.2 Domain Assignments

Flow policies are generic in the sense that they can be applied to different event systems. In order to relate a flow policy to a given event system, all events must be associated with security domains in the flow policy. This is the purpose of *domain assignments*.

Definition 3.3.8 (Domain assignment). Let $\mathcal{D}$ be a set of security domains. A *domain assignment* is a function $dom : E \to \mathcal{D}$ that assigns a security domain to every event. ◇

For a given security domain $D \in \mathcal{D}$, we denote the subset of events that have this domain, also by the name of the domain, i.e. we use D to denote $\{e \in E \mid dom(e) = D\}$. Moreover, we use that name in lower case, possibly with indices or primes, e.g., d_1, d_2, ..., to denote events in that domain.

3.3.3 Deriving Sets of Views from Flow Policies

For defining what it means that the restrictions of a flow policy are satisfied, we consider every domain separately. Like in Example 3.3.4, we put ourselves in the perspective of a user who operates in one particular domain and distinguish events depending on visibility and confidentiality. The result is the *view of that domain*.

Definition 3.3.9 (View of domain). Let $Pol = (\mathcal{D}, \rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow)$ be a flow policy and $dom : E \to \mathcal{D}$ be a domain assignment. The *view* $\mathcal{V}_D = (V_D, N_D, C_D)$ *of a security domain* $D \in \mathcal{D}$ in Pol under dom is defined by

$$\begin{aligned} V_D &= \{e \in E \mid dom(e) \rightsquigarrow_V D\} \\ N_D &= \{e \in E \mid dom(e) \rightsquigarrow_N D\} \\ C_D &= \{e \in E \mid dom(e) \not\rightsquigarrow D\} \end{aligned}$$ ◇

Example 3.3.10. The views of all security domains in the flow policies Pol_H, Pol_{HI}, Pol_{MLS}, and Pol_P from Figure 3.3 are depicted in Figure 3.4 where the domain assignment is left implicit. For example, for domain L in Pol_{HI} occurrences of events in L are visible. Occurrences of events in $H \setminus I$ and HI are not observable for L. While information about events

in $H \setminus I$ may be deduced, no information about occurrences of events in HI may be deduced. Consequently, the view of L in Pol_{HI} is $\mathcal{HI}_L = (L, H \setminus I, HI)$. ◇

	$\mathcal{H}_H$	$\mathcal{H}_L$	
Pol_H	$(E, \emptyset, \emptyset)$	$(L, \emptyset, H)$	
	$\mathcal{HI}_{HI}$	$\mathcal{HI}_L$	$\mathcal{HI}_{H\setminus HI}$
Pol_{HI}	$(E, \emptyset, \emptyset)$	$(L, H \setminus I, HI)$	$(E, \emptyset, \emptyset)$
	$\mathcal{MLS}_{TS}$	$\mathcal{MLS}_S$	$\mathcal{MLS}_U$
Pol_{MLS}	$(E, \emptyset, \emptyset)$	$(S \cup U, \emptyset, TS)$	$(U, \emptyset, TS \cup S)$
	$\mathcal{P}_P$	$\mathcal{P}_L$	$\mathcal{P}_F$
Pol_P	$(P \cup L, \emptyset, F)$	$(L \cup F, \emptyset, P)$	$(F, \emptyset, L \cup P)$

Figure 3.4: Views for all domains in flow policies from Figure 3.3

The view of a given domain expresses what the restrictions on the information flow are for this domain. The views of all domains in a flow policy under a given domain assignment are collected in a *basic scene*.

Definition 3.3.11 (Basic scene). Let $Pol = (\mathcal{D}, \rightsquigarrow_V, \rightsquigarrow_N, \not\rightsquigarrow)$ be a flow policy and $dom : E \to \mathcal{D}$ be a domain assignment. The *basic scene* $\mathcal{BS}$ for Pol and dom is the set of the views of all domains in $\mathcal{D}$, i.e. $\mathcal{BS} = \{\mathcal{V}_D \mid D \in \mathcal{D}\}$ where $\mathcal{V}_D$ is the view of D in Pol under dom. ◇

Example 3.3.12. All views in the basic scenes for the flow policies Pol_H, Pol_{HI}, Pol_{MLS}, and Pol_P from Figure 3.3 are depicted in Figure 3.4. ◇

Rather than specifying the set of views of an information flow property directly, flow policies and domain assignments can be used for this purpose. Given a flow policy and a domain assignment a set of views is uniquely determined, i.e. the basic scene (cf. Definition 3.3.11).

We now turn our attention to the other ingredient of information flow properties, i.e. the security predicate or, more specifically, the BSPs from which it can be assembled.

3.4 A Collection of Basic Security Predicates

In this section, we present a collection of concrete basic security predicates. Each of these BSPs is a closure property of sets of traces that is parametric in a view. In other words, the BSPs defined in this section comply with Definition 3.2.3. Given a view $\mathcal{V} = (V, N, C)$ and a possible trace τ, a BSP requires that a particular *perturbation* of τ can be *corrected* to another possible trace τ'. *Perturbing* a possible trace τ means to modify occurrences of confidential events (i.e. events in C) in this trace. Examples of perturbations are the deletion of the last confidential event in a trace or the insertion of a confidential event at a point where no other occurrences of confidential events follow. In general, the sequence t that results from perturbing τ needs not to be a possible trace of the given system. However, the requirement imposed by a BSP is that t can be corrected in some way to a sequence τ' that is a possible trace of the system. *Correcting* a sequence t means to modify occurrences of invisible events that are not confidential (i.e. events in N). Note that the sequences τ, t, and τ' are all

indistinguishable from each other for an observer with the view $\mathcal{V}$ because $\tau|_V = t|_V = \tau'|_V$ holds. If a BSP holds then the observation of the sequence $\tau|_V$ does not reveal whether τ or τ' has occurred (both traces are possible and could have generated this observation). More generally, if a BSP holds then there are *sufficiently many possible traces* that could have generated a given observation such that an adversary cannot deduce any information of a particular kind (depending on the respective BSP). For example, if a BSP holds that perturbs traces by deleting the last occurrence of a confidential event then the adversary cannot tell whether a trace with an occurrence of some confidential event or a trace without this occurrence has generated the given observation. The BSP ensures that for every possible trace with an occurrence of the confidential event there is another possible trace without this occurrence that yields the same observation. Hence, an adversary cannot deduce from his observation that the confidential event must have occurred.

The various BSPs that we define differ in the particular perturbation that they require and/or in the corrections that they permit. The structure underlying all definitions of BSPs in this section is viewed in Figure 3.5.

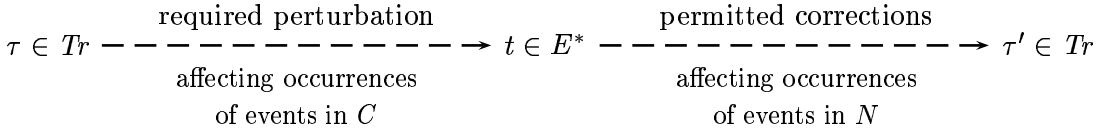


Figure 3.5: Pattern underlying the definition of all BSPs

For names of BSPs, we employ a uniform scheme. The name of each BSP that we define is a word according to the following regular expression:

$$(S \mid BS \mid FC)? \quad (R \mid D \mid I \mid IA)$$

The terminal symbols *R*, *D*, *I*, and *IA* in the name of a BSP indicate what kind of perturbation is required by that BSP. The (optional) terminal symbols *S*, *BS*, and *FC* indicate that the permitted corrections are restricted in some way. The syntactic expansions of all terminal symbols are listed in Figure 3.6, e.g., *S* stands for "Strict" and *BS* for "Backwards Strict". For example, a BSP with the name *BSIA* would be read as "Backwards-Strict Insertion of Admissible events". What it means for a BSP, e.g., to permit only "backwards-strict" corrections or to perturb a trace by "inserting admissible events" will be explained at different places of this section. Figure 3.7 gives an overview on where the concepts corresponding to each of these terminal symbols will be explained. A number next to each node in the diagram points to the respective section. The syntax diagram in the figure also specifies more precisely which words of the previously introduced grammar will be used as names of BSPs.

S	Strict	*R*	Removal of confidential events
BS	Backwards Strict	*D*	Deletion of confidential events
FC	Forward Correctable	*I*	Insertion of confidential events
		IA	Insertion of Admissible confidential events

Figure 3.6: Abbreviations used in names of BSPs

Simple BSPs (*R*, *D*, *I*) that perturb a possible trace by removing all confidential events or by either deleting or inserting a single occurrence of a confidential event will be introduced

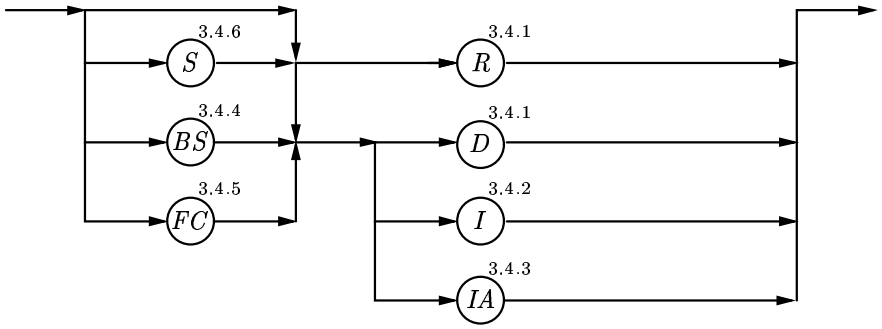


Figure 3.7: Syntax diagram for names of BSPs and pointers to sections

in Sections 3.4.1 and 3.4.2. These BSPs permit arbitrary corrections of perturbed traces. All other BSPs that we introduce (in Sections 3.4.3–3.4.6) are modifications of R, D, and I that result from relaxing the required perturbations or by restricting the permitted corrections. The different BSPs that we define will be compared to each other in Section 3.5.

3.4.1 Preventing Deductions about Occurrences of Events

We introduce two BSPs, R (for Removal) and D (for Deletion) that perturb a given trace, respectively, by *removing* all confidential events or by *deleting* the last occurrence of a confidential event. As we will show, these BSPs can be used to specify that an observer must not be able to deduce information about occurrences of confidential events.

Removal of Events. Given a view $\mathcal{V} = (V, N, C)$, $R_{\mathcal{V}}(Tr)$ perturbs a trace $\tau \in Tr$ by removing all occurrences of events in C and requires that this perturbation can be corrected by adapting occurrences of events in N to a possible trace $\tau' \in Tr$. Other corrections are not permitted. This means, no confidential events may occur in τ' ($\tau'|_C = \langle\rangle$) and τ' must yield the same observation as τ ($\tau'|_V = \tau|_V$). In summary, we obtain the following definition.

Definition 3.4.1. Let $\mathcal{V} = (V, N, C)$ be a view in E. The BSP R (Removal) is defined by:

$$R_{\mathcal{V}}(Tr) \quad \equiv \quad \forall \tau \in Tr.\, \exists \tau' \in Tr.\, (\tau'|_C = \langle\rangle \wedge \tau'|_V = \tau|_V) \qquad \diamond$$

The intuitive idea underlying the definition of R is as follows: If for every possible trace τ there is another possible trace τ' that does not contain any confidential events and that yields the same observation as τ then an adversary who observes $\tau|_V$ cannot tell whether τ or τ' has occurred. Both traces are possible and could have generated his observation.[5] Therefore, if $R_{\mathcal{V}}(Tr)$ holds then every observation (in V) can be generated by a possible trace without any confidential events. This means, for an adversary who observes occurrences of events in V, it is impossible to deduce any information about occurrences of confidential events (in C) from his observations. We illustrate the use of this BSP by the following two examples.

Example 3.4.2. Let us revisit the system from Example 2.2.9 that transmits high-level inputs directly to the low level. This system is intuitively insecure. Recall that $\mathtt{LEAK} = (E_{\mathtt{LEAK}}, I_{\mathtt{LEAK}}, O_{\mathtt{LEAK}}, Tr_{\mathtt{LEAK}})$ where $E_{\mathtt{LEAK}} = \{hi_n, lo_n \mid n \in \mathbb{N}\}$, $I_{\mathtt{LEAK}} = \{hi_n \mid n \in \mathbb{N}\}$, $O_{\mathtt{LEAK}} = \{lo_n \mid n \in \mathbb{N}\}$, and $Tr_{\mathtt{LEAK}} \subseteq E_{\mathtt{LEAK}}{}^*$ is the smallest set satisfying:

[5] More generally, he can deduce from his observation $\tau|_V$ and his knowledge of Tr that some trace from the *V-equivalence set* of τ, i.e. $\{\tau' \in Tr \mid \tau'|_V = \tau|_V\}$, has occurred but cannot narrow down the trace any further.

1. $\langle\rangle \in Tr_{\texttt{LEAK}}$,
2. if $n \in \mathbb{N}$ then $\langle hi_n \rangle \in Tr_{\texttt{LEAK}}$, and
3. if $n \in \mathbb{N}$ and $\tau \in Tr_{\texttt{LEAK}}$ then $\langle hi_n.lo_n \rangle.\tau \in Tr_{\texttt{LEAK}}$.

In this example, occurrences of high-level inputs are confidential and occurrences of low-level outputs are visible. The resulting view $\mathcal{V} = (V, N, C)$ is defined by $V = \{lo_n \mid n \in \mathbb{N}\}$, $N = \emptyset$, $C = \{hi_n \mid n \in \mathbb{N}\}$. Note that no corrections are permitted for this view because $N = \emptyset$ holds. Hence, $R_{\mathcal{V}}(Tr_{\texttt{LEAK}})$ requires that removing all occurrences of events in C from a possible trace results again in a possible trace. For example, the perturbation of $\langle hi_{42} \rangle$ is $\langle\rangle$. Since $\langle\rangle \in Tr_{\texttt{LEAK}}$ holds, the requirements of R are satisfied for this trace. However, the perturbation of the trace $\langle hi_{42}.lo_{42} \rangle$, i.e. $\langle lo_{42} \rangle$, is not a possible trace. Consequently, $R_{\mathcal{V}}(Tr_{\texttt{LEAK}})$ does not hold, i.e. the insecurity of system LEAK is correctly detected by R. ◇

Let us now illustrate how the insecurity of the system can be repaired by enlarging the set of possible traces.

Example 3.4.3. Let $\texttt{LEAK} = (E_{\texttt{LEAK}}, I_{\texttt{LEAK}}, O_{\texttt{LEAK}}, Tr_{\texttt{LEAK}})$ be defined like in Example 3.4.2 and let $\texttt{LEAK}_R = (E_{\texttt{LEAK}}, I_{\texttt{LEAK}}, O_{\texttt{LEAK}}, Tr_R)$ be defined by $Tr_R = Tr_{\texttt{LEAK}} \cup \{\tau|_{\{lo_n | n \in \mathbb{N}\}} \mid \tau \in Tr_{\texttt{LEAK}}\}$. That is, Tr_R results from $Tr_{\texttt{LEAK}}$ by adding the projections of all traces in $Tr_{\texttt{LEAK}}$ to low-level outputs. The difference in the behavior of LEAK and $\texttt{LEAK}_R$ is that $\texttt{LEAK}_R$ may also output sequences of random numbers to the low level without receiving any high-level input while LEAK only outputs numbers that it has received before from the high level. This means, a low-level observer of system $\texttt{LEAK}_R$ cannot distinguish whether he has received a (confidential) sequence from the high level or a (non-confidential) random sequence. Therefore, he cannot deduce anything about occurrences of high-level inputs, i.e. the insecurity has been repaired.

Since the set $\{\tau|_{\{lo_n | n \in \mathbb{N}\}} \mid \tau \in Tr_{\texttt{LEAK}}\}$ contains all traces that result from removing all confidential events from possible traces in Tr_R, $R_{\mathcal{V}}(Tr_R)$ also holds for the repaired system. ◇

Let us proceed with the introduction of the next BSP.

Stepwise Deletion of Events. Given a view $\mathcal{V} = (V, N, C)$, $D_{\mathcal{V}}(Tr)$ perturbs a possible trace $\tau \in Tr$ by deleting the last occurrence of an event in C and requires that this perturbation can be corrected by adapting occurrences of events in N to a possible trace $\tau' \in Tr$. Hence, for a possible trace $\beta.\langle c \rangle.\alpha$ (where $\beta \in E^*$ is arbitrary, $\alpha \in E^*$ contains no occurrences of events in C, and $c \in C$), another trace must be possible that can be represented by $\beta'.\alpha'$ (with $\alpha', \beta' \in E^*$) where the projection of α' and β' to $V \cup C$ must, respectively, equal the projections of α and β. In summary, we obtain the following definition.

Definition 3.4.4. Let $\mathcal{V} = (V, N, C)$ be a view in E. The BSP D (Deletion) is defined by:

$$
\begin{aligned}
&D_{\mathcal{V}}(Tr) \quad \equiv \\
&\quad \forall \alpha, \beta \in E^*. \forall c \in C. \\
&\quad [(\beta.\langle c \rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
&\quad \Rightarrow \exists \alpha', \beta' \in E^*. (\beta'.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle \wedge \beta'|_{V \cup C} = \beta|_{V \cup C})]
\end{aligned}
$$
◇

The intuitive idea underlying the definition of D is as follows: If for every possible trace $\beta.\langle c \rangle.\alpha$ there is another possible trace $\beta'.\alpha'$ that yields the same observation as $\beta.\langle c \rangle.\alpha$ then an adversary who observes $(\beta.\langle c \rangle.\alpha)|_V$ cannot tell whether $\beta.\langle c \rangle.\alpha$ or $\beta'.\alpha'$ has occurred. Both

traces are possible and could have generated his observation. In particular, he cannot deduce that the confidential event c must have occurred.

Note that D perturbs a trace by deleting only *a single occurrence* of a confidential event rather than removing *all occurrences* of confidential events. This is the difference between the BSPs D and R. However, Definition 3.4.4 can be applied recursively: If $\beta_1.\langle c'\rangle.\beta_2.\langle c\rangle.\alpha \in Tr$ (with $c, c' \in C$, $\alpha|_C = \langle\rangle$, $\beta_2|_C = \langle\rangle$) then $D_\mathcal{V}(Tr)$ implies that there are $\beta_1', \beta_2', \alpha' \in E^*$ with $\beta_1'.\langle c'\rangle.\beta_2'.\alpha' \in Tr$, $\beta_1'|_{V\cup C} = \beta_1|_{V\cup C}$, $\beta_2'|_{V\cup C} = \beta_2|_{V\cup C}$, and $\alpha'|_{V\cup C} = \alpha|_{V\cup C}$. Since $\beta_1'.\langle c'\rangle.\beta_2'.\alpha' \in Tr$ (with $c' \in C$, $(\beta_2'.\alpha')|_C = \langle\rangle$), $D_\mathcal{V}(Tr)$ implies that there are $\beta_1'', \beta_2'', \alpha'' \in E^*$ with $\beta_1''.\beta_2''.\alpha'' \in Tr$, $\beta_1''|_{V\cup C} = \beta_1'|_{V\cup C}$, $\beta_2''|_{V\cup C} = \beta_2'|_{V\cup C}$, and $\alpha''|_{V\cup C} = \alpha'|_{V\cup C}$. If β_1'' contains further occurrences of confidential events then D can be applied again and so on. A consequence of this recursive applicability is that, if $D_\mathcal{V}(Tr)$ holds then an adversary is prevented not only from deducing information about the *last* occurrence of a confidential event but also from deducing that *any* confidential events have occurred at all.

We illustrate the use of this BSP with the following examples.

Example 3.4.5. Let LEAK, $Tr_{\mathtt{LEAK}}$, and $\mathcal{V}$ be defined like in Example 3.4.2. Matching the pattern $\beta.\langle c\rangle.\alpha$ with the trace $\langle hi_{42}.lo_{42}\rangle$ yields $\beta = \langle\rangle$, $c = hi_{42}$, and $\alpha = \langle lo_{42}\rangle$. The preconditions of Definition 3.4.4 are satisfied because $\beta.\langle c\rangle.\alpha \in Tr_{\mathtt{LEAK}}$ and $\alpha|_C = \langle\rangle$ hold. Since $N = \emptyset$ holds, no corrections are permitted and, hence, $D_\mathcal{V}(Tr_{\mathtt{LEAK}})$ demands that $\beta.\alpha$ again is a possible trace. However, deleting hi_{42} in $\langle hi_{42}.lo_{42}\rangle$ results in $\beta.\alpha = \langle lo_{42}\rangle$, a trace that is not contained in $Tr_{\mathtt{LEAK}}$. Consequently, $D_\mathcal{V}(Tr_{\mathtt{LEAK}})$ does not hold, i.e. the insecurity of system LEAK is correctly detected by D. ◇

Let us now illustrate how the system can be modified in order to satisfy D.

Example 3.4.6. Let $\mathtt{LEAK}_R$, Tr_R, and $\mathcal{V}$ be defined like in Example 3.4.3. Matching the pattern $\beta.\langle c\rangle.\alpha$ with the trace $\langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle$ yields $\beta = \langle hi_{42}.lo_{42}\rangle$, $c = hi_{13}$, $\alpha = \langle lo_{13}\rangle$. The preconditions of Definition 3.4.4 are satisfied because $\beta.\langle c\rangle.\alpha \in Tr_R$ and $\alpha|_C = \langle\rangle$ hold. However, deleting c from $\beta.\langle c\rangle.\alpha$ results in $\beta.\alpha = \langle hi_{42}.lo_{42}.lo_{13}\rangle$, a trace that is not contained in Tr_R. Consequently, $D_\mathcal{V}(Tr_R)$ does not hold (although $R_\mathcal{V}(Tr_R)$ holds for this system).

In order to satisfy D it is necessary to add further traces to $Tr_{\mathtt{LEAK}}$. Let $\mathtt{LEAK}_D = (E_{\mathtt{LEAK}}, I_{\mathtt{LEAK}}, O_{\mathtt{LEAK}}, Tr_D)$ where $Tr_D \subseteq E^*_{\mathtt{LEAK}}$ is defined to be the smallest set satisfying:

1. If $\tau \in Tr_{\mathtt{LEAK}}$ then $\tau \in Tr_D$ and
2. if $\beta.\langle c\rangle.\alpha \in Tr_D$, $c \in C$, and $\alpha|_C = \langle\rangle$ then $\beta.\alpha \in Tr_D$.

This means, for a low-level observer it is impossible to tell whether a particular confidential event has occurred or not. There always is a possible trace without this event that could have generated the given observation. Therefore, he cannot deduce any information about occurrences of confidential events, i.e. the insecurity has been repaired.

Since Tr_D contains all traces that result from deleting the last occurrence of a confidential event in a possible trace in Tr_D, $D_\mathcal{V}(Tr_D)$ also holds for the repaired system. ◇

Obviously, D requires more possible traces to be added for repairing LEAK than R does ($Tr_{\mathtt{LEAK}} \subset Tr_R \subset Tr_D$ holds). However, this seems *not* to have any important advantages for preventing deductions about occurrences of confidential events. $\mathtt{LEAK}_R$ and $\mathtt{LEAK}_D$ both are intuitively "secure" in this sense. The main advantage of D over R is that its recursive definition makes various useful modifications possible (i.e. further BSPs to be introduced in Sections 3.4.4 and 3.4.5) that resemble basic ingredients of known information flow properties

(to be shown in Chapter 4). Let us illustrate the BSPs R and D with a few further very simple examples, which are illustrated also in Figure 3.8.

Example 3.4.7. Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ be an event system where $E_1 = \{l_1, h_1\}$ and $Tr_1 = \{\langle\rangle, \langle l_1\rangle, \langle l_1.h_1\rangle, \langle l_1.h_1.l_1\rangle\}$. Let $\mathcal{V}_1 = (V_1, N_1, C_1)$ be the view with $V_1 = \{l_1\}$, $N_1 = \emptyset$, and $C_1 = \{h_1\}$. The trace $\langle l_1.h_1.l_1\rangle$ yields the observation $\langle l_1.l_1\rangle$. However, no trace is possible that also yields this observation and in which no confidential events occur. Therefore, neither $R_{\mathcal{V}_1}(Tr_1)$ nor $D_{\mathcal{V}_1}(Tr_1)$ hold for this system. However, for the event system $ES_2 = (E_1, I_1, O_1, Tr_2)$ with $Tr_2 = Tr_1 \cup \{\langle l_1.l_1\rangle\}$, $R_{\mathcal{V}_1}(Tr_2)$ and $D_{\mathcal{V}_1}(Tr_2)$ are both satisfied.

	Removal		Deletion	
ES_1	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_1$	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_1$
	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\notin Tr_1$	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\notin Tr_1$
ES_2	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_2$	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_2$
	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_2$	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_2$
ES_3	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_3$	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_3$
	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_3$	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\notin Tr_3$
ES_4	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!n_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_4$	$\cdot\!-\!l_1\!\rightarrow\!\cdot\!-\!h_1\!\rightarrow\!\cdot\!-\!n_1\!\rightarrow\!\cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_4$
	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_4$	$\cdot\!-\!l_1\!\rightarrow\!\cdot \quad \cdot \quad \cdot\!-\!l_1\!\rightarrow\!\cdot$	$\in Tr_4$

Figure 3.8: Illustration of Example 3.4.7

Let us illustrate the difference between R and D with another example. Define $ES_3 = (E_1, I_1, O_1, Tr_3)$ and $Tr_3 = \{\langle\rangle, \langle l_1\rangle, \langle l_1.h_1\rangle, \langle l_1.h_1.h_1\rangle, \langle l_1.h_1.h_1.l_1\rangle, \langle l_1.l_1\rangle\}$. While $R_{\mathcal{V}_1}(Tr_3)$ holds, $D_{\mathcal{V}_1}(Tr_3)$ does not hold because $\langle l_1.h_1.h_1.l_1\rangle \in Tr_3$ and $\langle l_1.h_1.l_1\rangle \notin Tr_3$.

Let us now investigate R and D in the context of a view for which $N \neq \emptyset$ holds. Define $ES_4 = (E_4, I_1, O_1, Tr_4)$, $E_4 = \{l_1, h_1, n_1\}$, $Tr_4 = \{\langle\rangle, \langle l_1\rangle, \langle l_1.h_1\rangle, \langle l_1.h_1.n_1\rangle, \langle l_1.h_1.n_1.l_1\rangle, \langle l_1.l_1\rangle\}$ (i.e. after h_1 has occurred an event n_1 must occur before l_1 becomes enabled again) and $\mathcal{V}_4 = (\{l_1\}, \{n_1\}, \{h_1\})$. For this system, $R_{\mathcal{V}_4}(Tr_4)$ as well as $D_{\mathcal{V}_4}(Tr_4)$ hold. When checking that $D_{\mathcal{V}_4}(Tr_4)$ holds, e.g., the trace $\langle l_1.h_1.n_1.l_1\rangle$ is matched with $\beta.\langle c\rangle.\alpha$, resulting in $\beta = \langle l_1\rangle$, $c = h_1$, and $\alpha = \langle n_1.l_1\rangle$. For the choices $\beta' = \beta$ and $\alpha' = \langle l_1\rangle$, the requirements $\beta'.\alpha' \in Tr$, $\alpha'|_V = \alpha|_V$, $\alpha'|_C = \langle\rangle$, and $\beta'|_{V\cup C} = \beta|_{V\cup C}$ of Definition 3.4.4 are, indeed, satisfied. Note that, in this example, it is crucial to correct the perturbation because the perturbation itself (i.e. $\langle l_1.n_1.l_1\rangle$) is not a possible trace of the system. $\diamond$

3.4.2 Preventing Deductions about Nonoccurrences of Events

The BSPs R and D that we defined in the preceeding section prevent that information about occurrences of confidential events can be deduced. However, they do not rule out that it can be deduced that a particular confidential event has *not* occurred.

Example 3.4.8. Let $\texttt{LEAK}_D$, Tr_D, and $\mathcal{V}$ be defined like in Example 3.4.6. An adversary who observes $\langle lo_{42}.lo_{13}\rangle$ can deduce that a trace from the V-equivalence set of $\langle lo_{42}.lo_{13}\rangle$, i.e.

$$\{\langle lo_{42}.lo_{13}\rangle,\ \langle hi_{42}.lo_{42}.lo_{13}\rangle,\ \langle hi_{42}.lo_{42}.hi_{13}.lo_{13}\rangle\} \cup \{\langle hi_{42}.lo_{42}.hi_{13}.lo_{13}.hi_n\rangle \mid n \in \mathbb{N}\}$$

must have occurred. From this set, he can conclude, e.g., that hi_7 has *not* occurred before the occurrence of lo_{42} and that hi_{42} has *not* occurred in between the occurrence of lo_{42} and lo_{13}. Many further deductions about nonoccurrences of confidential events are possible. This means, although $R_{\mathcal{V}}(Tr_D)$ and $D_{\mathcal{V}}(Tr_D)$ hold, the adversary can deduce information about nonoccurrences of confidential events. ◇

If an adversary can deduce that certain confidential events have not occurred then this may be problematic for the security of a system. In particular, it is possible to establish a covert channel from a high-level Trojan horse to a low-level adversary by exploiting deductions about nonoccurrences of events. Let us illustrate the problem with two examples.

Example 3.4.9. Assume that $\texttt{LEAK}_D$ is used by a high-level Trojan horse that is capable of providing input to the system faster than $\texttt{LEAK}_D$ can send output to the low-level user (the adversary). The Trojan horse can transmit a sequence of natural numbers as follows: It inputs the first number n_1 (occurrence of event hi_{n_1}) before the system has output any number to the low-level user, it waits until the system has output a number to the low-level user that, according to Tr_D, must be n_1 (occurrence of event lo_{n_1}), inputs the next number n_2 (occurrence of event hi_{n_2}), etc. The low-level user observes a trace $\langle lo_{n_1}.lo_{n_2}\ldots\rangle$. From this observation and the knowledge of the set Tr_D of possible traces for $\texttt{LEAK}_D$, the adversary can deduce that high-level events other than hi_{n_1} cannot have occurred before lo_{n_1}, that high-level events other than hi_{n_2} cannot have occurred in between lo_{n_1} and lo_{n_2}, and so on. This information about nonoccurrences of events together with the knowledge of the protocol used by the Trojan horse is already enough to deduce the sequence that has been input by the Trojan horse. It is precisely the sequence that the adversary has received. ◇

Example 3.4.10. Another example, where deductions about nonoccurrences of events are problematic is a system that logs the actions of low-level users in order to discourage undesired actions. However, logs are not created permanently but only upon request. Clearly, low-level users should not know if their actions are logged or not and, consequently, the request to log the actions of a particular low-level user is a confidential event. If a low-level user could deduce from his observations that this event has not occurred then he would know when his actions *definitely are not logged* (no log-request has been issued). Deductions about occurrences of confidential events are not so problematic in this example: If the user deduces that the confidential event has occurred then he only learns that his actions *definitely are logged.* Consequently, in this setting a malicious low-level user is more interested in deducing nonoccurrences of confidential events than in deducing occurrences of these events. Conversely, for the security of the system it is more important to prevent deductions about nonoccurrences of confidential events than to prevent deductions about occurrences. ◇

We now introduce the BSP I (for Insertion) that perturbs a possible trace by *inserting* occurrences of confidential events. As we will show, this BSP can be used to prevent information about nonoccurrences of confidential events from being deduced.

Insertion of Events. Given a view $\mathcal{V} = (V, N, C)$, $I_{\mathcal{V}}(Tr)$ perturbs a trace $\tau \in Tr$ by inserting an event in C at a position where it is not followed by other confidential events and demands that this perturbation can be corrected by adapting occurrences of events in N to a possible trace $\tau' \in Tr$. Hence, for a possible trace $\beta.\alpha$ (where $\beta \in E^*$ is arbitrary and $\alpha \in E^*$ contains no occurrences of events in C) and an event $c \in C$, another trace must be possible

that can be represented by $\beta'.\langle c\rangle.\alpha'$ where the projection of α' and β' to $V \cup C$ must equal the projections of α and β, respectively. In summary, we obtain the following definition.

Definition 3.4.11. Let $\mathcal{V} = (V, N, C)$ be a view in E. The BSP I (Insertion) is defined by:

$$
\begin{aligned}
&I_{\mathcal{V}}(Tr) \quad \equiv \\
&\quad \forall \alpha, \beta \in E^*. \forall c \in C. \\
&\quad [(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
&\quad \Rightarrow \exists \alpha', \beta' \in E^*. (\beta'.\langle c\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle \wedge \beta'|_{V\cup C} = \beta|_{V\cup C})]
\end{aligned}
$$ ◇

The intuitive idea underlying the definition of I is the following: If for every possible trace $\beta.\alpha$ there is another possible trace $\beta'.\langle c\rangle.\alpha'$ that yields the same observation as $\beta.\alpha$ then an adversary who observes $(\beta.\alpha)|_V$ cannot tell whether $\beta.\alpha$ or $\beta'.\langle c\rangle.\alpha'$ has occurred. Both traces are possible and could have generated his observation. In particular, he cannot deduce that the confidential event c has *not* occurred.

Note that Definition 3.4.11 can be applied recursively: If $\beta.\alpha_1.\alpha_2 \in Tr$ (with $\alpha_1|_C = \langle\rangle$, $\alpha_2|_C = \langle\rangle$) and $c \in C$ then $I_{\mathcal{V}}(Tr)$ implies that there are $\beta', \alpha_1', \alpha_2' \in E^*$ with $\beta'.\langle c\rangle.\alpha_1'.\alpha_2' \in Tr$, $\beta'|_{V\cup C} = \beta|_{V\cup C}$, $\alpha_1'|_{V\cup C} = \alpha_1|_{V\cup C}$, and $\alpha_2'|_{V\cup C} = \alpha_2|_{V\cup C}$. Since $\beta'.\langle c\rangle.\alpha_1'.\alpha_2' \in Tr$ (where $\alpha_2'|_C = \langle\rangle$), $I_{\mathcal{V}}(Tr)$ implies for $c' \in C$ that there are $\beta'', \alpha_1'', \alpha_2'' \in E^*$ with $\beta''.\langle c\rangle.\alpha_1''.\langle c'\rangle.\alpha_2'' \in Tr$, $\beta''|_{V\cup C} = \beta'|_{V\cup C}$, $\alpha_1''|_{V\cup C} = \alpha_1'|_{V\cup C}$, and $\alpha_2''|_{V\cup C} = \alpha_2'|_{V\cup C}$. To insert further occurrences of confidential events, I can be applied again and so on.

Remark 3.4.12. The recursive application of I is similar to the recursive application of D. However, a difference is that, for I, the resulting trace $\beta'.\langle c\rangle.\alpha'$ always satisfies the precondition of the definition such that the process of stepwise insertion of occurrences of confidential events does not terminate. This is not the case for the definition of D. Consequently, sets of traces that satisfy I are of infinite size, unless $C = \emptyset$ holds, while sets of traces that satisfy D may be finite even if $C \neq \emptyset$. ◇

In Example 3.4.9, we explained how a Trojan horse could transmit information across the system $\mathtt{LEAK}_D$. The BSPs R and D did not detect this possibility for information leakage. Let us now check whether the newly introduced BSP I detects this insecurity.

Example 3.4.13. Let $\mathtt{LEAK}_D$, Tr_D, and $\mathcal{V}$ be defined like in Example 3.4.8. For $\beta = \langle hi_{42}.lo_{42}.hi_{13}\rangle$ and $\alpha = \langle lo_{13}\rangle$, the preconditions of Definition 3.4.11 are satisfied because $\beta.\alpha \in Tr_D$ and $\alpha|_C = \langle\rangle$ hold. However, inserting the confidential event $c = hi_7$ into $\beta.\alpha$ results in $\beta.\langle c\rangle.\alpha = \langle hi_{42}.lo_{42}.hi_{13}.hi_7.lo_{13}\rangle$, a trace that is not contained in Tr_D. Consequently, $I_{\mathcal{V}}(Tr_D)$ does not hold, i.e. the insecurity of the system is detected by I. ◇

Again, we show how the insecurity of the system can be repaired by adding possible traces.

Example 3.4.14. Let $\mathtt{LEAK}_I = (E_{\mathtt{LEAK}}, I_{\mathtt{LEAK}}, O_{\mathtt{LEAK}}, Tr_I)$ where $Tr_I \subseteq {E_{\mathtt{LEAK}}}^*$ is defined to be the smallest set satisfying:

1. If $\tau \in Tr_D$ then $\tau \in Tr_I$ and
2. if $\beta.\alpha \in Tr_I$, $c \in C$, and $\alpha|_C = \langle\rangle$ then $\beta.\langle c\rangle.\alpha \in Tr_I$.

That is, for a low-level observer it is impossible to deduce that a particular confidential event cannot have occurred. For each observation, there is a possible trace that could have generated the observation and in which this event occurs. Therefore, the observer cannot deduce any information about nonoccurrences of confidential events. The insecurity has been repaired.

Since Tr_I contains all traces that result from inserting a confidential event into a possible trace at a point where it is not followed by other confidential events, $I_{\mathcal{V}}(Tr_I)$ also holds for the repaired system. ◇

3.4.3 Compatibility with Confidential Computations

If $I_{\mathcal{V}}(Tr)$ holds for a given system then occurrences of visible events do not depend on nonoccurrences of confidential events. Therefore, an adversary cannot deduce from his observations that some confidential event has not occurred. To prevent such deductions from being possible is the whole purpose of the BSP I. Moreover, if $I_{\mathcal{V}}(Tr)$ holds for $\mathcal{V} = (V, N, C)$ then the enabledness of events in C does not depend on the history, i.e. I requires that confidential events can be inserted at any point of a trace (where no other occurrences of confidential events follow). Consequently, a *necessary* condition for the satisfaction of $I_{\mathcal{V}}(Tr)$ is that all sequences of events in C^* are possible for the system under consideration, i.e. $\{\tau|_C \mid \tau \in Tr\} = C^*$ must hold (recall that $\emptyset \in Tr$ because Tr is closed under prefixes). While this might be no problem with systems like $\mathtt{LEAK}_I$ in Example 3.4.14 because all confidential events are also input events, it poses a major problem with other systems. If $\{\tau|_C \mid \tau \in Tr\} = C^*$ holds for a system then it behaves *chaotically* in its confidential events. Conversely, if meaningful (non-chaotic) computations are modeled by sequences of events in C then $I_{\mathcal{V}}(Tr)$ is unlikely to hold. Let us investigate a simple example.

Example 3.4.15. Define $\mathtt{PIPE} = (E_{\mathtt{PIPE}}, I_{\mathtt{PIPE}}, O_{\mathtt{PIPE}}, Tr_{\mathtt{PIPE}})$, $E_{\mathtt{PIPE}} = \{hi_n, ho_n \mid n \in \mathbb{N}\}$, $I_{\mathtt{PIPE}} = \{hi_n \mid n \in \mathbb{N}\}$, and $O_{\mathtt{PIPE}} = \{ho_n \mid n \in \mathbb{N}\}$. Let $Tr_{\mathtt{PIPE}} \subseteq {E_{\mathtt{PIPE}}}^*$ be the smallest set satisfying:

1. $\langle\rangle \in Tr_{\mathtt{PIPE}}$,
2. if $n \in \mathbb{N}$ then $\langle hi_n \rangle \in Tr_{\mathtt{PIPE}}$, and
3. if $n \in \mathbb{N}$ and $\tau \in Tr_{\mathtt{PIPE}}$ then $\langle hi_n.ho_n \rangle.\tau \in Tr_{\mathtt{PIPE}}$.

According to this specification, PIPE receives a number n as high-level input (event hi_n) and immediately outputs this number to the high level (event ho_n). Hence, the difference from the system LEAK is that PIPE outputs the number to the high level rather than to the low level. For the view $\mathcal{V} = (V, N, C)$ with $V = \emptyset$, $N = \emptyset$, $C = E$, intuitively the security of PIPE is obvious. In particular, it is clear that information about nonoccurrences of confidential events cannot be deduced because the only observation possible is $\langle\rangle$ (there are no visible events). Nevertheless, PIPE does not satisfy $I_{\mathcal{V}}(Tr_{\mathtt{PIPE}})$. Moreover, repairing PIPE by adding possible traces results in $\mathtt{CHAOS} = (E_{\mathtt{PIPE}}, I_{\mathtt{PIPE}}, O_{\mathtt{PIPE}}, Tr_{\mathtt{CHAOS}})$ where $Tr_{\mathtt{CHAOS}} = {E_{\mathtt{PIPE}}}^*$ is the set of *all* traces over $E_{\mathtt{PIPE}}$. Hence, with respect to the intended functionality of PIPE, CHAOS is not a very useful system. Summarizing, the BSP I rejects the system PIPE although it intuitively is secure and the only repair of this system that satisfies I is not very useful. ◇

The previous example has shown that the requirement $I_{\mathcal{V}}(Tr)$ is sometimes too strong. Although the deduction of information about nonoccurrences of confidential events is impossible across a given system this system might not satisfy $I_{\mathcal{V}}(Tr)$. The problem is that in order to satisfy $I_{\mathcal{V}}(Tr)$, all sequences of confidential events must be possible for the given system. However, this implies that sequences of confidential events cannot model meaningful computations.

A straightforward solution to this problem would be to insert the formula

$$\exists \gamma \in E^*.\, (\gamma.\langle c\rangle \in Tr \wedge \gamma|_C = \beta|_C) \tag{3.1}$$

as an additional assumption into the definition of I, which results in:

$$\begin{aligned} &\forall \alpha, \beta \in E^*. \forall c \in C. \\ &\Big[(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle \wedge \underline{\exists \gamma \in E^*.\, (\gamma.\langle c\rangle \in Tr \wedge \gamma|_C = \beta|_C)}) \\ &\Rightarrow \exists \alpha', \beta' \in E^*.\, (\beta'.\langle c\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle \wedge \beta'|_{V\cup C} = \beta|_{V\cup C})\Big] \end{aligned} \tag{3.2}$$

Adding Condition (3.1) as an assumption weakens I in the sense that fewer perturbations are required. Confidential events only need to be insertable at a given position of a possible trace if they are enabled at this position in *some* possible trace. More precisely, an event $c \in C$ need only be insertable after β in $\beta.\alpha$ if $(\beta|_C).\langle c\rangle$ is a possible sequence of confidential events, i.e. if there is a possible trace $\gamma.\langle c\rangle$ such that $(\gamma.\langle c\rangle)|_C = (\beta|_C).\langle c\rangle$ holds. The positive effect of this relaxation is that occurrences of confidential events may depend on which confidential events have occurred so far. This means that (3.2) does not rule out meaningful confidential computations and, e.g., is satisfied for the intuitively secure system PIPE. However, unfortunately (3.2) does not ensure that deductions about nonoccurrences of confidential events are completely impossible and, hence, some insecurities might remain undetected (unless additional BSPs are enforced). This is illustrated by the following example.

Example 3.4.16. The system LEAKONCE is a modified version of system LEAK that stops after exactly one number has been leaked to the low level. That is, $\mathtt{LEAKONCE} = (E_{\mathtt{LEAK}}, I_{\mathtt{LEAK}}, O_{\mathtt{LEAK}}, Tr_{\mathtt{LEAKONCE}})$ where $E_{\mathtt{LEAK}} = \{hi_n, lo_n \mid n \in \mathbb{N}\}$, $I_{\mathtt{LEAK}} = \{hi_n \mid n \in \mathbb{N}\}$, $O_{\mathtt{LEAK}} = \{lo_n \mid n \in \mathbb{N}\}$, and $Tr_{\mathtt{LEAKONCE}} = \{\langle\rangle\} \cup \{\langle hi_n\rangle \mid n \in \mathbb{N}\} \cup \{\langle hi_n.lo_n\rangle \mid n \in \mathbb{N}\}$. For example, after observing $\langle lo_1\rangle$, an adversary can deduce that hi_2 cannot have occurred and after observing $\langle lo_2\rangle$ he can deduce that hi_1 cannot have occurred. Consequently, it is possible for him to deduce some information about nonoccurrences of confidential events with system LEAKONCE. Nevertheless, LEAKONCE satisfies (3.2). ◇

The reason why deductions of information about nonoccurrences of confidential events are possible in the previous example is that the low-level observation reveals enough information in order to infer whether Condition (3.1) holds or not. This problem can be avoided if R or D is enforced in addition to (3.2). Note that neither $R_{\mathcal{V}}(Tr_{\mathtt{LEAKONCE}})$ nor $D_{\mathcal{V}}(Tr_{\mathtt{LEAKONCE}})$ hold for LEAKONCE. In general, if $R_{\mathcal{V}}(Tr)$ or $D_{\mathcal{V}}(Tr)$ as well as (3.2) are satisfied for a view $\mathcal{V} = (V, N, C)$ in E and a set $Tr \subseteq E^*$ then the following equation holds for all $\tau, \tau' \in Tr$:

$$\{t|_C \mid t \in Tr \wedge t|_V = \tau|_V\} = \{t'|_C \mid t' \in Tr \wedge t'|_V = \tau'|_V\}\ . \tag{3.3}$$

Consequently, all sequences of confidential events that are possible with some observation (here: $\tau|_V$) are also possible with every other observation (here: $\tau'|_V$). This means, an observer might be able to deduce that certain sequences of confidential events cannot have occurred. However, which sequences can be excluded does not depend at all on observations of the dynamic system behavior but only on Tr, i.e. the specification of the system. Hence, the information that can be deduced is quite limited.

Moreover, if $R_{\mathcal{V}}(Tr)$ (or $D_{\mathcal{V}}(Tr)$) and (3.2) hold then for every $\tau \in Tr$ and every possible sequence $t_c \in C^*$ (i.e. $\exists \tau' \in Tr.\, \tau'|_C = t_c$) we have:

$$\forall t \in E^*.\, [(t|_C = t_c \wedge t|_V = \tau|_V) \Rightarrow \exists t' \in Tr.\, t'|_{V\cup C} = t|_{V\cup C}]\ . \tag{3.4}$$

Consequently, every interleaving (here: t) of a possible sequence of confidential events (here: t_c) with an observation (here: $\tau|_V$) can be corrected to a possible trace (here: t') by adapting occurrences of events in N. This means, an observer cannot narrow down the possible interleaving (of occurrences of confidential events and occurrences of visible events) that has generated his observation. That no information about the interleaving can be deduced is very important for security, as Guttman and Nadel have shown [GN88].

In summary, if (3.2) is required for a system then one of $R_{\mathcal{V}}(Tr)$ or $D_{\mathcal{V}}(Tr)$ should be required in addition. If this advice is respected then an observer (of occurrences of events in V) cannot narrow down the behavior that has occurred with the help of his observations. All the information about nonoccurrences of confidential events that he obtains can already be obtained statically from the system specification. More specifically, he can only deduce that sequences of confidential events not in $\{\tau|_C \mid \tau \in Tr\}$ cannot occur. With most systems, the possibility of such deductions will not be problematic. However, there might also be systems for which this is not true. Therefore, when requiring (3.2) instead of $I_{\mathcal{V}}(Tr)$ then one should justify that these deductions are not problematic for the system under consideration.

A more serious problem is that (3.2) is still too restrictive in the sense that this condition does not hold for some systems although they are intuitively secure. The problem is that Condition (3.1) only permits that confidential events depend on previous occurrences of confidential events in C but not on previous occurrences of events in V. However, information flow from V to C is noncritical and, hence, need not be prevented. The following example illustrates that (3.2) is too restrictive for some intuitively secure systems.

Example 3.4.17. Let $\mathtt{UP} = (E_{\mathtt{UP}}, I_{\mathtt{UP}}, O_{\mathtt{UP}}, Tr_{\mathtt{UP}})$ with $E_{\mathtt{UP}} = \{li_n, ho_n \mid n \in \mathbb{N}\}$, $I_{\mathtt{UP}} = \{li_n \mid n \in \mathbb{N}\}$, $O_{\mathtt{UP}} = \{ho_n \mid n \in \mathbb{N}\}$, and $Tr_{\mathtt{UP}} \subseteq E_{\mathtt{UP}}{}^*$ be defined as the smallest set satisfying:

1. $\langle\rangle \in Tr_{\mathtt{UP}}$,
2. if $n \in \mathbb{N}$ and $\tau \in Tr_{\mathtt{UP}}$ then $\langle li_n \rangle.\tau \in Tr_{\mathtt{UP}}$, and
3. if $n \in \mathbb{N}$ and $\tau \in Tr_{\mathtt{UP}}$ then $\langle li_n.ho_n \rangle.\tau \in Tr_{\mathtt{UP}}$.

According to this specification, UP receives a number n as low-level input (event li_n) and either immediately outputs this number to the high level (event ho_n) and waits for the next low-level input or waits for the next input without outputting anything. The main difference to the system PIPE is that UP receives the input from the low level rather than from the high level. Hence, UP creates a partial log of the low-level operations. For the view $\mathcal{V} = (V, N, C)$ with $V = \{li_n \mid n \in \mathbb{N}\}$, $N = \emptyset$, $C = \{ho_n \mid n \in \mathbb{N}\}$, intuitively no critical information about nonoccurrences of confidential events can be deduced because the system simply *upgrades* information that it receives from the low level. Nevertheless, UP does not satisfy (3.2).

Let us now try to repair the system by adding possible traces. Let $Tr_{I_{\mathrm{MOD}}} \subseteq E_{\mathtt{UP}}{}^*$ be the smallest set satisfying:

1. $\langle\rangle \in Tr_{I_{\mathrm{MOD}}}$,
2. if $n \in \mathbb{N}$ and $\tau \in Tr_{I_{\mathrm{MOD}}}$ then $\langle li_n \rangle.\tau \in Tr_{I_{\mathrm{MOD}}}$, and
3. if $t \in O_{\mathtt{UP}}{}^*$ and $\tau \in Tr_{I_{\mathrm{MOD}}}$ then $t.\tau \in Tr_{I_{\mathrm{MOD}}}$.

Obviously, (3.2) holds for $Tr_{I_{\mathrm{MOD}}}$ (and $R_{\mathcal{V}}(Tr_{I_{\mathrm{MOD}}})$, $D_{\mathcal{V}}(Tr_{I_{\mathrm{MOD}}})$ hold also). However, the resulting system is not very useful wrt. the intended functionality. The sequence of high-level

outputs is not a recording of low-level inputs any more (not even an incomplete one). Rather, a possible trace in $Tr_{I_{\mathrm{MOD}}}$ may contain spurious high-level outputs. When investigating the high-level output, e.g. the log, it is completely unclear whether a particular event ho_n records some low-level input that actually has occurred or not. ◇

The previous example has demonstrated that our first modification of the BSP I is still too restrictive for some secure systems. The problem is that information flow from V to C is ruled out although such information flow is not security critical. A straightforward solution to permit noncritical information flow would be to replace Condition (3.1) in (3.2) by the following stronger condition:[6]

$$\exists \gamma \in E^*.\, (\gamma.\langle c\rangle \in Tr \wedge \gamma|_E = \beta|_E) \tag{3.5}$$

It is easy to check that the resulting (weaker) modification of I is satisfied, e.g., for the intuitively secure system UP from Example 3.4.17.

However, this modified version must be applied with care. Replacing Condition (3.1) by Condition (3.5) results in a property that might be fulfilled for a given system although information about nonoccurrences of confidential events can be deduced. For instance, for the system UP in Example 3.4.17 it can be deduced from the observation $\langle li_{13}\rangle$ that the next event cannot be ho_7 or ho_{42} (rather it must be either ho_{13} or some arbitrary low-level event). Hence, our second modification of I does not prevent that information about *future* nonoccurrences of confidential events can be deduced. Such deductions are possible even if R or D is enforced in addition (e.g. UP satisfies $R_{\mathcal{V}}(Tr_{\mathtt{UP}})$ as well as $D_{\mathcal{V}}(Tr_{\mathtt{UP}})$). Depending on the particular application these deductions might be problematic or not. Therefore, when requiring our second modification of I one needs to justify that the permitted deductions do no pose a problem for security of the system under consideration.

Note that Conditions (3.1) and (3.5) differ only in the set for which the projection is constructed (set C in (3.1) and set E in (3.5)). This set is used to construct the projection. The following definition generalizes these two conditions by leaving the set for which the projection is constructed parametric. More specifically, the parameter is a function that takes a view as parameter and returns a set of events.

Definition 3.4.18 (ρ-admissibility). Let ρ be a function from views in E to subsets of E. An event $e \in E$ is *ρ-admissible* in Tr after a possible trace $\beta \in Tr$ for a view $\mathcal{V} = (V, N, C)$ in E if $Adm^{\rho}_{\mathcal{V}}(Tr, \beta, e)$ holds. Adm is defined as follows:

$$Adm^{\rho}_{\mathcal{V}}(Tr, \beta, e) \equiv \exists \gamma \in E^*.\, (\gamma.\langle e\rangle \in Tr \wedge \gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}) \qquad \diamond$$

Hence, $Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)$ holds if there is a possible trace $\gamma.\langle c\rangle \in Tr$ that coincides with $\beta.\langle c\rangle$ in all occurrences of events in $\rho(\mathcal{V})$.

We are now ready to introduce the general version of our modification of the BSP I.

Insertion of ρ-Admissible Events. Inserting ρ-admissibility as an additional assumption in the definition of I results in IA. IA is not only parametric in the view but also in the function ρ. Hence, we obtain a family IA^{ρ}, $IA^{\rho'}$, $IA^{\rho''}$, ... of BSPs.

Definition 3.4.19. Let ρ be a function from views in E to subsets of E and $\mathcal{V} = (V, N, C)$ be a view in E. The BSP IA^{ρ} (Insertion of ρ-Admissible events) is formally defined by:

[6] Note that Condition (3.5) is equivalent to $\beta.\langle c\rangle \in Tr$.

$IA^{\rho}_{\mathcal{V}}(Tr) \equiv$
$\forall \alpha, \beta \in E^*. \forall c \in C.$
$[(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle \wedge Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c))$
$\Rightarrow \exists \alpha', \beta' \in E^*. (\beta'.\langle c \rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle \wedge \beta'|_{V \cup C} = \beta|_{V \cup C})]$ ◇

The intuitive idea underlying the definition of IA^{ρ} is like the one underlying the definition of I. However, IA^{ρ} requires for a given view $\mathcal{V} = (V, N, C)$ only that one cannot deduce from observations in V that a *ρ-admissible* event in C has not occurred. Let us illustrate the use of IA^{ρ} with an example, which illustrates also the effects of using different parameters ρ.

Example 3.4.20. Let ρ_C and ρ_E be functions from views in E to subsets of E that, respectively, are defined by $\rho_C(V, N, C) = C$ and $\rho_E(V, N, C) = V \cup N \cup C = E$. For the system PIPE from Example 3.4.15 and the view $\mathcal{V} = (V, N, C)$ with $V = \emptyset$, $N = \emptyset$, $C = \{hi_n, ho_n \mid n \in \mathbb{N}\}$, $IA^{\rho_C}_{\mathcal{V}}(Tr_{\mathtt{PIPE}})$ and $IA^{\rho_E}_{\mathcal{V}}(Tr_{\mathtt{PIPE}})$ both hold. However, for the system UP from Example 3.4.17 and the view $\mathcal{V} = (V, N, C)$ with $V = \{li_n \mid n \in \mathbb{N}\}$, $N = \emptyset$, $C = \{ho_n \mid n \in \mathbb{N}\}$, $IA^{\rho_E}_{\mathcal{V}}(Tr_{\mathtt{UP}})$ holds but $IA^{\rho_C}_{\mathcal{V}}(Tr_{\mathtt{UP}})$ does not hold. ◇

This example suggests that the set returned by ρ should be as large as possible in order to avoid secure systems from being ruled out (e.g. $IA^{\rho_C}_{\mathcal{V}}(Tr_{\mathtt{UP}})$ does not hold for UP but $IA^{\rho_E}_{\mathcal{V}}(Tr_{\mathtt{UP}})$ holds). However, there is a trade-off in the choice of this set because if it is too large then it might remain undetected that deductions about nonoccurrences of confidential events are possible. Apparently, it can be quite difficult to comprehend the impact of a particular choice of ρ on the extent of possible deductions and the consequences for security. The following remark summarizes our previous discussions on this aspect.

Remark 3.4.21 (Impact of ρ-admissibility assumption). Let ρ be a function from views in E to subsets of E. If ES satisfies $IA^{\rho}_{\mathcal{V}}(Tr)$ for a view $\mathcal{V} = (V, N, C)$ then some information about the nonoccurrence of events in C might still be deducible.

In particular, if $IA^{\rho}_{\mathcal{V}}(Tr)$ holds then it is possible to deduce that the subsequence of events in C for any given possible trace is in the set $\{\tau|_C \mid \tau \in Tr\}$. If $\rho(\mathcal{V}) \subseteq C$, $R_{\mathcal{V}}(Tr)$ (or $D_{\mathcal{V}}(Tr)$), and $IA^{\rho}_{\mathcal{V}}(Tr)$ hold then no other information about nonoccurrences of events in C can be deduced (cf. the discussion after Example 3.4.16). However, unlike for $I_{\mathcal{V}}(Tr)$, $\{\tau|_C \mid \tau \in Tr\} = C^*$ is not implied by $IA^{\rho}_{\mathcal{V}}(Tr)$. This means, one can deduce that certain sequences of confidential events cannot occur.

If R and D are not required or $\rho(\mathcal{V}) \subseteq C$ does not hold then $IA^{\rho}_{\mathcal{V}}(Tr)$ does not prevent observations of the dynamic system behavior from being exploited to deduce additional information about nonoccurrences of confidential events (cf. the discussion after Example 3.4.17). For example, if $\rho(\mathcal{V}) \not\subseteq C$ then an observer might be able to learn that some event $c \in C$ cannot occur although $IA^{\rho}_{\mathcal{V}}(Tr)$, $R_{\mathcal{V}}(Tr)$, and $D_{\mathcal{V}}(Tr)$ hold. The reason for this is that an observation in V could reveal information about occurrences of events in $\rho(\mathcal{V}) \setminus C$ (e.g. that an event $e \in \rho(\mathcal{V}) \setminus C$ must have occurred), which together with the knowledge of the specification Tr might be enough (e.g. if each trace $\tau \in Tr$ only contains occurrences of either e or c) to deduce that c is not enabled. ◇

If this is problematic or not heavily depends on the system under consideration. Therefore, when applying IA^{ρ} instead of I one should justify carefully that no critical deductions about nonoccurrences of confidential events can be made despite IA^{ρ} holding. In particular, one should choose the parameter ρ with care.

3.4.4 Backwards-Strict Basic Security Predicates

While the BSPs R, D, I, and IA^{ρ} differ in the perturbations that they require, they all permit the same corrections, i.e. the corrected trace may differ from the perturbed trace arbitrarily in occurrences of events in N. The following example demonstrates that permitting arbitrary corrections of occurrences of events in N sometimes is too liberal.

Example 3.4.22. The system NOISYLEAK is a modified version of LEAK that is defined by the state-event system $\texttt{NOISYLEAK} = (S_{\texttt{NOISYLEAK}}, s_{\texttt{NOISYLEAK}}, E_{\texttt{NOISYLEAK}}, I_{\texttt{LEAK}}, O_{\texttt{LEAK}}, T_{\texttt{NOISYLEAK}})$ where $S_{\texttt{NOISYLEAK}} = \{\mathit{flag} : \mathit{BOOL}, \mathit{var} : \mathbb{N}\}$, $s_{\texttt{NOISYLEAK}} = \{\mathit{flag} \mapsto \mathit{ff}, \mathit{var} \mapsto 0\}$ and $E_{\texttt{NOISYLEAK}} = I_{\texttt{LEAK}} \cup O_{\texttt{LEAK}} \cup \{r_n \mid n \in \mathbb{N}\}$. Hence, NOISYLEAK consists of two state objects: a boolean flag *flag* (initially *ff*, for "false") and a variable *var* (initially 0). For every $n \in \mathbb{N}$, there are three events hi_n, lo_n, and r_n. An occurrence of hi_n models that number n is input to the system from the high-level environment. An occurrence of lo_n models that the number n is output to the low-level environment. An occurrence of event r_n models that a random number n has been chosen internally. The transition relation $T_{\texttt{NOISYLEAK}}$ is defined as follows (also cf. Figure 3.9): hi_n is always enabled (no preconditions). If hi_n occurs then *flag* remains unaffected and *var* is set to n. The event r_n is only enabled if $\mathit{flag} = \mathit{ff}$ holds. If r_n occurs then *flag* is set to *tt* (for "true") and *var* is set to n. lo_n is only enabled if $\mathit{flag} = \mathit{tt}$ and $\mathit{var} = n$ hold. If lo_n occurs then *flag* is reset to *ff* and *var* remains unaffected.

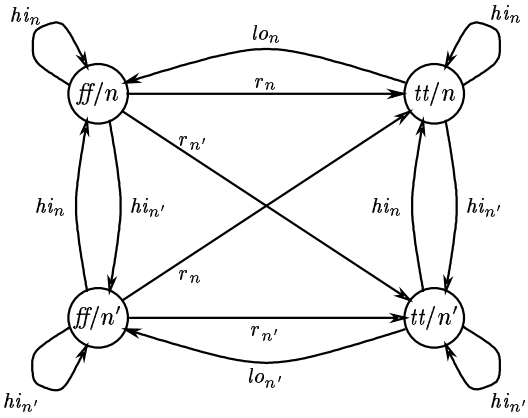


Figure 3.9: The transition relation $T_{\texttt{NOISYLEAK}}$ in Example 3.4.22

According to this specification, NOISYLEAK accepts high-level inputs at any time. If the flag is not set ($\mathit{flag} = \mathit{ff}$) then the high-level input can be overwritten by some random number. However, this happens exactly once before a low-level output because an occurrence of the random event r_n sets the flag ($\mathit{flag} = \mathit{tt}$). The internal variable can only be output if the flag is set ($\mathit{flag} = \mathit{tt}$) and, hence, the variable must have been overwritten by a random number at some prior point of time (after the last output of a number). The occurrence of a low-level output resets the flag ($\mathit{flag} = \mathit{ff}$) in order to enable the random event again and so on.

Let the view $\mathcal{V} = (V, N, C)$ be defined by $V = \{lo_n \mid n \in \mathbb{N}\}$, $N = \{r_n \mid n \in \mathbb{N}\}$, and $C = \{hi_n \mid n \in \mathbb{N}\}$. Let $Tr_{\texttt{NOISYLEAK}} \subseteq E_{\texttt{NOISYLEAK}}{}^*$ be the set of possible traces that is induced by $T_{\texttt{NOISYLEAK}}$. $D_{\mathcal{V}}(Tr_{\texttt{NOISYLEAK}})$, $R_{\mathcal{V}}(Tr_{\texttt{NOISYLEAK}})$, $I_{\mathcal{V}}(Tr_{\texttt{NOISYLEAK}})$, and $IA^{\rho}_{\mathcal{V}}(Tr_{\texttt{NOISYLEAK}})$ are all satisfied (where ρ is arbitrary). Nevertheless a Trojan horse that is part of the high-level environment can transmit information to the low-level environment.

For this purpose, the Trojan horse simply waits for the occurrence of the random event r_0 before it inputs a value to the system (event hi_n) and thereby rules out that the value is overwritten before it is output to the low-level environment. By repeatedly applying this procedure, the Trojan horse can easily transmit a sequence of natural numbers. For example, the transmission of the sequence $\langle 42.13.7 \rangle$ would result in a trace of the following form: (every occurrence of r_* is a place-holder for r_m where $m \in \mathbb{N}$ is arbitrary)

$$\langle r_*.hi_{42}.lo_{42}.r_*.hi_{13}.lo_{13}.r_*.hi_7.lo_7 \rangle \ .$$

Note that the sequence of numbers that is observed on the low level is identical to the sequence that the Trojan horse wanted to transmit. ◇

The above example demonstrates that a Trojan horse might be able to abuse a system for leaking information although this system satisfies $D_{\mathcal{V}}(Tr)$, $R_{\mathcal{V}}(Tr)$, $I_{\mathcal{V}}(Tr)$, and $IA^{\rho}_{\mathcal{V}}(Tr)$ (for arbitrary ρ). Let us investigate more closely at the example of the BSP I why this possibility to leak information remains undetected.

Example 3.4.23. For example, $\langle r_0.lo_0 \rangle$ is a possible trace of `NOISYLEAK` (i.e. $\langle r_0.lo_0 \rangle \in Tr_{\texttt{NOISYLEAK}}$). The trace $\langle r_0.hi_1.lo_0 \rangle$ is a perturbation of $\langle r_0.lo_0 \rangle$ that results from inserting hi_1. This trace is not possible for system `NOISYLEAK`. However, it can be corrected to the trace $\langle hi_1.r_0.lo_0 \rangle$, which is a possible trace (i.e. $\langle hi_1.r_0.lo_0 \rangle \in Tr_{\texttt{NOISYLEAK}}$). Note that the correction involves the deletion of r_0 *before* the occurrence of hi_1 (a *change in the past*) and the insertion of r_0 *after* the occurrence of hi_1 (a *change in the future*). This correction is *not causal* in the sense that it affects events that occur *before* the perturbation (deletion of r_0 before hi_1). Correcting occurrences of events before a perturbation corresponds to changing the *past*, which is not possible without making time warps. This is what the Trojan horse in Example 3.4.22 exploits in order to leak information to the low-level environment. It waits until a random event has occurred before it inputs a number and thereby makes sure that no correction is possible (the random event has already occurred and, hence, cannot be changed any more). ◇

The reason why the insecurity of system `NOISYLEAK` is not detected is that arbitrary corrections (including noncausal ones) are permitted by the BSP. Note that the BSPs R, D, and IA^{ρ} also do not detect this insecurity because they hold for `NOISYLEAK`.[7]

In general, in order to prevent information leakage like in Example 3.4.22, it is necessary to restrict the permitted corrections. Corrections that are *not causal*, i.e. that affect events that occur *before* the perturbation in a trace (changes in the past), should be ruled out. This is the purpose of the basic security predicates *BSD*, *BSI*, and $BSIA^{\rho}$ that result, respectively, from D, I, and IA^{ρ} by restricting the permitted corrections to causal ones. An additional *BS* (for Backwards Strict) in the name of these BSPs indicates that noncausal corrections (i.e. corrections that affect the past) are forbidden by these BSPs.

Definition 3.4.24. Let ρ be a function from views in E to subsets of E and $\mathcal{V} = (V, N, C)$ be a view in E. The basic security predicates *BSD* (Backwards-Strict Deletion), *BSI* (Backwards Strict Insertion), and $BSIA^{\rho}$ (Backwards-Strict Insertion of Admissible events) are formally defined by:

[7] For example, $\langle r_1.lo_0 \rangle$ is a D-perturbation that results from the possible trace $\langle r_1.hi_0.lo_0 \rangle$ by deleting hi_0. The perturbation can be corrected to the possible trace $\langle r_0.lo_0 \rangle$.

$$
\begin{aligned}
BSD_{\mathcal{V}}(Tr) \quad &\equiv \quad \forall \alpha, \beta \in E^*.\, \forall c \in C. \\
&\quad\; [(\beta.\langle c\rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
&\quad\; \Rightarrow \exists \alpha' \in E^*.\, (\beta.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle)] \\
BSI_{\mathcal{V}}(Tr) \quad &\equiv \quad \forall \alpha, \beta \in E^*.\, \forall c \in C. \\
&\quad\; [(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
&\quad\; \Rightarrow \exists \alpha' \in E^*.\, (\beta.\langle c\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle)] \\
BSIA^{\rho}_{\mathcal{V}}(Tr) \quad &\equiv \quad \forall \alpha, \beta \in E^*.\, \forall c \in C. \\
&\quad\; [(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle \wedge Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)) \\
&\quad\; \Rightarrow \exists \alpha' \in E^*.\, (\beta.\langle c\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle)] \qquad \diamond
\end{aligned}
$$

The above definition disallows any adaptations in β, and, in this respect, it differs from the definitions of D, I, and IA^{ρ} (on the right hand side of the implications: β rather than β' occurs). This means, corrections can only occur after the perturbation (i.e. all corrections are causal). All of these BSPs reject the intuitively insecure system `NOISYLEAK`. We leave the detailed check to the reader.

Let us proceed with the definition of further BSPs.

3.4.5 Forward-Correctable Basic Security Predicates

We obtain FCD^{Γ}, FCI^{Γ}, $FCIA^{\rho,\Gamma}$ from BSD, BSI, $BSIA^{\rho}$ by further restricting the permitted corrections and by relaxing the required perturbations. These BSPs are parametric in $\Gamma \in \mathcal{P}(E)^3$, a triple of sets of events that we denote by ∇, Δ, and Υ, respectively. Hence, we obtain three families of BSPs: FCD^{Γ}, $FCD^{\Gamma'}$, ...; FCI^{Γ}, $FCI^{\Gamma'}$, ...; and $FCIA^{\rho,\Gamma}$, $FCIA^{\rho',\Gamma'}$, ... These BSPs perturb possible traces only immediately before the occurrence of visible events.

Definition 3.4.25. Let ρ be a function from views in E to subsets of E, $\mathcal{V} = (V, N, C)$ be a view in E, and $\Gamma = (\nabla, \Delta, \Upsilon)$ be a triple where ∇, Δ, Υ are subsets of E. The basic security predicates FCD^{Γ} (Forward-Correctable Deletion), FCI^{Γ} (Forward-Correctable Insertion), and $FCIA^{\rho,\Gamma}$ (Forward-Correctable Insertion of ρ-Admissible events) are formally defined by:

$$
\begin{aligned}
&FCD^{\Gamma}_{\mathcal{V}}(Tr) \quad \equiv \\
&\quad \forall \alpha, \beta \in E^*.\, \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla. \\
&\quad [(\beta.\langle c.v\rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
&\quad \Rightarrow \exists \alpha' \in E^*.\, \exists \delta' \in (N \cap \Delta)^*.\, (\beta.\delta'.\langle v\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle)] \\
&FCI^{\Gamma}_{\mathcal{V}}(Tr) \quad \equiv \\
&\quad \forall \alpha, \beta \in E^*.\, \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla. \\
&\quad [(\beta.\langle v\rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \\
&\quad \Rightarrow \exists \alpha' \in E^*.\, \exists \delta' \in (N \cap \Delta)^*.\, (\beta.\langle c\rangle.\delta'.\langle v\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle)] \\
&FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr) \quad \equiv \\
&\quad \forall \alpha, \beta \in E^*.\, \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla. \\
&\quad [(\beta.\langle v\rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle \wedge Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)) \\
&\quad \Rightarrow \exists \alpha' \in E^*.\, \exists \delta' \in (N \cap \Delta)^*.\, (\beta.\langle c\rangle.\delta'.\langle v\rangle.\alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\rangle)] \qquad \diamond
\end{aligned}
$$

Let us point out the technical differences between the forward-correctable BSPs and their backwards-strict counterparts using BSD and FCD^{Γ} as an example. Unlike $BSD_{\mathcal{V}}(Tr)$, $FCD^{\Gamma}_{\mathcal{V}}(Tr)$ only requires that confidential events in $C \cap \Upsilon$ that occur immediately before a visible event in $V \cap \nabla$ can be deleted (assumptions $c \in C \cap \Upsilon$, $v \in V \cap \nabla$, $\beta.\langle c.v\rangle.\alpha \in Tr$).

Like *BSD*, FCD^{Γ} disallows corrections in β but permits corrections after the perturbation. However, corrections in between β and v may only involve the insertion of events from $N \cap \Delta$ ($\delta' \in (N \cap \Delta)^*$, $\beta.\delta'.\langle v \rangle.\alpha' \in Tr$). Summarizing, FCD^{Γ} requires *fewer perturbations* and permits *fewer adaptations* than *BSD*. The other pairs of corresponding BSPs, i.e. *BSI* and FCI^{Γ} as well as $BSIA^{\rho}$ and $FCIA^{\rho,\Gamma}$, are related to each other like *BSD* and FCD^{Γ}. Forward-correctable BSPs are mainly of interest for issues of compositionality. Compositionality of information flow properties will be investigated in Chapter 6.

3.4.6 Strict Basic Security Predicates

All BSPs that we have defined so far permit perturbations to be corrected by adapting occurrences of events in N in some way. We now introduce BSPs that do not permit any corrections. These BSPs are very restrictive because they require that every perturbation of a possible trace is again a possible trace (without corrections). An additional S (for Strict) in the name of these BSPs indicates that corrections are completely forbidden.

Definition 3.4.26. Let ρ be a function from views in E to subsets of E and $\mathcal{V} = (V, N, C)$ be a view in E. The basic security predicates *SR* (Strict Removal), *SD* (Strict Deletion), *SI* (Strict Insertion), and SIA^{ρ} (Strict Insertion of ρ-Admissible events) are formally defined by:

$$
\begin{aligned}
SR_{\mathcal{V}}(Tr) &\equiv \forall \tau \in Tr.\, \tau|_{E \setminus C} \in Tr \\
SD_{\mathcal{V}}(Tr) &\equiv \forall \alpha, \beta \in E^*. \forall c \in C. [(\beta.\langle c \rangle.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \Rightarrow \beta.\alpha \in Tr] \\
SI_{\mathcal{V}}(Tr) &\equiv \forall \alpha, \beta \in E^*. \forall c \in C. [(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle) \Rightarrow \beta.\langle c \rangle.\alpha \in Tr] \\
SIA^{\rho}_{\mathcal{V}}(Tr) &\equiv \forall \alpha, \beta \in E^*. \forall c \in C. \\
&\quad [(\beta.\alpha \in Tr \wedge \alpha|_C = \langle\rangle \wedge Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)) \Rightarrow \beta.\langle c \rangle.\alpha \in Tr] \qquad \diamond
\end{aligned}
$$

Since strict BSPs are very restrictive (neither corrections in α nor in β are permitted), they are mostly of theoretical interest.

3.4.7 Other Basic Security Predicates

We have proposed a collection of BSPs that constitute the basic building blocks of *MAKS*. They suffice for the purposes of this thesis. In particular, they suffice for the representation of various known information flow properties. Nevertheless, it might be that further BSPs will be added in the future (for convenience or for theoretical reasons). *MAKS* is defined as an *open framework* that permits such additions.

3.5 Basic Security Predicates in Comparison

In the previous section, we have introduced a collection of BSPs that can be used to assemble security predicates. Two dimensions of BSPs can be distinguished in this collection. BSPs in the first dimension perturb a trace by *removing* or *deleting* occurrences of confidential events. These BSPs can be used to specify that deductions about occurrences of confidential events must not be possible. BSPs in the second dimension perturb a trace by *inserting* occurrences of confidential events. These BSPs can be used to specify that deductions about nonoccurrences of confidential events must not be possible.

When specifying an information flow property, one needs to decide whether it is necessary to rule out that information about occurrences of confidential events can be deduced

and whether it is necessary to rule out that information about nonoccurrences of confidential events can be deduced. Depending on this decision, the security predicate needs to be assembled only from BSPs in the first dimension, only from BSPs in the second dimension, or from BSPs in both dimensions. In this section, we compare the various BSPs that we have proposed to each other and derive a taxonomy of BSPs for each dimension. We start with the comparison of BSPs in the first dimension.

3.5.1 First Dimension of Basic Security Predicates

The BSPs R, D, BSD, SR, SD, and FCD^{Γ} constitute the first dimension in our collection of BSPs. All of these BSPs perturb a given trace by *removing* or by *deleting* occurrences of confidential events. Let us now review these BSPs in more detail.

From the perspective of security, the BSPs R and D are equally well suited to rule out that critical information about occurrences of confidential events can be deduced. Therefore, if the choice is between these two BSPs then one can safely select the less restrictive BSP (i.e. R). However, R and D cannot detect certain possibilities for information leakage, as has been demonstrated by Example 3.4.22. This problem occurs because both of these BSPs permit arbitrary corrections of occurrences of events in N. The culprit is that noncausal corrections are permitted, i.e. corrections that may affect occurrences of events in a given trace before the point where the actual perturbation occurs. The BSPs BSD, SR, and SD do not permit noncausal corrections and, hence, they do not suffer from the problem pointed out in Example 3.4.22. The strict BSPs SR and SD do not even permit any corrections at all. However, forbidding corrections completely is quite restrictive and causal corrections do not cause any problems wrt. undetected possibilities of information leakage (at least no such problems are known to date). Therefore, if the choice is between BSD, SR, and SD then one can safely select the BSP that permits at least some corrections (i.e. BSD). Forward correctable BSPs should only be used in combination with another BSP from the same dimension. The motivation of using a forward correctable BSP in the definition of an information flow property is to obtain a composable property. Compositionality of information flow properties will be discussed in Chapter 6.

In summary, BSD appears to be superior to all other BSPs in the first dimension. R and D are too liberal for certain applications. SR and SD are unnecessarily restrictive. However, this does not mean that, when specifying an information flow property, BSD is the only sensible choice of a BSP in the first dimension. For example, if information leakage like in Example 3.4.22 cannot occur with the system under consideration[8] then R or D could be used instead of BSD. Moreover, as we will show in Chapter 4, all of R, D, BSD, SR, SD, and FCD^{Γ} correspond to basic ingredients of previously proposed information flow properties.

In our discussion, we stated that some BSPs are more restrictive than others. Let us now justify these statements by deriving a taxonomy of BSPs in the first dimension.

Theorem 3.5.1 (Ordering BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E. The following implications are valid:

1. $D_{\mathcal{V}}(Tr) \Rightarrow R_{\mathcal{V}}(Tr)$,
2. $BSD_{\mathcal{V}}(Tr) \Rightarrow D_{\mathcal{V}}(Tr)$,

[8]For instance, because no program in the high-level environment can observe occurrences of the random events r_n.

3. $SD_{\mathcal{V}}(Tr) \Rightarrow BSD_{\mathcal{V}}(Tr)$,
4. $SR_{\mathcal{V}}(Tr) \Rightarrow R_{\mathcal{V}}(Tr)$, and
5. $SD_{\mathcal{V}}(Tr) \Rightarrow SR_{\mathcal{V}}(Tr)$. ◇

Proof. The validity of implications 2, 3, and 4 follows immediately from the definitions of the involved BSPs because all corrections that comply with the definition of a BSP on the left hand side of these implications also comply with the definition of the respective BSP on the right hand side. For proving 2, choose $\beta' = \beta$; for proving 3, choose $\alpha' = \alpha$; for proving 4, choose $\tau' = \tau$ in the definition of the respective BSP on the right hand side.

The validity of implications 1 and 5 can be shown by a simple inductive argument. As an example, we prove $D_{\mathcal{V}}(Tr) \Rightarrow R_{\mathcal{V}}(Tr)$ in detail. Assume that $D_{\mathcal{V}}(Tr)$ holds. Moreover, assume $C \neq \emptyset$ because, otherwise, $R_{\mathcal{V}}(Tr)$ would hold trivially. Let $\tau \in Tr$ be arbitrary. By induction on the length of $\tau|_C$ we prove that there is a trace $\tau' \in Tr$ for which $\tau'|_C = \langle\rangle$ and $\tau'|_V = \tau|_V$ hold: In the base case, $\tau|_C = \langle\rangle$ holds and the proposition follows with the choice $\tau' = \tau$. In the induction step, there are $n > 0$ occurrences of events in C in τ and τ can be represented by $\beta.\langle c\rangle.\alpha$ where $c \in C$ and $\alpha|_C = \langle\rangle$, i.e. c is the last occurrence of an event in C. From $D_{\mathcal{V}}(Tr)$ we obtain that there are traces $\alpha', \beta' \in E^*$ with $\beta'.\alpha' \in Tr$, $\alpha'|_C = \langle\rangle$, $\alpha'|_V = \alpha|_V$, and $\beta'|_{V \cup C} = \beta|_{V \cup C}$. Since $\beta'.\alpha'$ has $n-1$ occurrences of events in C, the proposition follows from the induction assumption. The argument for the fifth implication is along the same lines. □

Theorem 3.5.2 (Ordering BSPs). Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E for which $V_2 \subseteq V_1$, $C_2 \subseteq C_1$, $N_2 \supseteq N_1$ hold. The following implications are valid:

1. $R_{\mathcal{V}_1}(Tr) \Rightarrow R_{\mathcal{V}_2}(Tr)$
2. $D_{\mathcal{V}_1}(Tr) \Rightarrow D_{\mathcal{V}_2}(Tr)$
3. $BSD_{\mathcal{V}_1}(Tr) \Rightarrow BSD_{\mathcal{V}_2}(Tr)$ ◇

Proof. The validity of implication 1 follows immediately from the definition of R.

The validity of implications 2 and 3 can be shown by induction. As an example, we prove $BSD_{\mathcal{V}_1}(Tr) \Rightarrow BSD_{\mathcal{V}_2}(Tr)$ in detail. Assume that $BSD_{\mathcal{V}_1}(Tr)$ holds. Moreover, assume that $C_2 \neq \emptyset$ because, otherwise, $BSD_{\mathcal{V}_2}(Tr)$ would hold trivially. Let $\beta.\langle c\rangle.\alpha \in Tr$ be arbitrary with the restriction that $c \in C_2$ and $\alpha|_{C_2} = \langle\rangle$ hold. By induction on the length of $\alpha|_{C_1}$ we prove that there is a trace $\alpha' \in E^*$ with $\beta.\alpha' \in Tr$, $\alpha'|_{C_1} = \langle\rangle$ (implies $\alpha'|_{C_2} = \langle\rangle$), $\alpha'|_{V_1} = \alpha|_{V_1}$ (implies $\alpha'|_{V_2} = \alpha|_{V_2}$): In the base case, $\alpha|_{C_1} = \langle\rangle$ holds and we obtain from $BSD_{\mathcal{V}_1}(Tr)$ that there is a trace $\alpha' \in E^*$ with $\beta.\alpha' \in Tr$, $\alpha'|_{C_1} = \langle\rangle$, and $\alpha'|_{V_1} = \alpha|_{V_1}$ ($c \in C_1$ holds because $C_2 \subseteq C_1$). Thus, the proposition holds in the base case. In the step case, there are $n > 0$ occurrences of events from C_1 in α. Thus, α can be represented by $\gamma.\langle c'\rangle.\delta$ such that $c' \in C_1$ and $\delta|_{C_1} = \langle\rangle$. We obtain from $BSD_{\mathcal{V}_1}(Tr)$ that there is a trace $\delta' \in E^*$ such that $\beta.\langle c\rangle.\gamma.\delta' \in Tr$, $\delta'|_{C_1} = \langle\rangle$, and $\delta'|_{V_1} = \delta|_{V_1}$. Since $(\beta.\langle c\rangle.\gamma.\delta)|_{V_1} = (\beta.\langle c\rangle.\alpha)|_{V_1}$ and since $\beta.\langle c\rangle.\gamma.\delta'$ has $n-1$ occurrences of events in C_1, the proposition follows from the induction assumption. The argument for implication 2 is along the same lines. □

Theorems 3.5.1 and 3.5.2 give rise to the following theorem, which establishes a taxonomy of BSPs in the first dimension. Together with the corresponding theorem for the second dimension (Theorem 3.5.12), it constitutes the main contribution of the current section.

Theorem 3.5.3 (Taxonomy first dimension). Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E for which $V_2 \subseteq V_1$, $C_2 \subseteq C_1$, $N_2 \supseteq N_1$ hold. The BSPs in the first dimension are ordered by implication like depicted in Figure 3.10. ◇

Proof. The proposition follows from Theorems 3.5.1 and 3.5.2. □

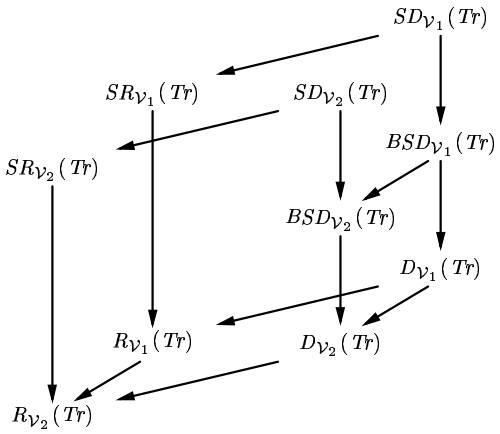


Figure 3.10: Ordering BSPs in first dimension according to Theorem 3.5.3

The following two theorems show how FCD^{Γ} can be integrated into the taxonomy.

Theorem 3.5.4 (Ordering BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E. Let $\Gamma = (\nabla, \Delta, \Upsilon)$ where ∇, Δ, Υ are subsets of E.

If $SD_{\mathcal{V}}(Tr)$ then $FCD^{\Gamma}_{\mathcal{V}}(Tr)$ holds. ◇

Proof. The implication follows immediately from the definitions of SD and FCD^{Γ}. □

Theorem 3.5.5 (Ordering BSPs). Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E for which $V_2 \subseteq V_1$, $C_2 = C_1$, $N_2 \supseteq N_1$ hold. Let $\Gamma_1 = (\nabla_1, \Delta_1, \Upsilon_1)$ and $\Gamma_2 = (\nabla_2, \Delta_2, \Upsilon_2)$ where $\nabla_1, \Delta_1, \Upsilon_1, \nabla_2, \Delta_2, \Upsilon_2$ are subsets of E for which $V_2 \cap \nabla_2 \subseteq V_1 \cap \nabla_1$, $C_2 \cap \Upsilon_2 \subseteq C_1 \cap \Upsilon_1$, and $N_2 \cap \Delta_2 \supseteq N_1 \cap \Delta_1$ hold.

If $FCD^{\Gamma_1}_{\mathcal{V}_1}(Tr)$ then $FCD^{\Gamma_2}_{\mathcal{V}_2}(Tr)$ holds. ◇

Proof. Assume $FCD^{\Gamma_1}_{\mathcal{V}_1}(Tr)$, $\beta.\langle c.v\rangle.\alpha \in Tr$, $c \in C_2 \cap \Upsilon_2$, $v \in V_2 \cap \nabla_2$, and $\alpha|_{C_2} = \langle\rangle$. From $C_2 \cap \Upsilon_2 \subseteq C_1 \cap \Upsilon_1$, we obtain $c \in C_1 \cap \Upsilon_1$. From $V_2 \cap \nabla_2 \subseteq V_1 \cap \nabla_1$, we obtain $v \in V_1 \cap \nabla_1$. From $C_2 = C_1$, we obtain $\alpha|_{C_1} = \langle\rangle$. Hence, $FCD^{\Gamma_1}_{\mathcal{V}_1}(Tr)$ is applicable and an application yields that there are $\alpha', \delta' \in E^*$ for which $\beta.\delta'.\langle v\rangle.\alpha' \in Tr$, $\alpha'|_{V_1} = \alpha|_{V_1}$, $\alpha'|_{C_1} = \langle\rangle$, and $\delta' \in (N_1 \cap \Delta_1)^*$. From $V_2 \subseteq V_1$, we obtain $\alpha'|_{V_2} = \alpha|_{V_2}$. From $C_2 = C_1$, we obtain $\alpha'|_{C_2} = \langle\rangle$. From $N_2 \cap \Delta_2 \supseteq N_1 \cap \Delta_1$, we obtain $\delta' \in (N_2 \cap \Delta_2)^*$. Consequently, $FCD^{\Gamma_2}_{\mathcal{V}_2}(Tr)$ holds. □

Let us now investigate the effects of specific views on the validity of BSPs in the first dimension. Firstly, let us assume that $N = \emptyset$ holds. Under this assumption there is no difference between the strict version, the backwards-strict version, and the non-strict version of a BSP. This is generalized by the following theorem.

Theorem 3.5.6. Let $\mathcal{V}_1 = (V_1, N_1, C_1)$ and $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E. If $V_2 = V_1 \cup N_1$, $N_2 = \emptyset$, and $C_2 = C_1$ hold then the following equivalences are valid:

1. $SR_{\mathcal{V}_1}(Tr) \Leftrightarrow R_{\mathcal{V}_2}(Tr)$
2. $SD_{\mathcal{V}_1}(Tr) \Leftrightarrow BSD_{\mathcal{V}_2}(Tr)$
3. $BSD_{\mathcal{V}_1}(Tr) \Leftrightarrow D_{\mathcal{V}_2}(Tr)$ ◇

Proof. All equivalences follow immediately from the definitions of the respective BSPs. □

If there are no confidential events or no visible events in the given view then most BSPs in the first dimension are trivially fulfilled. This is summarized by the following theorem.

Theorem 3.5.7 (Trivially-fulfilled BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E. The following statements are valid:

1. If $C = \emptyset$ then $R_{\mathcal{V}}(Tr)$, $D_{\mathcal{V}}(Tr)$, $BSD_{\mathcal{V}}(Tr)$, $SR_{\mathcal{V}}(Tr)$, $SD_{\mathcal{V}}(Tr)$, and $FCD^{\Gamma}_{\mathcal{V}}(Tr)$ hold.
2. If $V = \emptyset$ then $R_{\mathcal{V}}(Tr)$, $D_{\mathcal{V}}(Tr)$, $BSD_{\mathcal{V}}(Tr)$, and $FCD^{\Gamma}_{\mathcal{V}}(Tr)$ hold.

Proof. All propositions follow immediately from the definitions of the respective BSPs. □

For certain choices of parameters, FCD^{Γ} holds trivially or follows from *BSD*. These facts about the forward correctable BSP will be of interest in Chapter 6.

Theorem 3.5.8 (Trivially-fulfilled BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E and $\Gamma = (\nabla, \Delta, \Upsilon)$ where ∇, Δ, Υ are subsets of E. The following statements are valid:

1. If $\nabla = \emptyset$ or $\Upsilon = \emptyset$ then $FCD^{\Gamma}_{\mathcal{V}}(Tr)$ holds.
2. If $N \subseteq \Delta$ then $BSD_{\mathcal{V}}(Tr)$ implies $FCD^{\Gamma}_{\mathcal{V}}(Tr)$. ◇

Proof. Both propositions follow immediately from the definitions of the respective BSPs. □

3.5.2 Second Dimension of Basic Security Predicates

The BSPs *I*, *BSI*, *SI*, FCI^{Γ}, IA^{ρ}, $BSIA^{\rho}$, SIA^{ρ}, and $FCIA^{\rho,\Gamma}$ constitute the second dimension of our collection of BSPs. All of these BSPs perturb a given trace by *inserting* occurrences of confidential events. Thus, they prevent deductions about nonoccurrences of confidential events. Let us now review these BSPs in more detail.

The BSPs *I*, *BSI*, *SI*, and FCI^{Γ} are related to each other similarly to how *D*, *BSD*, *SD*, and FCD^{Γ} are related to each other (in the first dimension). Hence, *I* permits arbitrary corrections and, hence, information leakage like in Example 3.4.22 cannot be detected by this BSP. *BSI* and *SI* do not suffer from this problem because they do not permit noncausal corrections. *SI* does not even permit any corrections at all. However, forbidding corrections completely is quite restrictive and causal corrections do not cause any problems wrt. undetected possibilities of information leakage (at least no such problems are known to date). Therefore, if the choice is between *BSI* and *SI* then one can safely select the BSP that permits at least some corrections (i.e. *BSI*). The forward correctable BSP, i.e. FCI^{Γ}, should always be used in combination with another BSP. This is like in the first dimension.

One shortcoming of *I*, *BSI*, and *SI* is that these BSPs only hold for systems for which all sequences of confidential events are possible. This is a very restrictive requirement because it implies that systems must behave chaotically in the confidential events and, hence, the sequences of confidential events cannot model meaningful computations. Obviously, there are

systems that do not satisfy this requirement although they are intuitively secure (e.g. system PIPE in Example 3.4.15 or system UP in Example 3.4.17). To overcome this shortcoming was the motivation for introducing the BSPs IA^{ρ}, $BSIA^{\rho}$, SIA^{ρ}, and $FCIA^{\rho,\Gamma}$, which do not require that all sequences of confidential events must be possible but, otherwise, are identical to *I*, *BSI*, *SI*, and FCI^{Γ}, respectively.

IA^{ρ}, $BSIA^{\rho}$, SIA^{ρ}, and $FCIA^{\rho,\Gamma}$ only require the insertion of confidential events at points in a trace where their occurrence "makes sense". What it means to "make sense" is defined by the functional parameter ρ that, given a view $\mathcal{V}$, returns some set of events. Examples are the function ρ_C that returns the set of confidential events (i.e. $\rho_C(V, N, C) = C$) or the function ρ_E that returns the set of all events (i.e. $\rho_E(V, N, C) = V \cup N \cup C = E$). For a trace $\beta.\alpha \in Tr$ (with $\alpha|_C = \langle\rangle$), e.g., $SIA^{\rho}_{\mathcal{V}}(Tr)$ only requires that an event $c \in C$ can be inserted in between β and α if there is a trace $\gamma.\langle c\rangle \in Tr$ such that $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$ holds. That $\gamma.\langle c\rangle$ is a possible trace shows that the sequence $\gamma|_{\rho(\mathcal{V})}.\langle c\rangle$ "makes sense" and, hence, also $\beta|_{\rho(\mathcal{V})}.\langle c\rangle$ "makes sense" ($\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$ holds). If there is no such sequence, i.e. the occurrence of c does not "make sense" at this point, then $SIA^{\rho}_{\mathcal{V}}(Tr)$ does not require the insertion of this event at this point. For example, for the parameter ρ_C, the event c need only be insertable in between β and α if the sequence $\beta|_C.\langle c\rangle$ makes sense, i.e. if there is some possible trace that has the same projection to C like $\beta.\langle c\rangle$. This means, although $SIA^{\rho_C}_{\mathcal{V}}(Tr)$ holds, the enabledness of confidential events may depend on previous occurrences of confidential events. Having dependencies between occurrences of events in C implies that there might be information flow from C to C. Obviously, information flow within C is not problematic. If the parameter ρ_E is chosen instead then the enabledness of events in C may also depend on previous occurrences of events in V and N. Having information flow from V and N to C is also non-critical. We only want to prevent that information flows from C to V but do not want to limit information flow in the other direction. In general, the *larger* the set $\rho(\mathcal{V})$ is, the more non-critical information flow is permitted.

Unfortunately, the side effect of permitting non-critical information flow is that also some information leakage from C to V becomes possible. This means, IA^{ρ}, $BSIA^{\rho}$, and SIA^{ρ} may fail to detect that it is possible to deduce information about nonoccurrences although *I*, *BSI*, and *SI* would detect this possibility. This has been illustrated by the system LEAKONCE in Example 3.4.16. Moreover, IA^{ρ}, $BSIA^{\rho}$, and SIA^{ρ} may fail to detect a possibility to leak information for one choice of the parameter ρ that they would detect with a different choice of this parameter. In general, the *smaller* the set $\rho(\mathcal{V})$ is, the fewer possibilities for information leakage will remain undetected. However, even if the parameter $\rho_{\emptyset}$ (defined by $\rho_{\emptyset}(\mathcal{V}) = \emptyset$) is chosen, an observer can deduce that certain confidential events cannot have occurred (i.e. the confidential events that are never enabled).[9] Usually, this possibility to deduce information about nonoccurrences of confidential events will not be problematic. However, this depends on the system under consideration. If the parameter ρ_C is chosen instead of $\rho_{\emptyset}$ then even more information can be deduced about nonoccurrences of confidential events (cf. Example 3.4.16). However, if a corresponding BSP from the first dimension is enforced in addition to IA^{ρ}, $BSIA^{\rho}$, or SIA^{ρ} then the possibility to deduce information is quite limited. In this case, an observer can only deduce that certain *sequences* of confidential events cannot occur (i.e. the sequences of confidential events that are impossible, in general). Again, this possibility to deduce information about nonoccurrences of confidential events

[9]Note that *I*, *BSI*, and *SI* do not hold for systems where some confidential events are never enabled while IA^{ρ}, $BSIA^{\rho}$, and SIA^{ρ} may hold for such systems.

usually will not be problematic. In both of these cases (i.e. parameter $\rho_\emptyset$ or parameter ρ_C in combination with a corresponding BSP from the first dimension), an observer does not obtain additional information when the system runs. The information that he can obtain is limited to what can already be deduced statically from the system specification. For a different choice of the parameter ρ, this is not true anymore. More specifically, if $\rho(\mathcal{V})$ contains events in N or V then the observer might be capable to deduce additional information from his observations of the dynamic system behavior (cf. Remark 3.4.21). To what extent deductions are possible and whether they are problematic or not depends on the system under consideration.

In summary, the backwards-strict BSPs *BSI* and $BSIA^\rho$ appear superior to all other BSPs in the second dimension. The non-strict BSPs *I* and IA^ρ are too liberal for certain applications. The strict BSPs *SI* and SIA^ρ are unnecessarily restrictive. However, this does not mean that all BSPs except for *BSI* and $BSIA^\rho$ are useless. As we will show in Chapter 4, *I*, IA^ρ, and SIA^ρ also correspond to basic ingredients of previously proposed information flow properties. When choosing between *BSI* and $BSIA^\rho$ the only choice that definitely prevents all deductions about nonoccurrences of confidential events is *BSI*. In other words, preferring *BSI* over $BSIA^\rho$ is always safe. However, for certain systems that intuitively are secure, *BSI* is a too restrictive requirement. In these cases, one has to choose $BSIA^\rho$. However, if $BSIA^\rho$ is chosen, one has to be very careful that no possibility to deduce critical information about nonoccurrences of confidential events remains undetected. In order to limit the deductions that are not detected by $BSIA^\rho$, it is advisable to enforce a corresponding BSP from the first dimension in addition (i.e. *BSD*). Moreover, when choosing the parameter ρ, one has to pay attention to the trade-off between permitting non-critical information flow from V, N, and C to C (favoring a *large* set $\rho(\mathcal{V})$) and ruling out potentially critical information flow (favoring a *small* set $\rho(\mathcal{V})$). In general, there is no choice of ρ that is best for all purposes.

Let us now derive a taxonomy of BSPs in the second dimension.

Theorem 3.5.9 (Ordering BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E and ρ be a function from views in E to subsets of E. The following implications are valid:

1. $SI_\mathcal{V}(Tr) \Rightarrow BSI_\mathcal{V}(Tr)$,
2. $BSI_\mathcal{V}(Tr) \Rightarrow I_\mathcal{V}(Tr)$,
3. $SIA^\rho_\mathcal{V}(Tr) \Rightarrow BSIA^\rho_\mathcal{V}(Tr)$,
4. $BSIA^\rho_\mathcal{V}(Tr) \Rightarrow IA^\rho_\mathcal{V}(Tr)$,
5. $SI_\mathcal{V}(Tr) \Rightarrow SIA^\rho_\mathcal{V}(Tr)$,
6. $BSI_\mathcal{V}(Tr) \Rightarrow BSIA^\rho_\mathcal{V}(Tr)$, and
7. $I_\mathcal{V}(Tr) \Rightarrow IA^\rho_\mathcal{V}(Tr)$. ◇

Proof. The validity of implications 1–4 follows immediately from the definitions of the involved BSPs because all corrections that comply with the definition of a BSP on the left hand side of these implications also comply with the definition of the respective BSP on the right hand side. For proving 1 or 3, choose $\alpha' = \alpha$; for proving 2 or 4, choose $\beta' = \beta$ in the definition of the respective BSP on the right hand side.

Implications 5–7 hold because the BSPs on the right hand side of these implications differ from the ones on the left hand side only in that they have an additional assumption, i.e. $Adm^\rho_\mathcal{V}(Tr, \beta, c)$. □

Theorem 3.5.10 (Ordering BSPs). Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E for which $V_2 \subseteq V_1$, $C_2 = C_1$, $N_2 \supseteq N_1$ hold and ρ_1, ρ_2 be functions from views in E to subsets of E for which $\rho_1(\mathcal{V}_1) \subseteq \rho_2(\mathcal{V}_2)$ holds. The following implications are valid:

1. $I_{\mathcal{V}_1}(Tr) \Rightarrow I_{\mathcal{V}_2}(Tr)$,
2. $BSI_{\mathcal{V}_1}(Tr) \Rightarrow BSI_{\mathcal{V}_2}(Tr)$,
3. $SI_{\mathcal{V}_1}(Tr) \Rightarrow SI_{\mathcal{V}_2}(Tr)$,
4. $IA^{\rho_1}_{\mathcal{V}_1}(Tr) \Rightarrow IA^{\rho_2}_{\mathcal{V}_2}(Tr)$,
5. $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr) \Rightarrow BSIA^{\rho_2}_{\mathcal{V}_2}(Tr)$, and
6. $SIA^{\rho_1}_{\mathcal{V}_1}(Tr) \Rightarrow SIA^{\rho_2}_{\mathcal{V}_2}(Tr)$. ◇

Before proving Theorem 3.5.10, let us introduce a lemma that is helpful in that proof.

Lemma 3.5.11. Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E, $e \in E$, and $\beta \in Tr$. For any two functions ρ_1, ρ_2 from views in E to subsets of E with $\rho_1(\mathcal{V}_1) \subseteq \rho_2(\mathcal{V}_2)$ holds $Adm^{\rho_2}_{\mathcal{V}_2}(Tr, \beta, e) \Rightarrow Adm^{\rho_1}_{\mathcal{V}_1}(Tr, \beta, e)$. ◇

Proof. If $Adm^{\rho_2}_{\mathcal{V}_2}(Tr, \beta, e)$ and $\beta \in Tr$ then, according to Definition 3.4.18, there is a trace $\gamma \in E^*$ for which $\gamma.\langle e \rangle \in Tr$ and $\gamma|_{\rho_2(\mathcal{V}_2)} = \beta|_{\rho_2(\mathcal{V}_2)}$ hold. From $\rho_1(\mathcal{V}_1) \subseteq \rho_2(\mathcal{V}_2)$, we obtain $\gamma|_{\rho_1(\mathcal{V}_1)} = \beta|_{\rho_1(\mathcal{V}_1)}$. Consequently, $Adm^{\rho_1}_{\mathcal{V}_1}(Tr, \beta, e)$ holds. □

Proof (of Theorem 3.5.10). Implications 1–3 hold because the only requirements concerning the sets N and V in the definitions of the involved BSPs are $\beta'|_V = \beta|_V$ and $\alpha'|_V = \alpha|_V$; these formulas occur only on the right hand side of the implication in each of these definitions (positive polarity); and, hence, changing the view by moving elements from V to N cannot make any of *I*, *BSI*, *SI* false. The same argument can be applied to prove implications 4–6 where Lemma 3.5.11 is used in order to deal with ρ-admissibility. Note that the assumption $C_1 = C_2$ is essential for the validity of these implications. □

Theorems 3.5.9 and 3.5.10 give rise to the following theorem. This theorem establishes a taxonomy of BSPs in the second dimension. Together with the corresponding theorem for the first dimension (Theorem 3.5.3), it constitutes the main contribution of the current section.

Theorem 3.5.12 (Taxonomy second dimension). Let $\mathcal{V}_1 = (V_1, N_1, C_1)$ and let $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E for which $V_2 \subseteq V_1$, $C_2 = C_1$, $N_2 \supseteq N_1$ hold. Let ρ_1, ρ_2 be functions from views in E to subsets of E for which $\rho_1(\mathcal{V}_1) \subseteq \rho_2(\mathcal{V}_2)$ holds. The BSPs in the second dimension are ordered by implication like depicted in Figure 3.11. ◇

Proof. The proposition follows from Theorems 3.5.9 and 3.5.10. □

The following two theorems integrate FCI^{Γ} and $FCIA^{\rho,\Gamma}$ into the taxonomy.

Theorem 3.5.13 (Ordering BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E, ρ be a function from views in E to subsets of E, and $\Gamma = (\nabla, \Delta, \Upsilon)$ where ∇, Δ, Υ are subsets of E. The following implications are valid:

1. $SI_{\mathcal{V}}(Tr) \Rightarrow FCI^{\Gamma}_{\mathcal{V}}(Tr)$,
2. $SIA^{\rho}_{\mathcal{V}}(Tr) \Rightarrow FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$, and

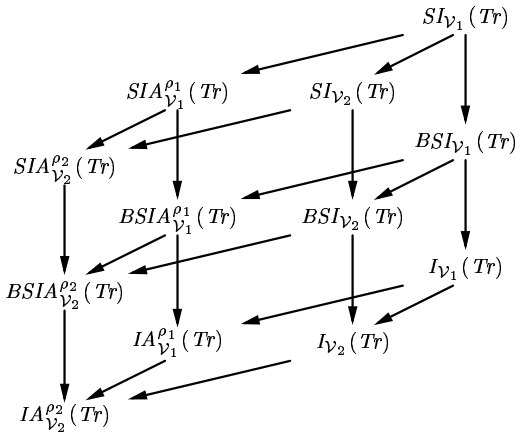


Figure 3.11: Ordering BSPs in second dimension according to Theorem 3.5.12

3. $FCI^{\Gamma}_{\mathcal{V}}(Tr) \Rightarrow FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$. ◇

Proof. The implications follow immediately from the definitions of SI, SIA^{ρ}, FCI^{Γ}, and $FCIA^{\rho,\Gamma}$. □

Theorem 3.5.14 (Ordering BSPs). Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E for which $V_2 \subseteq V_1$, $C_2 = C_1$, $N_2 \supseteq N_1$ hold, ρ_1, ρ_2 be functions from views in E to subsets of E for which $\rho_1(\mathcal{V}_1) \subseteq \rho_2(\mathcal{V}_2)$ holds, $\Gamma_1 = (\nabla_1, \Delta_1, \Upsilon_1)$, and $\Gamma_2 = (\nabla_2, \Delta_2, \Upsilon_2)$ where $\nabla_1, \Delta_1, \Upsilon_1, \nabla_2, \Delta_2, \Upsilon_2$ are subsets of E for which $V_2 \cap \nabla_2 \subseteq V_1 \cap \nabla_1$, $C_2 \cap \Upsilon_2 \subseteq C_1 \cap \Upsilon_1$, and $N_2 \cap \Delta_2 \supseteq N_1 \cap \Delta_1$ hold. The following implications are valid:

1. $FCI^{\Gamma_1}_{\mathcal{V}_1}(Tr) \Rightarrow FCI^{\Gamma_2}_{\mathcal{V}_2}(Tr)$ and
2. $FCIA^{\rho_1,\Gamma_1}_{\mathcal{V}_1}(Tr) \Rightarrow FCIA^{\rho_2,\Gamma_2}_{\mathcal{V}_2}(Tr)$. ◇

Proof. The implications follow from the definitions of FCI^{Γ} and $FCIA^{\rho,\Gamma}$. □

Let us now investigate the effects of specific views on the validity of BSPs in the second dimension. Firstly, let us assume that $N = \emptyset$ holds. Under this assumption there is no difference between the strict version, the backwards-strict version, and the non-strict version of a BSP. This is generalized by the following theorem.

Theorem 3.5.15. Let $\mathcal{V}_1 = (V_1, N_1, C_1)$, $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E and ρ_1, ρ_2 be functions from views in E to subsets of E. If $V_2 = V_1 \cup N_1$, $N_2 = \emptyset$, $C_2 = C_1$, and $\rho_1(\mathcal{V}_1) = \rho_2(\mathcal{V}_2)$ hold then the following equivalences are valid:

1. $SI_{\mathcal{V}_1}(Tr) \Leftrightarrow BSI_{\mathcal{V}_2}(Tr)$,
2. $SIA^{\rho_1}_{\mathcal{V}_1}(Tr) \Leftrightarrow BSIA^{\rho_2}_{\mathcal{V}_2}(Tr)$,
3. $BSI_{\mathcal{V}_1}(Tr) \Leftrightarrow I_{\mathcal{V}_2}(Tr)$, and
4. $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr) \Leftrightarrow IA^{\rho_2}_{\mathcal{V}_2}(Tr)$. ◇

Proof. All equivalences follow immediately from the definitions of the respective BSPs. □

If there are no confidential events or no visible events then most BSPs in the second dimension are trivially fulfilled. This is summarized in the following theorem.

Theorem 3.5.16 (Trivially-fulfilled BSPs). Let ρ be a function from views in E to subsets of E and $\mathcal{V} = (V, N, C)$ be a view in E. The following statements are valid:

1. If $C = \emptyset$ then $I_{\mathcal{V}}(Tr)$, $IA^{\rho}_{\mathcal{V}}(Tr)$, $BSI_{\mathcal{V}}(Tr)$, $BSIA^{\rho}_{\mathcal{V}}(Tr)$, $SI_{\mathcal{V}}(Tr)$, $SIA^{\rho}_{\mathcal{V}}(Tr)$, $FCI^{\Gamma}_{\mathcal{V}}(Tr)$, and $FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$ hold.
2. (a) If $V = \emptyset$ then $FCI^{\Gamma}_{\mathcal{V}}(Tr)$ and $FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$ hold.
   (b) If $V = \emptyset$ and $\rho(\mathcal{V}) \supseteq C \cup N$ then $IA^{\rho}_{\mathcal{V}}(Tr)$ and $BSIA^{\rho}_{\mathcal{V}}(Tr)$ hold.
   (c) If $V = \emptyset$ and $total(ES, C)$ then $I_{\mathcal{V}}(Tr)$ and $BSI_{\mathcal{V}}(Tr)$ hold.

For certain choices of parameters, FCI^{Γ} and $FCIA^{\rho,\Gamma}$ hold trivially or follow from *BSI*. These facts about the forward correctable BSP will be of interest in Chapter 6.

Theorem 3.5.17 (Trivially-fulfilled BSPs). Let $\mathcal{V} = (V, N, C)$ be a view in E, ρ be a function from views in E to subsets of E, and $\Gamma = (\nabla, \Delta, \Upsilon)$ where ∇, Δ, Υ are subsets of E. The following statements are valid:

1. If $\nabla = \emptyset$ or $\Upsilon = \emptyset$ then $FCI^{\Gamma}_{\mathcal{V}}(Tr)$ and $FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$ hold.
2. If $N \subseteq \Delta$ then $BSI_{\mathcal{V}}(Tr)$ implies $FCI^{\Gamma}_{\mathcal{V}}(Tr)$ and $BSIA^{\rho}_{\mathcal{V}}(Tr)$ implies $FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$. ◇

Proof. These propositions follow immediately from the definitions of the respective BSPs. □

3.6 Summary

In this chapter, we have proposed *MAKS*, the Modular Assembly Kit for Security Properties, which is a framework for the *uniform* representation of information flow properties.

In *MAKS*, every information flow property is represented by two components: a set of views and a security predicate. The purpose of the *set of views* is to specify the security requirements of a given application. Each view $\mathcal{V} = (V, N, C)$ in this set specifies a particular restriction on the permitted flow of information from the perspective of a (potentially malicious) observer: observing occurrences of events in V (i.e. the visible events) may not reveal any information about which events in C (i.e. the confidential events) have or have not occurred. Sets of views can be specified by flow policies using an intuitive graphical notation (cf. Section 3.3). Our notion of flow policies is based on the three relations $\rightsquigarrow_V$, $\rightsquigarrow_N$ and $\not\rightsquigarrow$, which allows one to express all sensible relationships between security domains wrt. visibility and confidentiality. This was not possible with the traditional approach to specifying flow policies, which was based on only two relations. The purpose of the *security predicate* is to define formally what it means that there is no information flow, i.e. under which conditions does a system satisfy the restrictions specified by a given view. A key novelty of *MAKS*, in comparison to prior frameworks [McL94a, FG95, McL96, Zak96, ZL97], is that information flow properties are specified in a modular way: every security predicate is specified by a

conjunction of a nonempty set of BSPs. This modular representation of security predicates provides a basis for reducing complex reasoning about information flow properties to reasoning about quite simple BSPs. In the following chapters, we will extensively exploit this possibility in order to simplify our investigation of information flow properties.

Each concrete BSP has been defined by a quite simple closure condition on sets of traces: after *perturbing* a possible trace (by modifying occurrences of events in C) it must be possible to *correct* this perturbation (by adapting occurrences of events in N) to a possible trace. Since occurrences of visible events (i.e. events in V) are left unchanged, a BSP ensures that so many traces possibly could have generated a given observation that it is impossible for an adversary to deduce confidential information from his observations. The name of a BSP indicates the required perturbation: if all occurrences of confidential events are removed then an R occurs in the name; if the last occurrence is deleted then a D occurs in the name; and if an occurrence of a confidential event is inserted then an I occurs in the name. If the insertion of confidential events is only required at positions where they are ρ-admissible then the I is followed by a A^ρ (resulting in IA^ρ). BSPs that perturb traces by removing or deleting confidential events prevent the observer from deducing information about *occurrences* of confidential events, and BSPs that perturb traces by inserting confidential events prevent him from deducing information about *nonoccurrences* of confidential events. Hence, in our collection of BSPs, preventing information flow amounts to preventing deductions about occurrences and nonoccurrences of confidential events. This suffices to represent the well known information flow properties from the literature, as we will demonstrate in Chapter 4.

The name of a BSP also indicates whether the permitted corrections are constrained: if none of *BS*, *S* or *FC* occurs in the name then arbitrary corrections are permitted; if *BS* (for backwards strict) occurs in the name then only causal corrections are permitted; and if *S* (for strict) occurs in the name then no corrections are permitted at all. Forward correctable BSPs (*FC* occurs in the name) perturb traces only at positions that are immediately followed by a visible event and, moreover, they only permit causal corrections where the corrections before the visible event are constrained even further (cf. Section 3.4.5).

In order to facilitate their intuitive understanding, we have compared the various BSPs and have derived a taxonomy that clarifies which BSPs are more restrictive than others. We have also discussed the advantages and disadvantages of each BSP. This has resulted in quite concrete advice regarding which BSPs should be used under which circumstances.

All BSPs that we introduced have been carefully motivated in order to avoid any "out of the blue" definitions of BSPs.[10] However, this does not exclude the addition of further BSPs in the future if these are found to be useful. In other words, *MAKS* is an *open framework*.

MAKS will serve as the basis for our investigations throughout this thesis.

[10]Note that a *formal* justification that our BSPs are *"right"* for specifying security is impossible. BSPs are used to *define* what security means. Hence, they constitute the formal point of reference in our considerations. In general, it is impossible to verify this point of reference formally and, at best, its value can be justified with appeal to our intuitive understanding by informal explanations and examples (as we have done).

Chapter 4

A Comparison of Information Flow Properties

4.1 Introduction

It is both tedious and expensive to verify the security of a given system. Hence, one better makes sure that the specification of the security requirements is adequate (before the verification begins). This means, on the one hand, that the chosen security property, indeed, expresses the intuitive security requirements and, on the other hand, that the security property is not more restrictive than necessary to express these requirements.

In this chapter, we analyze well known information flow properties from the literature in order to gain a deeper understanding of these properties. In particular, we want to clarify for each of these properties what restrictions are imposed on the permitted flow of information (i.e. where noninterference is demanded by a property) and under which conditions these restrictions are satisfied by a given system (i.e. what noninterference means for this property). Moreover, we want to show which information flow properties are more restrictive than others and what is gained by using a particular property instead of a less restrictive one.

More specifically, we show how generalized noninterference [McC87], forward correctability [JT88], nondeducibility for outputs [GN88], noninference [O'H90], generalized noninference [McL94a], separability [McL94a], and the perfect security property [ZL97] can be represented in *MAKS*. Each of these information flow properties is expressed using the uniform representation of *MAKS*, i.e. by specifying a set of views and a security predicate. From this representation, we immediately obtain *where* noninterference is demanded by a given property (explicated by the set of views) and *what* noninterference means for this property (explicated by the security predicate). Moreover, the modular definition of the security predicate reveals if an information flow property prevents deductions about occurrences of confidential events (use of a BSP in the first dimension) and if it prevents deductions about nonoccurrences of confidential events (use of a BSP in the second dimension). Although these facts are rather obvious, given the representation of an information flow property in *MAKS*, they often are not as obvious from the original definition of this property. The modular representation of information flow properties in *MAKS* also allows us to reduce the comparison of information flow properties to the comparison of simpler BSPs. This reduction considerably simplifies the process of comparing complex information flow properties to each other. Our comparison culminates in a taxonomy of known information flow properties.

In the following, we derive some interesting insights about the various known information flow properties. For example, we show that generalized noninference [McL94a] permits noncausal corrections and, hence, it cannot be used to rule out possibilities of information leakage like the one explained in Example 3.4.22. Using noninference [O'H90], which is a more restrictive property than generalized noninference, has the advantage that this possibility of information leakage will be detected. The reason for this is that noninference does not permit noncausal corrections. Hence, from the view point of security, noninference is preferable to generalized noninference. These are the kind of insights that we aim for with our analysis of known information flow properties in this chapter.

We also propose some novel information flow properties that overcome deficiencies of previously proposed ones. For example, we derive GNI^* from generalized noninterference [McC87] (abbreviated by GNI) by replacing BSI with $BSIA^{\rho_C}$. The advantage of GNI^* is that it can also be used for systems that are not input total while GNI cannot be used for such systems.

By representing the well known information flow properties in *MAKS*, we demonstrate that our framework is quite expressive. Nevertheless, it is possible to exploit the uniform representation of information flow properties in *MAKS* to simplify reasoning about these properties. That is, the concepts of our framework provide enough structure for this purpose. More specifically, reasoning about complex information flow properties can be reduced to reasoning about simpler BSPs. In this chapter, we exploit this possibility to simplify the comparison of information flow properties. Further exploitations of this possibility will follow in subsequent chapters.[1] In summary, *MAKS* combines expressiveness with uniformity. As we will show, this is a major improvement over prior frameworks because these either emphasized expressiveness (at the cost of uniformity) or uniformity (at the cost of expressiveness).

Overview. In Section 4.2, it is shown how various information flow properties from the literature can be represented in *MAKS*. Based on this representation, the properties are compared to each other in Section 4.3. In Section 4.4, the main results of this chapter are summarized and are put into perspective with results obtained with other frameworks.

Notational Conventions. $ES = (E, I, O, Tr)$ denotes an event system and $\mathcal{V} = (V, N, C)$ denotes a view in E. L and H denote subsets of E that are referred to as "low-level events" and "high-level events", respectively. Moreover, let $\mathcal{HI}_L = (L, H \setminus I, H \cap I)$ and $\mathcal{H}_L = (L, \emptyset, H)$ denote the views of domain L in Pol_{HI} and Pol_H, respectively (cf. Examples 3.3.3 and 3.3.6). For brevity, we also write $\mathcal{HI}$ and $\mathcal{H}$ instead of $\mathcal{HI}_L$ and $\mathcal{H}_L$, respectively.

4.2 Assembling Known Information Flow Properties

In this section, we demonstrate how various information flow properties from the literature can be represented in *MAKS*. The known information flow properties that we investigate include *generalized noninterference* [McC87, McL94a, ZL97], *forward correctability* [JT88], *nondeducibility for outputs* [GN88], *noninference* [O'H90], *generalized noninference* [McL94a], *separability*[McL94a], and the *perfect security property* [ZL97]. Although all of these properties have been originally defined for a two-level security policy that distinguishes a high level

[1] In Chapter 5, we will use this approach to simplify the derivation of verification techniques for information flow properties. In Chapter 6, we will use it to derive compositionality results.

and a low level, they differ in where they require absence of information flow. For example, separability requires that there is absolutely no information flow from the high level to the low level while generalized noninterference requires only that no information flows from high-level *input* to the low level. From the representation of these properties in *MAKS* that we will present in this section, these facts are very obvious: separability enforces the view $\mathcal{H}$ while generalized noninterference enforces the view $\mathcal{HI}$. More generally, each of the known information flow properties enforces one of these two views (as we will show). The definitions of noninterference that underly the various information flow properties are even more diverse.

4.2.1 Generalized Noninterference

We start our investigation with McCullough's *generalized noninterference* [McC87]. The motivation for defining this property was to overcome deficiencies of an earlier information flow property, namely *nondeducibility* [Sut86]. Historically, nondeducibility was the first generalization of Goguen and Meseguer's noninterference [GM82] suitable for deterministic as well as nondeterministic systems. However, it was found that nondeducibility is too weak as a security property because it cannot detect various forms of information leakage. After the proposal of generalized noninterference it had become obsolete. Generalized noninterference has received much attention by different researchers and, to date, is one of the most investigated information flow properties. In particular, there are several compositionality results for this property, positive as well as negative ones (e.g. [McC87, McL94a, ZL95, ZL96, ZL98]). Moreover, there are different variants of generalized noninterference, most notably, the original definition [McC87] and an interleaving-based version [McL94a, ZL97]. As we will show, these two variants are not equivalent to each other, neither in the mathematical sense nor in their value as security properties. Let us now show how the two variants of generalized noninterference can be represented in *MAKS* and what we gain from this representation.

The Original Definition of Generalized Noninterference. An event system ES satisfies *generalized noninterference*, denoted by $GNI(ES)$, iff for every possible trace $\tau \in Tr$ and every perturbation $t \in E^*$ formed from τ by adding and deleting high-level inputs there is a possible trace $\tau' \in Tr$ such that τ' is the same as t in the constant portion and differs from t in the changed portion only in high-level non-inputs. *GNI* can be formally defined as follows.

Definition 4.2.1 (Generalized noninterference [McC87]).

$$\begin{aligned} GNI(ES) \equiv\ & \forall t_1, t_2, t_3 \in E^*. \\ & ((t_1.t_2 \in Tr \wedge t_3|_{E\setminus(H\cap I)} = t_2|_{E\setminus(H\cap I)}) \\ & \Rightarrow \exists t_4 \in E^*. (t_1.t_4 \in Tr \wedge t_4|_{L\cup(H\cap I)} = t_3|_{L\cup(H\cap I)})) \end{aligned}$$ ◇

In this definition, $t_1.t_3$ is the perturbation of the possible trace $t_1.t_2$ that is constructed by modifying occurrences of high-level inputs in t_2 (i.e. t_1 is the constant portion of $\tau = t_1.t_2$). The trace $t_1.t_4$ is a correction of this perturbation (i.e. $\tau' = t_1.t_4$ does not differ from τ in the constant portion). We refrain from trying a more intuitive explanation of generalized noninterference at this point. In our opinion, generalized noninterference can be much easier understood from its representation in *MAKS* than from its original definition. This is also true for most other information flow properties investigated in this section.

In the following lemma, we show which BSPs are implied by *GNI* and, conversely, which combination of BSPs implies *GNI*.

Lemma 4.2.2. The following implications are valid:

$$GNI(ES) \Rightarrow BSD_{\mathcal{HI}}(Tr) \quad (4.1)$$

$$GNI(ES) \Rightarrow BSI_{\mathcal{HI}}(Tr) \quad (4.2)$$

$$(BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr)) \Rightarrow GNI(ES) \quad (4.3)$$

◇

Proof. For proving $BSD_{\mathcal{HI}}(Tr)$ in (4.1), let $\alpha, \beta \in E^*$ and $hi \in H \cap I$ be arbitrary. Moreover, assume that $\beta.\langle hi\rangle.\alpha \in Tr$ and $\alpha|_{H\cap I} = \langle\rangle$ hold. Applying Definition 4.2.1 for $t_1 = \beta$, $t_2 = \langle hi\rangle.\alpha$, and $t_3 = \alpha$ yields that there is a trace $t_4 \in E^*$ with $t_1.t_4 \in Tr$ and $t_4|_{L\cup(H\cap I)} = t_3|_{L\cup(H\cap I)}$. Hence, for $\alpha' = t_4$, we have $\beta.\alpha' \in Tr$, $\alpha'|_L = t_3|_L = t_2|_L = \alpha|_L$, and $\alpha'|_{H\cap I} = t_3|_{H\cap I} = \alpha|_{H\cap I} = \langle\rangle$, i.e. $BSD_{\mathcal{HI}}(Tr)$ holds.

The proof of (4.2) is similar to the one of (4.1) where $t_1 = \beta$, $t_2 = \alpha$, and $t_3 = \langle hi\rangle.\alpha$ should be chosen when applying Definition 4.2.1.

For proving (4.3), two inductive arguments are used. Let $t_1, t_2, t_3 \in E^*$ be arbitrary such that $t_1.t_2 \in Tr$ and $t_3|_{E\setminus(H\cap I)} = t_2|_{E\setminus(H\cap I)}$ hold. By induction on the number of occurrences of high-level input events in t_2, we obtain from $BSD_{\mathcal{HI}}(Tr)$ that there is a trace $t' \in E^*$ with $t_1.t' \in Tr$, $t'|_L = t_2|_L$, and $t'|_{H\cap I} = \langle\rangle$, i.e. $t_1.t'$ results from $t_1.t_2$ by deleting all occurrences of high-level inputs in t_2 in a stepwise manner (from right to left). From $t_3|_{E\setminus(H\cap I)} = t_2|_{E\setminus(H\cap I)}$ and $t'|_L = t_2|_L$, we obtain that $t'|_L = t_3|_L$ holds. By applying $BSI_{\mathcal{HI}}(Tr)$, the occurrences of high-level inputs in t_3 can now be inserted into $t_1.t'$ in a stepwise manner (from left to right). By induction over the number of occurrences of high-level input events in t_3, we obtain that there is a trace $t_4 \in E^*$ with $t_1.t_4 \in Tr$ and $t_4|_{L\cup(H\cap I)} = t_3|_{L\cup(H\cap I)}$. □

The representation of *GNI* in *MAKS* is an immediate consequence of Lemma 4.2.2.

Theorem 4.2.3 (Assembling *GNI*). The following equivalence is valid:

$$GNI(ES) \Leftrightarrow (BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr)) \quad \diamond$$

Proof. The equivalence follows from Lemma 4.2.2. □

Note that *GNI* constructs perturbations of possible traces by modifying occurrences of high-level inputs at *arbitrary positions* in a trace while *BSD* and *BSI* perturb a trace only by deleting or inserting occurrences of high-level inputs at *positions where they are not followed by other high-level inputs*. Nevertheless, *BSD* and *BSI* suffice for the representation of *GNI*.

Interleaving-Based Generalized Noninterference. McLean proposed a variant of generalized noninterference that requires that every interleaving of a sequence of high-level inputs with every possible observation can be corrected to a possible trace by adapting occurrences of high-level internal events and high-level output events [McL94a]. Since this variant of generalized noninterference is based on the construction of interleavings, we refer to it as *interleaving-based generalized noninterference* (abbreviated by *IBGNI*). The following definition reflects the event-based formalization of *IBGNI* proposed in [ZL97].

Definition 4.2.4 (Interleaving-based generalized noninterference [ZL97]).

$$IBGNI(ES) \equiv \forall \tau_l \in Tr.\, \forall t_{hi} \in (H \cap I)^*.\, \forall t \in E^*.\\ [t \in interleaving(t_{hi}, \tau_l|_L) \Rightarrow \exists \tau' \in Tr.\, \tau'|_{L\cup(H\cap I)} = t] \quad \diamond$$

For deriving a representation of *IBGNI*, we proceed like for *GNI*: we prove a lemma that shows which BSPs are implied by *IBGNI* and, conversely, which combination of BSPs implies *IBGNI*. The representation theorem for *IBGNI* is then a simple consequence of this lemma. The same approach will be used to derive representations of all other information flow properties in *MAKS*. However, for clarity of the presentation, we only present the representation theorems from now on. The corresponding lemmas can be found in Appendix A.

The following theorem shows how to represent *IBGNI* in *MAKS*.

Theorem 4.2.5 (Assembling *IBGNI*). The following equivalence is valid:

$$IBGNI(ES) \quad \Leftrightarrow \quad D_{\mathcal{HI}}(Tr) \wedge I_{\mathcal{HI}}(Tr) \qquad \diamond$$

Proof. The equivalence follows from Lemma A.1.1 and Theorem 3.5.3.[2] □

Insights from Representation in *MAKS*. Both variants of generalized noninterference, i.e. *GNI* and *IBGNI*, require that there is *no information flow from high-level inputs to the low-level*. This is an immediate consequence of the use of the view $\mathcal{HI}$ in the representation of these properties (cf. Theorems 4.2.3 and 4.2.5). Recall that $\mathcal{HI} = (L, H \setminus I, H \cap I)$ holds, which means that the high-level inputs are the confidential events ($C = H \cap I$ holds) and the low-level events are the visible events ($V = L$ holds). Another consequence of using $\mathcal{HI}$ is that information flow from other high-level events (internal events and output events) to the low level is not explicitly forbidden by these properties. Therefore, neither *GNI* nor *IBGNI* should be used as security properties for systems that generate new secrets internally.

Another immediate consequence of the representation theorems is that both properties prevent deductions about *occurrences* as well as about *nonoccurrences* of high-level inputs. This is because BSPs from both dimensions are used in the representation of these properties.

The representation theorems also clarify what the differences are between *GNI* and *IBGNI*. While *GNI* is assembled from backwards-strict BSPs (i.e. *BSD* and *BSI*), *IBGNI* is assembled from the corresponding non-strict BSPs (i.e. *D* and *I*). This implies that *GNI* and *IBGNI* are not equivalent to each other in the mathematical sense. Moreover, they also differ in the degree of security that they ensure. In Example 3.4.22, we have shown that non-strict BSPs fail to detect certain possibilities for information leakage because they permit noncausal corrections. Consequently, *IBGNI* also fails to detect such insecurities because it is assembled from non-strict BSPs. However, backwards-strict BSPs can be used for detecting such insecurities and so can *GNI*. Hence, from the view point of security, the original definition of generalized noninterference is preferable to the later proposed interleaving-based variant.

Since *BSI* is used in the representation of *GNI*, a necessary condition for the satisfaction of *GNI* is that all high-level inputs are always enabled. In other words, $GNI(ES)$ implies $total(ES, H \cap I)$. *IBGNI* is more liberal in this respect because it does not require totality in the set of high-level inputs, however, it is not much more liberal. Both of these information flow properties require that all sequences of confidential events are possible. That is, if $GNI(ES)$ or $IBGNI(ES)$ holds then $\{\tau|_{H\cap I} \mid \tau \in Tr\} = (H \cap I)^*$ must also hold. This is because *BSI* and *I* are used, respectively, in the representations of *GNI* and *IBGNI*. If these conditions are too restrictive for a given system then one might want to use slightly

[2] Alternatively, *IBGNI* can also be represented by $R_{\mathcal{HI}}(Tr) \wedge I_{\mathcal{HI}}(Tr)$. The equivalence $IBGNI(ES) \Leftrightarrow R_{\mathcal{HI}}(Tr) \wedge I_{\mathcal{HI}}(Tr)$ also follows from Lemma A.1.1 and Theorem 3.5.3.

more liberal variants of generalized noninterference instead. We will propose such variants in Remark 4.2.6.

The representation of *GNI* and *IBGNI* in *MAKS* makes the similarities and differences of these properties very clear. All facts that we have pointed out above could be easily read off this representation. In our opinion, these facts are not as obvious from the original definitions of these properties (cf. Definitions 4.2.1 and 4.2.4), and this is why we claim that the representation in *MAKS* helps to understand information flow properties better.

Remark 4.2.6. In our above discussion, we pointed out that *GNI* and *IBGNI* both presume all sequences of high-level inputs to be possible. If this assumption is not satisfied by a given system then *GNI* and *IBGNI* reject this system as "insecure". However, in Section 3.4.3, we have shown that there are systems that are intuitively secure although they do not satisfy this assumption. This was the motivation for introducing the basic security predicates IA^{ρ}, $BSIA^{\rho}$, and SIA^{ρ}, which do not make this problematic assumption. They only require the insertion of confidential events at points of a trace where they are ρ-admissible.

By replacing *BSI* with $BSIA^{\rho_C}$ in the representation of *GNI* and by replacing *I* with IA^{ρ_C} in the representation of *IBGNI*, we obtain the two information flow properties GNI^* and $IBGNI^*$ that do no require that all sequences of confidential events are possible. This means, these novel information flow properties do not rule out that sequences of confidential events model meaningful computations.[3] Since $\rho(\mathcal{V}) = C$ holds for every view $\mathcal{V} = (V, N, C)$ and also a corresponding BSP from the first dimension is enforced in addition to the respective BSP from the second dimension (i.e. *BSD* in addition to $BSIA^{\rho_C}$ and *D* in addition to IA^{ρ_C}), a low-level observer cannot exploit his observations of the running system to deduce information beyond what he knows already from the system specification (for a more detailed discussion of these issues cf. Remark 3.4.21 and Section 3.5.2). Hence, despite GNI^* and $IBGNI^*$ being more liberal wrt. meaningful confidential computations than *GNI* and *IBGNI*, respectively, they do not differ much wrt. the degree of security that they ensure. ◇

Let us summarize the definitions of our novel information flow properties GNI^* and $IBGNI^*$.

Definition 4.2.7 (GNI^*). $GNI^*(ES) \equiv BSD_{\mathcal{HI}}(Tr) \wedge BSIA^{\rho_C}_{\mathcal{HI}}(Tr)$ ◇

Definition 4.2.8 ($IBGNI^*$). $IBGNI^*(ES) \equiv D_{\mathcal{HI}}(Tr) \wedge IA^{\rho_C}_{\mathcal{HI}}(Tr)$ ◇

The most important facts about the information flow properties *GNI*, *IBGNI*, GNI^*, and $IBGNI^*$ that can be obtained from their representation in *MAKS* are viewed in Table 4.1. How to read this table is explained in Remark 4.2.9. The same kind of table will also be used for summarizing facts about all other information flow properties in this section. This uniformity will simplify comparisons between different properties.

Remark 4.2.9 (Reading the summaries of facts). The *first three columns* in Table 4.1 indicate where noninterference is demanded by a given property. Depending on the set C in the view that is used in the representation of a property, an information flow property prevents information flow from high-level inputs to the low level (if $H \cap I \subseteq C$ holds), from high-level internals to the low level (if $H \setminus (I \cup O) \subseteq C$ holds), or from high-level outputs to the low level (if $H \cap O \subseteq C$ holds). The *fourth column* indicates whether an information

[3] McLean's interleaving-based variant of generalized noninterference [McL94a] neither requires systems to be input total nor all sequences of high-level input events to be possible. In this respect, his property is more closely resembled by our property $IBGNI^*$ than by the property *IBGNI* as introduced in [ZL97].

	set C in the view			BSPs from which dimensions and ρ-admissibility							strict BSPs
	information flow from high inputs to low is prevented	information flow from high internals to low is prevented	information flow from high outputs to low is prevented	deductions about occurrences of events in C prevented	deductions about nonocc. of events in C *prevented completely*	deductions about nonocc. of events in C *limited to static knowledge*	deductions about nonocc. of events in C *prevented to some extent*	totality in high inputs not required	not all sequences of events in C need to be possible	set of events on which enabledness of events in C may depend	information leakage because of noncausal corrections is avoided
GNI	√			√	√	√	√			$\emptyset$	√
IBGNI	√			√	√	√	√	√		$H \setminus I$	
GNI*	√			√		√	√	√	√	$H \cap I$	√
IBGNI*	√			√		√	√	√	√	H	

Table 4.1: Facts about *GNI*, *IBGNI*, *GNI**, and *IBGNI** in summary

flow property prevents deductions about occurrences of confidential events or, in other words, whether a BSP from the first dimension is used in the representation of the property. The next three columns indicate whether deductions about nonoccurrences of confidential events are prevented and to what extent. Deductions about nonoccurrences of confidential events are prevented completely (*fifth column*) if one of *I*, *BSI*, or *SI* occurs in the representation of an information flow property. If IA^ρ, $BSIA^\rho$, or SIA^ρ is used in the representation instead then such deductions are only prevented to some extent (*7th column*). Depending on the choice of the parameter ρ, the possible deductions are limited to information that the observer can already deduce from the system specification without observing the dynamic behavior of the system (i.e. his static knowledge). To prevent information beyond this static knowledge from being deduced is desirable (*sixth column*). Whether this is the case or not can be easily read off the representation of a property in *MAKS* (Does $\rho(V, N, C) \subseteq C$ hold and is a corresponding BSP from the first dimension enforced in addition?). Note that the entries in the fifth columns for *GNI* and *IBGNI* differ from the ones for *GNI** and *IBGNI**. What is gained by giving up that deductions about nonoccurrences of confidential events are prevented completely is reflected in columns 8–10. *Column 8* indicates whether a given property is also applicable for systems that are not input total (true for *IBGNI*, *GNI**, *IBGNI** but not for *GNI*). *Column 9* indicates whether sequences of confidential events may correspond to meaningful computations (not demanding chaotic behavior in C). Note that for the information flow properties in Table 4.1, the entries in this column are inverse to the ones in the fifth column. *Column 10* explicates more precisely on which events the enabledness of events in C may

depend. For example, *GNI* requires totality in $H \cap I$ and, hence, the enabledness of events in C may not depend on any other events (entry $\emptyset$). In contrast to this, *GNI** permits that the enabledness of confidential events may depend on previous occurrences of high-level inputs (entry $H \cap I$) because $BSIA^{\rho_C}$ is used in its representation and $\rho(\mathcal{HI}) = H \cap I$ holds. Information flow properties that are assembled from non-strict BSPs permit that occurrences of events in C also depend on previous occurrences of events in N. This is why the sets for *IBGNI* and *IBGNI** in the 10th column contain high-level internal events and high-level output events (entries $H \setminus I$ and H). In general, the larger the set on which the enabledness of events in C may depend, the more noncritical dependencies are permitted. However, if non-strict BSPs are used then certain possibilities for information leakage may remain undetected (as pointed out in Example 3.4.22). Whether an information flow property prevents such insecurities is indicated by *column 11*. For example, *GNI* prevents such information leakage because it is assembled from backwards strict BSPs while *IBGNI* does not because it is assembled from non-strict BSPs.

In general, check marks in the table indicate a desirable fact about a property. ◇

4.2.2 Forward Correctability

The next information flow property that we investigate is Johnson and Thayer's *forward correctability* [JT88]. The motivation for defining this property was to overcome a deficiency of an earlier information flow property, namely *restrictiveness* [McC87]. Historically, restrictiveness was the first information flow property that is preserved under composition in general (generalized noninterference is not compositional in general). However, restrictiveness imposes very strong requirements on a system. That these requirements can be liberalized while retaining compositionality as well as the desired restrictions on the information flow, was the key observation that led to the definition of forward correctability. After the proposal of forward correctability, restrictiveness had become obsolete. Besides forward correctability, Johnson and Thayer proposed a family of information flow properties, namely 2-forward correctability, 3-forward correctability, ... All of these properties are also improvements over restrictiveness but none of them has advantages over forward correctability (also named 1-forward correctability). If a security property is needed that prevents information leakage from high-level input to the low level and that is also composable then forward correctability still appears to be the best choice available to date.[4]

Definition 4.2.10 (Forward correctability [JT88]). [5]

$$
\begin{aligned}
FC(ES) \quad \equiv \quad & \forall t_1, t_2 \in E^*.\, \forall hi \in H \cap I.\, \forall li \in L \cap I. \\
& \left(\begin{array}{l}
((t_1.[\langle li \rangle].t_2 \in Tr \wedge t_2|_{H \cap I} = \langle\rangle) \\
\quad \Rightarrow \exists t_3 \in E^*.\, (t_1.\langle hi \rangle.[\langle li \rangle].t_3 \in Tr \wedge t_3|_L = t_2|_L \wedge t_3|_{H \cap I} = \langle\rangle)) \\
\wedge ((t_1.\langle hi \rangle.[\langle li \rangle].t_2 \in Tr \wedge t_2|_{H \cap I} = \langle\rangle) \\
\quad \Rightarrow \exists t_3 \in E^*.\, (t_1.[\langle li \rangle].t_3 \in Tr \wedge t_3|_L = t_2|_L \wedge t_3|_{H \cap I} = \langle\rangle))
\end{array} \right)
\end{aligned}
$$

◇

Forward correctability perturbs a possible trace $t_1.\langle li \rangle.t_2$ by *inserting* an occurrence of an arbitrary high-level input event hi before the occurrence of the low-level input event li, which

[4] In Chapter 6, we will propose *weakened forward correctability*, a novel information flow property that constitutes an improvement over forward correctability.

[5] In the definition of forward correctability, $[\langle li \rangle]$ is a place holder that can be replaced by the empty trace $\langle\rangle$ or by $\langle li \rangle$. Implicitly, Johnson and Thayer assume that this replacement is done consistently for every trace when the definition is applied.

results in the trace $t_1.\langle hi\rangle.\langle li\rangle.t_2$. The requirement is that this trace can be corrected by adapting occurrences of high-level internal events and high-level output events in t_2 to a possible trace $t_1.\langle hi\rangle.\langle li\rangle.t_3$ that yields the same observation. Note that forward correctability *does not permit arbitrary causal corrections* of this trace but rather only permits corrections after the occurrence of *li*. However, this restriction on the correction only applies if the high-level input event is inserted at a position that is immediately followed by an occurrence of a low-level input event. Hence, if a possible trace has the form $t_1.t_2$ where t_2 does not start with an occurrence of a low-level input event then arbitrary causal corrections of the perturbation $t_1.\langle hi\rangle.t_2$ are permitted. The second implication in Definition 4.2.10 requires that possible traces can also be perturbed by *deleting* an occurrence of a high-level input event where the constraints on the correction are like for the insertion of high-level inputs events. We refrain from trying an intuitive explanation of forward correctability at this point because it is much easier to understand this property given its representation in *MAKS*.

Let us now show how forward correctability can be represented in *MAKS*.

Theorem 4.2.11 (Assembling *FC*). For $\Gamma_{FC}=(I,\emptyset,I)$, we have the following equivalence:

$$FC(ES) \quad \Leftrightarrow \quad BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr) \wedge FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \wedge FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \qquad \diamond$$

Proof. The equivalence follows immediately from Lemma A.2.1. □

Insights from Representation in *MAKS*. A comparison of the representation of forward correctability (Theorem 4.2.11) with the representation of generalized noninterference (Theorem 4.2.3) reveals that these two information flow properties are very similar to each other. Both of them enforce the view $\mathcal{HI}$ and are assembled from *BSD* and *BSI*. More generally, all facts about *GNI* that we have pointed out in Section 4.2.1 are also true for forward correctability. The only difference between these two properties is that $FCD^{\Gamma_{FC}}$ and $FCI^{\Gamma_{FC}}$ are used in the representation of forward correctability but not in the representation of generalized noninterference. This means that $FCD^{\Gamma_{FC}}$ and $FCI^{\Gamma_{FC}}$ correspond to basic ingredients of forward correctability that generalized noninterference does not have. This difference is responsible for the fact that forward correctability is preserved under composition in general, while generalized noninterference is not. This is a first example for the value of forward correctable BSPs. Further will follow in Chapter 6 where we investigate the compositionality of information flow properties in detail.

Remark 4.2.12. Like *GNI*, *FC* also requires totality in the set of high-level input events, i.e. $FC(ES)$ implies $total(ES, H \cap I)$. By replacing *BSI* and $FCI^{\Gamma_{FC}}$ with $BSIA^{\rho_C}$ and $FCIA^{\rho_C,\Gamma_{FC}}$ in the representation of forward correctability, we arrive at a novel information flow property (namely FC^*) that does not make this totality assumption. ◇

Definition 4.2.13 (*FC).** Let $\Gamma_{FC} = (I,\emptyset,I)$.

$$FC^*(ES) \quad \equiv \quad BSD_{\mathcal{HI}}(Tr) \wedge BSIA^{\rho_C}_{\mathcal{HI}}(Tr) \wedge FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \wedge FCIA^{\rho_C,\Gamma_{FC}}_{\mathcal{HI}}(Tr) \qquad \diamond$$

The most important facts about the information flow properties *FC* and FC^* that can be obtained from their representation in *MAKS* are viewed in Table 4.2 (cf. Remark 4.2.9 for how to read this kind of table).

4.2.3 Nondeducibility for Outputs

Generalized noninterference and forward correctability are not adequate security properties for systems that generate secrets internally because they only prevent the inference of information

	set C in the view			BSPs from which dimensions and ρ-admissibility							strict BSPs
	information flow from high inputs to low is prevented	information flow from high internals to low is prevented	information flow from high outputs to low is prevented	deductions about occurrences of events in C prevented	deductions about nonocc. of events in C *prevented completely*	deductions about nonocc. of events in C *limited to static knowledge*	deductions about nonocc. of events in C *prevented to some extent*	totality in high inputs not required	not all sequences of events in C need to be possible	set of events on which enabledness of events in C may depend	information leakage because of noncausal corrections is avoided
FC	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$			$\emptyset$	$\surd$
FC^*	$\surd$			$\surd$		$\surd$	$\surd$	$\surd$	$\surd$	$H \cap I$	$\surd$

Table 4.2: Facts about FC and FC^* in summary

about occurrences and nonoccurrences of *high-level inputs*. This limitation motivated the definition of *nondeducibility for outputs* by Guttman and Nadel [GN88]. In comparison to generalized noninterference or forward correctability, nondeducibility for outputs is a more appropriate security property for systems that *generate secrets* because this information flow property prevents deductions of information about occurrences and nonoccurrences of *all* high-level events, including high-level input, internal, and output events.

Nondeducibility for outputs permits noncritical information flow from the low level to the high level to some extent. More specifically, the high-level behavior may depend on input that is provided by low-level users (but not on any other low-level behavior). The possible user inputs are modeled by a set $UI \subseteq I$ of input events. Moreover, nondeducibility for outputs is a composable information flow property.[6]

An event system ES satisfies *nondeducibility for outputs*, denoted by $NDO(ES)$, iff every trace $t \in E^*$ is a possible trace of ES under the assumption that t equals some possible trace in its occurrences of low-level events (i.e. $\exists \tau_l \in Tr.\, \tau_l|_L = t|_L$) and t equals some possible trace in its occurrences of events in $H \cup (L \cap UI)$ (i.e. $\exists \tau_{hlui} \in Tr.\, \tau_{hlui}|_{H \cup (L \cap UI)} = t|_{H \cup (L \cap UI)}$). In other words, t only needs to be a possible trace if the sequence of low-level events and the sequence of $H \cup (L \cap UI)$-events both make sense.

Definition 4.2.14 (Nondeducibility for outputs [GN88]). Let ES be *input total*. For a set $UI \subseteq I$ of user inputs, *nondeducibility for outputs* is defined as follows:

$$\begin{aligned} NDO(ES) \quad \equiv \quad & \forall \tau_l, \tau_{hlui} \in Tr.\, \forall t \in E^*. \\ & [(t|_L = \tau_l|_L \wedge t|_{H \cup (L \cap UI)} = \tau_{hlui}|_{H \cup (L \cap UI)}) \Rightarrow t \in Tr] \end{aligned} \qquad \diamond$$

[6] To be precise, compositionality of nondeducibility for outputs is only ensured if events in UI are not used for communicating between system components. This means, the set of user inputs of one component must be disjoint from the sets of events of all other components.

Nondeducibility for outputs has been introduced under the assumption of input totality, i.e. that $total(ES, I)$ holds. The following example illustrates that NDO is not satisfactory from the viewpoint of security if the assumption of input totality were dropped.

Example 4.2.15. Let $ES_{NDO} = (E_{NDO}, I_{NDO}, O_{NDO}, Tr_{NDO})$ be an event system such that $E_{NDO} = \{hi, li\}$, $I_{NDO} = \{hi, li\}$, $O_{NDO} = \emptyset$, $Tr_{NDO} = \{\langle hi.li\rangle, \langle hi\rangle, \langle\rangle\}$, and $UI_{NDO} = \{li\}$. Note that ES_{NDO} is *not* input total. However, for this example, let us presume that the definition of NDO would nevertheless be applicable. Under this assumption, $NDO(ES_{NDO})$ holds ($t|_{H\cup(L\cap UI)} = \tau_{hlui}|_{H\cup(L\cap UI)}$ implies $t = \tau_{hlui}$ because $L = L \cap UI_{NDO}$). Nevertheless, one can deduce from the observation $\langle li\rangle$ that $\langle hi.li\rangle$ has occurred, i.e. there is information leakage from the high level to the low level although NDO holds. This shows that simply dropping the assumption of input totality in the definition of NDO is not a good idea because one arrives at a security property that is too weak to detect some insecurities. ◇

Let us now show how nondeducibility for outputs can be represented in *MAKS*.

Theorem 4.2.16 (Assembling *NDO*). For a set $UI \subseteq I$ of user inputs and the function ρ_{UI} defined by $\rho_{UI}(V, N, C) = C \cup (V \cap UI)$ the following equivalence is valid if $total(ES, I)$ holds:

$$NDO(ES) \quad \Leftrightarrow \quad BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho_{UI}}_{\mathcal{H}}(Tr) \qquad \diamond$$

Proof. The equivalence follows from Lemma A.3.1 and Theorems 3.5.3 and 3.5.12.[7] □

Insights from Representation in *MAKS*. That nondeducibility for outputs prevents deductions about *occurrences* as well as *nonoccurrences* of confidential events can be easily read off its representation in *MAKS* (BSPs from both dimensions are used). Moreover, the use of the view $\mathcal{H}$ (rather than of the view $\mathcal{HI}$) in this representation clarifies that NDO indeed prevents deductions about occurrences and nonoccurrences of *all* high-level events rather than only of high-level inputs (recall that $C = H$ holds for the view $\mathcal{H}$). In this respect, NDO differs from all information flow properties that we have considered so far in this chapter. Hence, NDO is a more appropriate security property for systems that generate secrets internally than any of *GNI*, *IBGNI*, *GNI**, *IBGNI**, *FC*, and *FC**.

$BSIA^{\rho_{UI}}$ is used in the representation of NDO and $\rho_{UI}(V, N, C) \subseteq C$ does not hold in general. From this, we learn that NDO does not rule out possibilities for deductions about nonoccurrences of confidential events completely. More specifically, despite $NDO(ES)$ holding an observer might be able to deduce from his observations of the running system up to some point of time that some confidential event cannot occur in the future (because it will not be enabled). The reasons for these kind of deductions have been discussed in detail already in Chapter 3 (cf. Remark 3.4.21 and Section 3.5.2). Whether the possibility of such deductions poses a problem for the security of a given system depends on the particular system and cannot be answered in general. This means, when applying NDO one should justify carefully that NDO is indeed an appropriate security property for the system under consideration, in particular that the permitted deductions about the impossibility of high-level events to occur in the future do not constitute a security breach.

[7] There are various alternative representations of NDO in *MAKS*. This is mostly due to the fact that the view $\mathcal{H}$ does not permit any corrections and, hence, the non-strict, backwards-strict, and strict version of a BSP are equivalent to each other (cf. Theorems 3.5.6 and 3.5.15). More specifically, for $\mathrm{BSP}' \in \{R, SR, D, BSD, SD\}$ and $\mathrm{BSP}'' \in \{IA^{\rho_{UI}}, BSIA^{\rho_{UI}}, SIA^{\rho_{UI}}\}$, the equivalence $NDO(ES) \Leftrightarrow \mathrm{BSP}'_{\mathcal{H}}(Tr) \wedge \mathrm{BSP}''_{\mathcal{H}}(Tr)$ holds. All instances of this equivalence follow from Lemma A.3.1 and Theorems 3.5.3 and 3.5.12.

	set C in the view			BSPs from which dimensions and ρ-admissibility							strict BSPs
	information flow from high inputs to low is prevented	information flow from high internals to low is prevented	information flow from high outputs to low is prevented	deductions about occurrences of events in C prevented	deductions about nonocc. of events in C *prevented completely*	deductions about nonocc. of events in C *limited to static knowledge*	deductions about nonocc. of events in C *prevented to some extent*	totality in high inputs not required	not all sequences of events in C need to be possible	set of events on which enabledness of events in C may depend	information leakage because of noncausal corrections is avoided
NDO	√	√	√	√			√		√	$H \cup (L \cap UI)$	√
NDO^*	√	√	√	√			√	√	√	$H \cup (L \cap UI)$	√

Table 4.3: Facts about NDO and NDO^* in summary

Remark 4.2.17 (NDO^*). Dropping the assumption of input totality in the definition of NDO results in an unsatisfactory security property because insecurities like, e.g., the one in Example 4.2.15 remain undetected. However, the representation of NDO in *MAKS* (cf. Theorem 4.2.16) would detect the insecurity in Example 4.2.15. $BSD_{\mathcal{H}}(Tr_{NDO})$ does not hold for the example system. This means, the representation of NDO in *MAKS* is equivalent to the original definition for systems that are input total and, for systems that are not input total, it even is preferable to the original definition. By dropping the input totality assumption we obtain a novel information flow property NDO^*. ◇

Definition 4.2.18 (NDO^*). $NDO^*(ES) \equiv BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_{UI}}(Tr)$ ◇

The most important facts about NDO and NDO^* that follow from their representation in *MAKS* are summarized in Table 4.3 (cf. Remark 4.2.9 for how to read this kind of table).

4.2.4 Noninference

The information flow property that, to date, is known under the name *noninference* has evolved from three different information flow properties, namely Jacob's *ignorance of progress* [Jac89], O'Halloran's *noninference* [O'H90], and O'Halloran's *weak ignorance of progress* [O'H90]. All three properties prevent that some user can leak information through his "window" of the system (i.e. the part of the interface that he can access) to the "window" of another user. However, none of them provides any protection against the collaboration of users. That is, collaborating users might be able to leak confidential information across a given system although this system satisfies ignorance of progress, noninference, and weak ignorance of progress. The differences between the three properties are not very substantial and, hence, have disappeared over time. Moreover, in order to rule out information leakage by

collaborating users, the window-based approach to formulating restrictions on the permitted flow of information has been replaced by multi-level security policies. This has resulted in the property that, to date, is known under the name *noninference* [McL94a, ZL97].

An event system ES satisfies *noninference*, denoted by $NF(ES)$, iff every trace $\tau' \in E^*$ that results by removing all occurrences of high-level events from a possible trace τ (i.e. $\tau' = \tau|_L$) is also a possible trace. *NF* can be formally defined as follows.

Definition 4.2.19 (Noninference [ZL97]).

$$NF(ES) \quad \equiv \quad \forall \tau \in Tr.\, \tau|_L \in Tr \qquad \diamond$$

Interestingly, it has been found that two other information flow properties that were developed independently from noninference are equivalent. Focardi and Gorrieri showed that Wittbold and Johnson's *nondeducibility on strategies* [WJ90] is equivalent to noninference in a trace-based setting [FG95].[8] Schneider showed that *may-noninterference*, another information flow property, is also equivalent to noninference [Sch01].

Theorem 4.2.20 (Assembling *NF*). The following equivalence is valid:

$$NF(ES) \quad \Leftrightarrow \quad R_{\mathcal{H}}(Tr) \qquad \diamond$$

Proof. The equivalence follows from Lemma A.4.1 and Theorem 3.5.3.[9] □

Insights from Representation in *MAKS*. The most prominent difference of noninference to all other information flow properties that we have investigated so far is that its representation involves only a single BSP. Hence, noninference is a very primitive information flow property that prevents deductions about occurrences of confidential events but not about nonoccurrences of confidential events (R is a BSP from the first dimension). In [McL94a], it has been already noted that noninference does not prevent low-level observations from being influenced by the insertion of *high-level inputs*. Our statement is a generalization of this observation: noninference does not prevent low-level observations from being influenced by the insertion of *arbitrary high-level events*. The representation of noninference in *MAKS* provides a suitable basis for deriving a modification of noninference that prevents such dependencies in a goal-directed way. Namely, by adding some BSP from the second dimension (e.g. $BSIA^{\rho_C}$), noninference can be strengthened such that the insertion of high-level events cannot influence low-level observations in any essential way.

Further facts about noninference that can be easily obtained from its representation in *MAKS* are viewed in Table 4.4 (cf. Remark 4.2.9 for how to read this kind of table).

4.2.5 Generalized Noninference

McLean derived *generalized noninference* from noninference by relaxing the restrictions on the permitted flow of information [McL94a]. The motivation for defining generalized noninference was the observation that noninference does not permit the enabledness of high-level outputs to

[8] Focardi and Gorrieri preferred the names *nondeducibility on compositions* (*NDC*) and *strong nondeterministic noninterference* (*SNNI*) instead of, respectively, nondeducibility on strategies and noninference. Nevertheless, there are no essential differences between these properties.

[9] Since the view $\mathcal{H}$ does not permit any corrections the non-strict and strict version of a BSP are equivalent to each other (cf. Theorem 3.5.6). Therefore, *NF* can also be represented by $SR_{\mathcal{H}}(Tr)$.

depend closely on occurrences of low-level events. For example, a system where the occurrence of some low-level input enforces that this event is recorded by some high-level output before the next low-level input can occur would be rejected by noninference as "insecure", even if the system only constructs a high-level log of the low-level input that always is up-to-date and, hence, intuitively is secure [Man00c]. Generalized noninference is much more liberal wrt. dependencies between high-level outputs and low-level events and, in particular, would not reject the example system.

Rather than preventing information flow from all high-level events to the low level, generalized noninference only prevents information flow from high-level inputs to the low level.

Definition 4.2.21 (Generalized noninference [ZL97]).

$$GNF(ES) \quad \equiv \quad \forall \tau \in Tr.\, \exists \tau' \in Tr.\, [\tau'|_{H \cap I} = \langle\rangle \wedge \tau'|_L = \tau|_L] \qquad \diamond$$

Generalized noninference perturbs a possible trace τ by removing all occurrences of high-level input events (resulting in the trace $\tau|_{E \setminus (H \cap I)}$) and requires that this perturbation can be corrected to a possible trace τ' by adapting occurrences of high-level internal events and occurrences of high-level output events.

Representing *GNF* in *MAKS* is straightforward.

Theorem 4.2.22 (Assembling *GNF*). The following equivalence is valid:

$$GNF(ES) \quad \Leftrightarrow \quad R_{\mathcal{HI}}(Tr) \qquad \diamond$$

Proof. The equivalence follows from Lemma A.5.1. □

Insights from Representation in *MAKS*. The representation of generalized noninference in *MAKS* reflects the similarity to noninference. The only difference to the representation of noninference (cf. Theorem 4.2.20) is that the view $\mathcal{HI}$ is used (instead of $\mathcal{H}$). This also confirms that generalized noninference is a very weak security property. It does not prevent deductions about nonoccurrences of high-level events and it does not prevent any information flow from high-level internal or output events to the low level.

Further facts about *GNF* that can be read of its representation in *MAKS* are summarized in Table 4.4 (cf. Remark 4.2.9 for how to read this kind of table).

4.2.6 Separability

Separability is another information flow property that was proposed by McLean [McL94a]. The motivation for defining this property was to combine the advantages of noninference and generalized noninference (i.e. having a simple definition) with the advantages of restrictiveness and forward correctability (i.e. being composable and being more restrictive than generalized noninference and noninference).

Separability is a very restrictive information flow property because it rules out information flow between the high level and the low level in both directions. This means, if separability holds for a given system then there is neither information flow from the high level to the low level nor information flow from the low level to the high level. Logically, this is equivalent to a physical separation of the high level part of the system from the low level part. In

	set C in the view			BSPs from which dimensions and ρ-admissibility							strict BSPs
	information flow from high inputs to low is prevented	information flow from high internals to low is prevented	information flow from high outputs to low is prevented	deductions about occurrences of events in C prevented	deductions about nonocc. of events in C *prevented completely*	deductions about nonocc. of events in C *limited to static knowledge*	deductions about nonocc. of events in C *prevented to some extent*	totality in high inputs not required	not all sequences of events in C need to be possible	set of events on which enabledness of events in C may depend	information leakage because of noncausal corrections is avoided
NF	$\surd$	$\surd$	$\surd$	$\surd$				$\surd$	$\surd$	$H \cup L$	$\surd$
GNF	$\surd$			$\surd$				$\surd$	$\surd$	$H \cup L$	

Table 4.4: Facts about *NF* and *GNF* in summary

other words, separability can be used to logically simulate physical *air-gaps* between system components, the same idea as in Rushby's separation kernel [Rus81a].[10]

An event system *ES* satisfies *separability*, denoted by $SEP(ES)$, iff every interleaving of a possible sequence of high-level events with a possible low-level observation is a possible trace.

Definition 4.2.23 (Separability [ZL97]).

$$SEP(ES) \equiv \forall \tau_l, \tau_h \in Tr.\ interleaving(\tau_h|_H, \tau_l|_L) \subseteq \{\tau \in Tr \mid \tau|_L = \tau_l|_L\} \qquad \diamond$$

Let us now show how separability can be represented in *MAKS*.

Theorem 4.2.24 (Assembling *SEP*). For the function ρ_C that is defined by $\rho_C(V, N, C) = C$ the following equivalence is valid:

$$SEP(ES) \quad \Leftrightarrow \quad BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_C}(Tr) \qquad \diamond$$

Proof. The equivalence follows from Lemma A.6.1 and Theorems 3.5.3 and 3.5.12.[11] □

[10] Intuitively, the underlying idea was to build secure systems from physically separated components that each satisfy a component-specific security policy. In such a system, the communication channels between components would be easily identifiable because they correspond to physical connections, i.e. wires. Having only a limited number of communication channels could then be exploited in order to simplify the proof that the overall system satisfies its security policy under the assumption that each component satisfies its local security policy. However, rather than implementing the chosen distributed architecture physically (in hardware) this architecture would be simulated by the separation kernel (in software). Following this approach, the role of information flow properties would be to specify when a particular physical distribution is simulated correctly, thereby providing a basis for a formal verification of the simulation. More recently, these ideas have been related to the problem of simulating the partitioning of different system modules in embedded fault-tolerant systems in the context of Integrated Modular Avionics (IMA) [Rus99].

[11] There are various alternative representations of *SEP* in *MAKS*. This is mostly due to the fact that the view $\mathcal{H}$ does not permit any corrections and, hence, the non-strict, backwards-strict, and strict version of a BSP are equivalent to each other (cf. Theorems 3.5.6 and 3.5.15). More specifically, for $BSP' \in \{R, SR, D, BSD, SD\}$ and $BSP'' \in \{IA^{\rho_C}, BSIA^{\rho_C}, SIA^{\rho_C}\}$, the equivalence $SEP(ES) \Leftrightarrow BSP'_{\mathcal{H}}(Tr) \wedge BSP''_{\mathcal{H}}(Tr)$ holds. All instances of this equivalence follow from Lemma A.6.1 and Theorems 3.5.3 and 3.5.12.

Insights from Representation in *MAKS*. Most notably, the representation of separability in *MAKS* reveals that this property has close similarities with nondeducibility for outputs. More specifically, both properties establish the view $\mathcal{H}$ and they are assembled from *BSD* and an instance of *BSIA*. The only difference between these properties is how *BSIA* is instantiated (parameter ρ_{UI} in *NDO* and ρ_C in *SEP*). The benefit of using ρ_C instead of ρ_{UI} is that deductions about nonoccurrences are limited to what an observer could already deduce from the system specification without observing the running system (his static knowledge). This means, a system that satisfies separability is more "secure" than a system that satisfies nondeducibility for outputs but does not satisfy separability. The price paid for being more conservative in this respect is that noncritical information flow from the low level to the high level is also ruled out (recall that *NDO* permitted at least some information flow from low-level user inputs to the high level). Depending on what is more important for a given application, being restrictive wrt. deductions about nonoccurrences of confidential events or being liberal wrt. noncritical information flow, separability is preferable over nondeducibility for outputs or vice versa.

Also note that separability is assembled from the same BSPs like *GNI** (cf. Definition 4.2.7). This means, these two properties are based on the same definition of noninterference. The only difference between these properties is where information flow is restricted (views $\mathcal{H}$ and $\mathcal{HI}$, respectively).

Further facts about *SEP* that follow from its representation in *MAKS* are summarized in Table 4.5 (cf. Remark 4.2.9 for how to read this kind of table).

4.2.7 Perfect Security Property

Zakinthinos and Lee derived the *perfect security property* [ZL97] by weakening separability. The motivation for the definition of the perfect security property was the observation that separability prevents not only critical information flow from the high-level to the low-level but also noncritical information flow from the low-level to the high-level. The perfect security property was defined with the objective to have a security property that rules out critical information flow like separability but that does not restrict noncritical information flow.

An event system *ES* satisfies the *perfect security property*, denoted by *PSP*(*ES*), iff for every possible trace $\tau \in Tr$ the projection of τ to low-level events is a possible trace and if a trace $t \in E^*$ is formed from a possible trace by adding a single high-level event h at a position where h is enabled and is not followed by other high-level events then t is a possible trace. This is summarized in the following definition.

Definition 4.2.25 (Perfect security property [ZL97]).

$$\begin{aligned} PSP(ES) \equiv\ & (\forall \tau \in Tr.\, \tau|_L \in Tr) \\ & \wedge \forall \alpha, \beta \in E^*.\, [(\beta.\alpha \in Tr \wedge \alpha|_H = \langle\rangle) \\ & \qquad \Rightarrow \forall h \in H.\, (\beta.\langle h\rangle \in Tr \Rightarrow \beta.\langle h\rangle.\alpha \in Tr)] \end{aligned} \qquad \diamond$$

In order to better understand this property, let us now represent it in *MAKS*.

Theorem 4.2.26 (Assembling *PSP*). For the function ρ_E that is defined by $\rho_E(V, N, C) = V \cup N \cup C = E$ the following equivalence is valid:

$$PSP(ES) \quad \Leftrightarrow \quad BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_E}(Tr) \qquad \diamond$$

Proof. The equivalence follows from Lemma A.7.1 and Theorems 3.5.3 and 3.5.12.[12] □

Insights from Representation in *MAKS*. Interestingly, the close similarity between the perfect security property and separability is even more obvious from the representation of these properties in *MAKS* ($BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_E}(Tr)$ and $BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_C}(Tr)$) than from their definitions (cf. Definitions 4.2.23 and 4.2.25). Moreover, from the representation of the perfect security property in *MAKS*, it is obvious that this property is also quite similar to nondeducibility for outputs (represented by $BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_{UI}}(Tr)$). All three properties (*PSP*, *SEP*, and *NDO*) enforce the view $\mathcal{H}$, and they are assembled from *BSD* and an instance of *BSIA*. The parameter ρ_E used in the representation of *PSP* indicates that *PSP* is the most liberal of these properties wrt. noncritical information flow from the low level to the high level[13] (a desirable property). However, *PSP* is also more liberal than *SEP* or *NDO* wrt. the possibility of deductions of information about future nonoccurrences of high-level events (an undesirable property). Which of the three properties is best depends on the particular system under consideration and cannot be answered in general (because of the trade-off between permitting noncritical information flow and ruling out that deductions about future nonoccurrences are possible). This means, the name perfect security property is slightly misleading because *PSP* is not in general "*the perfect*" security property.

Further facts about *PSP* that follow from its representation in *MAKS* are summarized in Table 4.5 (cf. Remark 4.2.9 for how to read this kind of table).

4.3 Information Flow Properties in Comparison

In the previous section, we have demonstrated how generalized noninterference, forward correctability, nondeducibility for outputs, noninference, generalized noninference, separability, and the perfect security property can be represented in *MAKS*. The representation in *MAKS* clarified several facts about these well known information flow properties, many of them being not as obvious from the original definitions of these properties (cf. the summary of Tables 4.1–4.5 in Table 4.6). Moreover, for some of these properties, we have proposed modifications that overcome limitations of the original properties. This has led to the novel information flow properties *GNI**, *IBGNI**, *FC**, and *NDO**.

In this section, we investigate the relations between these information flow properties more closely. In order to structure the comparison, we start with properties that enforce the view $\mathcal{H}$, then turn our attention to properties that enforce the view $\mathcal{HI}$, and, finally, relate properties that enforce different views.

4.3.1 Information Flow Properties with the View $\mathcal{H}$

The view $\mathcal{H}$ is used in the representation of nondeducibility for outputs (cf. Theorem 4.2.16), of noninference (cf. Theorem 4.2.20), of separability (cf. Theorem 4.2.24), and of the perfect security property (cf. Theorem 4.2.26). This means that each of these properties prevents

[12] There are various alternative representations of *PSP* in *MAKS*. This is mostly due to the fact that the view $\mathcal{H}$ does not permit any corrections and, hence, the non-strict, backwards-strict, and strict version of a BSP are equivalent to each other (cf. Theorems 3.5.6 and 3.5.15). More specifically, for $\text{BSP}' \in \{R, SR, D, BSD, SD\}$ and $\text{BSP}'' \in \{IA^{\rho_E}, BSIA^{\rho_E}, SIA^{\rho_E}\}$, the equivalence $PSP(ES) \Leftrightarrow \text{BSP}'_{\mathcal{H}}(Tr) \wedge \text{BSP}''_{\mathcal{H}}(Tr)$ holds. All instances of this equivalence follow from Lemma A.7.1 and Theorems 3.5.3 and 3.5.12.

[13] $\rho_E(\mathcal{V}) \supseteq \rho_{UI}(\mathcal{V}) \supseteq \rho_C(\mathcal{V})$ holds.

	set C in the view			BSPs from which dimensions and ρ-admissibility							strict BSPs
	information flow from high inputs to low is prevented	information flow from high internals to low is prevented	information flow from high outputs to low is prevented	deductions about occurrences of events in C prevented	deductions about nonocc. of events in C *prevented completely*	deductions about nonocc. of events in C *limited to static knowledge*	deductions about nonocc. of events in C *prevented to some extent*	totality in high inputs not required	not all sequences of events in C need to be possible	set of events on which enabledness of events in C may depend	information leakage because of noncausal corrections is avoided
SEP	√	√	√	√		√	√	√	√	H	√
PSP	√	√	√	√			√	√	√	$H \cup L$	√

Table 4.5: Facts about *SEP* and *PSP* in summary

deductions about occurrences or nonoccurrences of *all high-level events* with the same rigor, no matter whether they are input events, internal events, or output events. The difference between the four properties lies in the BSPs from which they are assembled or, in other words, *what* deductions they rule out. Noninference is assembled from a single BSP only, namely R. Since R is a BSP from the first dimension, noninference only prevents deductions about *occurrences* of high-level events. The representations of nondeducibility for outputs, of separability, and of the perfect security property comply with the pattern $BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho}_{\mathcal{H}}(Tr)$. Hence, these properties prevent deductions about *occurrences* of high-level events (due to *BSD*) and, unlike noninference, also about *nonoccurrences* of high-level events (due to $BSIA^{\rho}$). The parameter used for instantiating *BSIA* differs: ρ_{UI} is used for nondeducibility for outputs, ρ_C for separability, and ρ_E for the perfect security property. This means, these information flow properties differ only in the extent to which deductions about nonoccurrences of confidential events are prevented (separability is the most rigorous among them in this respect) and in the extent to which noncritical information flow is permitted (the perfect security property is the most liberal among them in this respect). In Table 4.6, this is reflected by the different entries for these properties in columns 6 and 10.

Interestingly, for nondeducibility for outputs, the representation in *MAKS* even is preferable to the original definition because it also makes sense for systems that are not input total while the original definition can only be applied to systems that are input total. This observation was the motivation for using the representation of nondeducibility for outputs as the definition of a novel information flow property, namely NDO^* (also cf. column 8 of Table 4.6). Since NDO^* is preferable to *NDO*, we will only consider NDO^* from now on and, in particular, will use the term "nondeducibility for outputs" to refer to NDO^*.

Given the representation in *MAKS* it becomes a straightforward task to order these information flow properties. On the left hand side of Figure 4.1 it is shown how separability,

nondeducibility for outputs, the perfect security property, and noninference can be ordered by implication. This ordering can be easily justified given the representation of these properties in *MAKS* (cf. the right hand side of Figure 4.1) and the ordering of BSPs in each dimension (cf. Theorems 3.5.3 and 3.5.12). The ordering reflects that separability requires more perturbations than nondeducibility for outputs (because it requires the insertion of high-level events at positions where they are ρ_C-admissible rather than only at positions where they are ρ_{UI}-admissible). Moreover, the ordering reflects that nondeducibility for outputs requires more perturbations than the perfect security property and that the perfect security property requires more perturbations than noninference. Recall that none of these four properties permits any corrections.

Figure 4.1: Ordering information flow properties (cf. Theorem 4.3.1)

For future reference, the ordering of these information flow properties is summarized in the following theorem.

Theorem 4.3.1. Let $UI \subseteq I$ be a set of user inputs. The following implications are valid:

1. $SEP(ES) \Rightarrow NDO^*(ES)$,
2. $NDO^*(ES) \Rightarrow PSP(ES)$, and
3. $PSP(ES) \Rightarrow NF(ES)$. ◇

Proof. These implications follow from the representations of *SEP*, NDO^*, *PSP*, and *NF* in *MAKS* and the taxonomies of BSPs (cf. Theorems 3.5.3 and 3.5.12). □

Given the ordering, a natural question is why one should enforce a more restrictive information flow property rather than a less restrictive one. What is gained by enforcing *SEP* instead of NDO^*; what is gained by enforcing NDO^* instead of *PSP*; and what is gained by enforcing *PSP* instead of *NF*? Table 4.6 provides a good basis for answering these kind of questions about information flow properties without having to look at the formal details of their respective representation in *MAKS* (or their original definitions). While the entries in many columns are identical for *NF*, *PSP*, NDO^*, and *SEP*, the differences between these properties are reflected in columns 6, 7, and 10. Column 7 indicates that by moving from *NF* to the more restrictive *PSP*, one gains that *deductions about nonoccurrences of high-level events are prevented to some extent.* NDO^* further limits the information that can be deduced about nonoccurrences of high-level events but it is difficult to quantify this difference more precisely. However, for *SEP* the difference to *PSP* (or NDO^*) can be easily quantified

because *SEP limits the information* about nonoccurrences of high-level events that can be deduced by an observer *to what the observer can deduce already from the system specification* without observing the running system (i.e. his static knowledge). This is what one gains when moving from *PSP* or NDO^* to *SEP* (indicated by column 6 in Table 4.6). However, there are also disadvantages when moving to a more restrictive property and the entries in column 10 provide some evidence for this. For example, *SEP* rules out information flow from the low level to the high level although such information flow is not critical wrt. security. The entry H in column 10 shows that the enabledness of high-level events, indeed, may only depend on previous occurrences of high-level events (but not on occurrences of low-level events). NDO^* permits some noncritical information flow from the low level to the high level, namely from low-level user inputs to the high level (entry $H \cup (L \cap UI)$). Finally, *PSP* does not restrict noncritical information flow at all (entry $H \cup L$). These differences between *SEP*, NDO^*, and *PSP* are due to the different parameters that are used to instantiate *BSIA* (respectively ρ_C, ρ_{UI}, and ρ_E). Apparently, other choices of this parameter are also possible in combination with the pattern $BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho}_{\mathcal{H}}(Tr)$. However, so far no other parameters than these three have been used to define information flow properties. It is the representation in *MAKS* that has revealed this possibility. The possibility to use arbitrary parameters ρ allows one to determine more precisely which deductions about nonoccurrences of confidential events are prevented and what noncritical information flow is permitted than this would be possible by choosing only between ρ_C, ρ_{UI}, and ρ_E. In other words, one obtains a finer granularity in this decision. In particular, the parameter ρ can be defined such that it closely matches the needs of a particular application (wrt. permitted noncritical information flow and wrt. preventing deductions about nonoccurrences of high-level events).

4.3.2 Information Flow Properties with the View $\mathcal{HI}$

The view $\mathcal{HI}$ is used in the representations of generalized noninterference (cf. Theorem 4.2.3), interleaving-based generalized noninterference (cf. Theorem 4.2.5), forward correctability (cf. Theorem 4.2.11), and generalized noninference (cf. Theorem 4.2.22). It is also used in the definitions of GNI^* (cf. Definition 4.2.7), $IBGNI^*$ (cf. Definition 4.2.8), and FC^* (cf. Definition 4.2.13), the novel properties that we proposed in Section 4.2. The use of $\mathcal{HI}$ means that each of these information flow properties prevents deductions about occurrences or nonoccurrences of high-level inputs but not about occurrences or nonoccurrences of high-level internal or output events. The difference between these properties lies in the BSPs from which they are assembled or, in other words, *what* deductions they rule out. Generalized noninference is assembled from a single BSP only, namely R. Since R is a BSP from the first dimension, generalized noninference only prevents deductions about *occurrences* of high-level inputs. The representations of generalized noninterference, interleaving-based generalized noninterference, and forward correctability all involve a BSP from the second dimension (i.e. I or *BSI*). Hence, these properties prevent deductions about *occurrences* of high-level inputs and, unlike generalized noninference, also about *nonoccurrences* of high-level inputs. However, these three information flow properties differ in the corrections that they permit after the perturbation (forward correctability is most rigorous among them in this respect).

Given the representation in *MAKS* it becomes a straightforward task to order these information flow properties. In Figure 4.2 one can see how *FC*, *GNI*, *IBGNI*, *GNF*, FC^*, GNI^*, and $IBGNI^*$ can be ordered by implication. This ordering can be easily justified given the representation of these properties in *MAKS* (cf. Figure 4.3) and the ordering of BSPs in each

dimension (cf. Theorems 3.5.3 and 3.5.12). The ordering reflects that *IBGNI* requires more perturbations than *GNF* (i.e. the insertion of high-level inputs), that *GNI* requires the same perturbations as *IBGNI* but permits fewer corrections (i.e. only backwards-strict corrections), and that *FC* permits even fewer corrections. Moreover, the ordering reflects that each of the novel properties $IBGNI^*$, GNI^*, and FC^* requires slightly fewer perturbations than the respective property from which it has been derived (because high-level inputs are only inserted at positions of a trace where they are ρ_C-admissible).

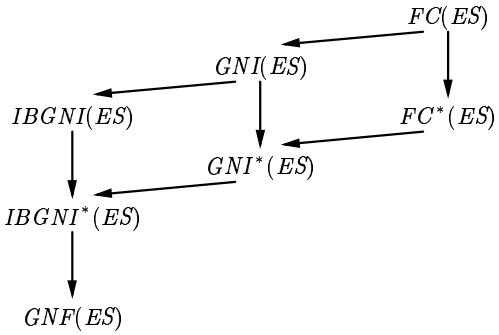


Figure 4.2: Ordering information flow properties (cf. Theorem 4.3.2)

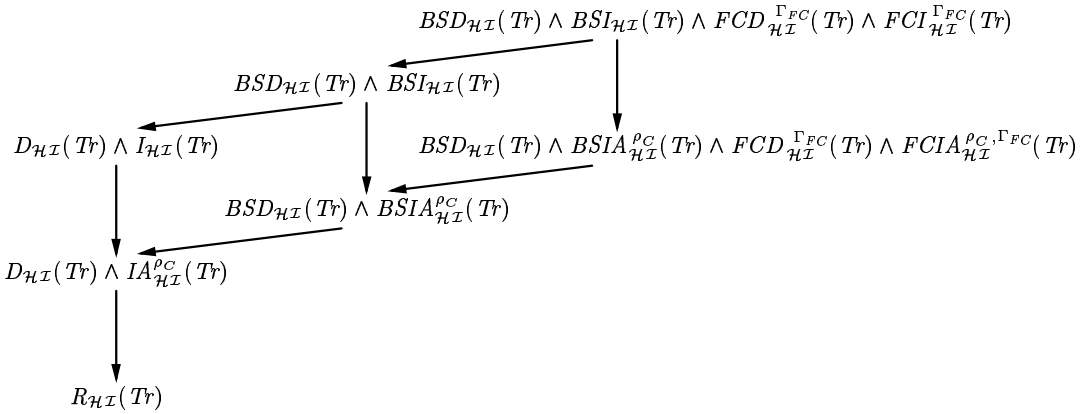


Figure 4.3: Ordering the representations (cf. Theorem 4.3.2)

For future reference, the ordering of these information flow properties is summarized in the following theorem.

Theorem 4.3.2. The following implications are valid:

1. $FC(ES) \Rightarrow GNI(ES)$,
2. $GNI(ES) \Rightarrow IBGNI(ES)$,
3. $FC(ES) \Rightarrow FC^*(ES)$,
4. $GNI(ES) \Rightarrow GNI^*(ES)$,
5. $IBGNI(ES) \Rightarrow IBGNI^*(ES)$,
6. $FC^*(ES) \Rightarrow GNI^*(ES)$,

7. $GNI^*(ES) \Rightarrow IBGNI^*(ES)$, and
8. $IBGNI^*(ES) \Rightarrow GNF(ES)$. ◇

Proof. These implications follow from the representations of *FC*, *GNI*, *IBGNI*, and *GNF* in *MAKS*, the definitions of FC^*, GNI^*, and $IBGNI^*$, and the taxonomies of BSPs (cf. Theorems 3.5.3 and 3.5.12). □

Again, we ask what is gained by enforcing a more restrictive information flow property instead of a less restrictive one. Like in Section 4.3.1, we exploit Table 4.6 for answering this question without having to look at the formal details of the representation of information flow properties in *MAKS* (or the original definitions). The entries in the first four columns are identical for *FC*, *GNI*, *IBGNI*, *GNF*, FC^*, GNI^*, and $IBGNI^*$ but the entries in columns 5–11 are not. Column 7 indicates that by moving from *GNF* to the more restrictive $IBGNI^*$ one gains that *deductions about nonoccurrences of high-level inputs are prevented to some extent.* According to column 6, $IBGNI^*$ *limits the information* that can be deduced about nonoccurrences of high-level inputs *to what an observer can deduce already from the system specification* without observing the running system. Column 5 reveals what is gained by moving from $IBGNI^*$ to *IBGNI*, namely *IBGNI prevents deductions about nonoccurrences of confidential events* (almost) *completely.* When *IBGNI* holds for a given system then the only possibility to deduce information about occurrences or nonoccurrences of high-level inputs is by exploiting the fact that *IBGNI* permits noncausal corrections (indicated by the missing check mark in column 11). In Example 3.4.22, we have illustrated how this possibility for deductions can be exploited by a Trojan horse for leaking information from the high level to the low level. By moving to the more restrictive *GNI*, this danger of information leakage is avoided (cf. column 11). In comparison to *GNI*, the advantage of *FC* is that it is a composable information flow property.[14] However, there are also disadvantages when moving to a more restrictive property and the entries in column 10 provide some evidence for this. For example, *FC* and *GNI* do not permit that the enabledness of high-level inputs depend on other occurrences of events (entry $\emptyset$). More specifically, *FC* and *GNI* require that high-level input events are always enabled (column 8). *IBGNI* is slightly less restrictive in this respect (entry $H \setminus I$), which is achieved at the price of permitting noncausal corrections (column 11). The novel properties FC^*, GNI^*, and $IBGNI^*$ have the advantage that the enabledness of high-level inputs may depend on previous occurrences of high-level inputs. Note that by moving to these properties one trades the check mark in column 5 for the check marks in columns 8 and 9 (together with a more liberal entry in column 10). Whether it is better to use an information flow property that has a check mark in column 5 or one that has check marks in columns 8 and 9 cannot be answered in general. This depends on the particular application under consideration.

4.3.3 Information Flow Properties with Different Views

In Sections 4.3.1 and 4.3.2, we have compared information flow properties to each other that enforce the same view (i.e. view $\mathcal{H}$ in Section 4.3.1 and view $\mathcal{HI}$ in Section 4.3.2). In this section, we relate these two classes of information flow properties to each other.

[14] In the table, this advantage of *FC* is not reflected. Compositionality of information flow properties will be the topic of Chapter 6.

In general, properties in the second class (enforcing view $\mathcal{HI}$) do not imply properties in the first class (enforcing view $\mathcal{H}$) because the view $\mathcal{HI}$ imposes fewer restrictions on the permitted flow of information than the view $\mathcal{H}$ does. However, properties in the first class might imply properties in the second class. Let us now investigate more closely which implications are valid.

Noninference and generalized noninference are both assembled from a single BSP only, namely R. The difference between these properties lies in the view that they enforce. Since noninference enforces the view $\mathcal{H}$, it prevents deductions about occurrences of high-level inputs as well as about occurrences of high-level internal and output events. In contrast to this, generalized noninference prevents only deductions about occurrences of high-level inputs. This already suggests that noninference is a more restrictive property than generalized noninference. The following theorem shows that this is, indeed, the case.

Theorem 4.3.3. If $NF(ES)$ holds then $GNF(ES)$ also holds. ◇

Proof. This implication follows from the representations of *NF* and *GNF* in *MAKS* and the taxonomy of BSPs in the first dimension (cf. Theorem 3.5.3). □

By transitivity of logical implication, we obtain from Theorems 4.3.1 and 4.3.3 that *PSP*, NDO^*, and *SEP* also imply *GNF*. Let us now try to establish a relation between *SEP*, NDO^*, *PSP*, *NF* and our novel properties FC^*, GNI^*, $IBGNI^*$. As we will show, the connection between these two sets of properties is not very close because only the strongest property from the first set (i.e. *SEP*) implies the weakest property from the second set (i.e. $IBGNI^*$). The following example demonstrates that *SEP*, NDO^*, *PSP*, *NF* do not imply GNI^* or FC^*.

Example 4.3.4. Recall the system PIPE from Example 3.4.15. For this system, we have $\mathcal{HI} = (\emptyset, \{ho_n \mid n \in \mathbb{N}\}, \{hi_n \mid n \in \mathbb{N}\})$ and $\mathcal{H} = (\emptyset, \emptyset, \{hi_n, ho_n \mid n \in \mathbb{N}\})$. Obviously, $SEP(\texttt{PIPE})$ holds. From Theorem 4.3.1, we obtain that $NDO^*(\texttt{PIPE})$, $PSP(\texttt{PIPE})$, and $NF(\texttt{PIPE})$ also hold. However, $GNI^*(\texttt{PIPE})$ does not hold because $\langle hi_1.hi_2\rangle$ is a perturbation of $\langle hi_1\rangle$ (resulting from the insertion of hi_2) but there is no causal correction of $\langle hi_1.hi_2\rangle$ to a possible trace of PIPE. From Theorem 4.3.2, we obtain that $FC^*(\texttt{PIPE})$ also does not hold.

The reason why $SEP(\texttt{PIPE})$ holds despite $GNI^*(\texttt{PIPE})$ not holding is that $\rho_C(\mathcal{H}) = \{hi_n, ho_n \mid n \in \mathbb{N}\}$ differs from $\rho_C(\mathcal{HI}) = \{hi_n \mid n \in \mathbb{N}\}$. Hence, GNI^* require the insertion of hi_2 immediately after hi_1 in the trace $\langle hi_1\rangle$ (hi_2 is ρ_C-admissible after hi_1 wrt. the view $\mathcal{HI}$) but *SEP* does not require that hi_2 can be inserted at this position of this trace (hi_2 is *not* ρ_C-admissible after hi_1 wrt. the view $\mathcal{H}$). ◇

The following theorem shows that, in contrast to GNI^* and FC^*, $IBGNI^*$ is implied by *SEP*.

Theorem 4.3.5. If $SEP(ES)$ holds then $IBGNI^*(ES)$ also holds. ◇

Proof. According to Theorem 4.2.24, we have $SEP(ES) \Leftrightarrow (BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_C}(Tr))$. According to Definition 4.2.8, we have $IBGNI^*(ES) \Leftrightarrow (D_{\mathcal{HI}}(Tr) \wedge IA_{\mathcal{HI}}^{\rho_C}(Tr))$.

From Theorem 3.5.3, we obtain $BSD_{\mathcal{H}}(Tr) \Rightarrow D_{\mathcal{HI}}(Tr)$ because $H \cap I \subseteq H$ holds (choose $\mathcal{V}_1 = \mathcal{H}$ and $\mathcal{V}_2 = \mathcal{HI}$ in Theorem 3.5.3).

It remains to be shown that $(BSD_{\mathcal{H}}(Tr) \wedge BSIA_{\mathcal{H}}^{\rho_C}(Tr)) \Rightarrow IA_{\mathcal{HI}}^{\rho_C}(Tr)$ also holds. Assume $BSD_{\mathcal{H}}(Tr)$ and $BSIA_{\mathcal{H}}^{\rho_C}(Tr)$. Let $\alpha, \beta \in E^*$ and $hi \in H \cap I$ such that $\beta.\alpha \in Tr$, $\alpha|_{H\cap I} = \langle\rangle$, and $Adm_{\mathcal{HI}}^{\rho_C}(Tr, \beta, hi)$. From $BSD_{\mathcal{H}}(Tr)$, we obtain $(\beta.\alpha)|_L \in Tr$ (inductive argument). $Adm_{\mathcal{HI}}^{\rho_C}(Tr, \beta, hi)$ implies that there is a trace $\gamma \in E^*$ with $\gamma.\langle hi\rangle \in Tr$ and $\gamma|_{H\cap I} = \beta|_{H\cap I}$.

We now insert the sequence $(\gamma.\langle hi\rangle)|_H$ into $(\beta.\alpha)|_L$ in a stepwise manner. $BSIA^{\rho_C}_{\mathcal{H}}(Tr)$ implies that there is a trace $\beta' \in E^*$ with $\beta'|_{L\cup(H\cap I)} = \beta|_{L\cup(H\cap I)}$, $\beta'|_H = \gamma|_H$, and $\beta'.\langle hi\rangle.(\alpha|_L) \in Tr$ (detailed argument is by induction over the length of $(\gamma.\langle hi\rangle)|_H$). From $\beta'.\langle hi\rangle.(\alpha|_L) \in Tr$, $(\alpha|_L)|_L = \alpha|_L$, $(\alpha|_L)|_{H\cap I} = \langle\rangle$, $\beta'|_{L\cup(H\cap I)} = \beta|_{L\cup(H\cap I)}$, and Definition 3.4.11, we conclude that $IA^{\rho_C}_{\mathcal{HI}}(Tr)$ holds (choose $\alpha' = \alpha|_L$). □

Recall that NDO^* and $IBGNI^*$ are defined by $BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho_{UI}}_{\mathcal{H}}(Tr)$ and $R_{\mathcal{HI}}(Tr) \wedge IA^{\rho_C}_{\mathcal{HI}}(Tr)$, respectively. This means, NDO^* requires the insertion of high-level events only at positions where they are ρ_{UI}-admissible (wrt. the view $\mathcal{H}$) while $IBGNI^*$ requires the insertion of high-level inputs at positions where they are ρ_C-admissible (wrt. the view $\mathcal{HI}$). Since $\rho_{UI}(\mathcal{H}) \subseteq \rho_C(\mathcal{HI})$ does *not* hold, in general, NDO^* does not imply $IBGNI^*$. Moreover, since *PSP* and *NF* are weaker than NDO^*, they also do not imply $IBGNI^*$.

Let us now try to establish a relationship between *SEP*, NDO^*, *PSP*, *NF* and *FC*, *GNI*, *IBGNI*. The entries in column 9 in Table 4.6 show that *FC*, *GNI*, and *IBGNI* require all sequences of high-level inputs to be possible. Moreover, the entries in column 8 reveal that *FC* and *GNI* even require high-level input events to be always enabled, i.e. the system under consideration must be *total* in the set $H \cap I$. The properties *SEP*, NDO^*, *PSP*, and *NF* do not impose such requirements. Therefore, none of *SEP*, NDO^*, *PSP*, and *NF* implies any of *FC*, *GNI*, and *IBGNI*, in general. However, if we restrict our attention to systems that are total in $H \cap I$ then *PSP* implies *FC*. This is shown by the following theorem.

Theorem 4.3.6. If $PSP(ES)$ and $total(ES, H \cap I)$ hold then $FC(ES)$ also holds. ◇

Proof. According to Theorem 4.2.26, we have $PSP(ES) \Leftrightarrow (BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho_E}_{\mathcal{H}}(Tr))$. From Theorem 4.2.11, we obtain $FC(ES) \Leftrightarrow (BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr) \wedge FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \wedge FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr))$.

From Theorem 3.5.3, we obtain $BSD_{\mathcal{H}}(Tr) \Rightarrow BSD_{\mathcal{HI}}(Tr)$ because $H \cap I \subseteq H$ holds (choose $\mathcal{V}_1 = \mathcal{H}$ and $\mathcal{V}_2 = \mathcal{HI}$ in Theorem 3.5.3).

It remains to prove the following three implications:

$$(BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho_E}_{\mathcal{H}}(Tr) \wedge total(ES, H \cap I)) \Rightarrow BSI_{\mathcal{HI}}(Tr) \quad (4.4)$$

$$(BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho_E}_{\mathcal{H}}(Tr) \wedge total(ES, H \cap I)) \Rightarrow FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \quad (4.5)$$

$$(BSD_{\mathcal{H}}(Tr) \wedge BSIA^{\rho_E}_{\mathcal{H}}(Tr) \wedge total(ES, H \cap I)) \Rightarrow FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \quad (4.6)$$

For proving (4.4), assume $BSD_{\mathcal{H}}(Tr)$, $BSIA^{\rho_E}_{\mathcal{H}}(Tr)$, and $total(ES, H \cap I)$. Let $\alpha, \beta \in E^*$ and $hi \in H \cap I$ such that $\beta.\alpha \in Tr$ and $\alpha|_{H\cap I} = \langle\rangle$. From $BSD_{\mathcal{H}}(Tr)$, we obtain $\beta.(\alpha|_L) \in Tr$ (inductive argument). $Adm^{\rho_E}_{\mathcal{H}}(Tr, \beta, hi)$ holds because $total(ES, H \cap I)$ and $hi \in H \cap I$ hold. $BSIA^{\rho_E}_{\mathcal{H}}(Tr)$ implies $\beta.\langle hi\rangle.(\alpha|_L) \in Tr$. From $\beta.\langle hi\rangle.(\alpha|_L) \in Tr$, $(\alpha|_L)|_L = \alpha|_L$, $(\alpha|_L)|_{H\cap I} = \langle\rangle$, and Definition 3.4.24, we conclude that $BSI_{\mathcal{HI}}(Tr)$ holds (choose $\alpha' = \alpha|_L$).

For proving (4.5), assume $BSD_{\mathcal{H}}(Tr)$, $BSIA^{\rho_E}_{\mathcal{H}}(Tr)$, and $total(ES, H \cap I)$. Let $\alpha, \beta \in E^*$, $hi \in H \cap I$, and $li \in L \cap I$ such that $\beta.\langle hi.li\rangle.\alpha \in Tr$ and $\alpha|_{H\cap I} = \langle\rangle$. From $BSD_{\mathcal{H}}(Tr)$, we obtain $\beta.\langle hi.li\rangle.(\alpha|_L) \in Tr$ (inductive argument). $BSD_{\mathcal{H}}(Tr)$ implies $\beta.\langle li\rangle.(\alpha|_L) \in Tr$. From $\beta.\langle li\rangle.(\alpha|_L) \in Tr$, $(\alpha|_L)|_L = \alpha|_L$, $(\alpha|_L)|_{H\cap I} = \langle\rangle$, and Definition 3.4.25, we conclude that $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$ holds (choose $\alpha' = \alpha|_L$ and $\delta' = \langle\rangle$).

For proving (4.6), assume $BSD_{\mathcal{H}}(Tr)$, $BSIA^{\rho_E}_{\mathcal{H}}(Tr)$, and $total(ES, H \cap I)$. Let $\alpha, \beta \in E^*$, $hi \in H \cap I$, and $li \in L \cap I$ such that $\beta.\langle li\rangle.\alpha \in Tr$ and $\alpha|_{H\cap I} = \langle\rangle$. From $BSD_{\mathcal{H}}(Tr)$, we obtain $\beta.\langle li\rangle.(\alpha|_L) \in Tr$ (inductive argument). $Adm^{\rho_E}_{\mathcal{H}}(Tr, \beta, hi)$ holds because $total(ES, H \cap I)$ and

$hi \in H \cap I$. $BSIA^{\rho_E}_{\mathcal{H}}(Tr)$ implies $\beta.\langle hi.li\rangle.(\alpha|_L) \in Tr$. From $\beta.\langle hi.li\rangle.(\alpha|_L) \in Tr$, $(\alpha|_L)|_L = \alpha|_L$, $(\alpha|_L)|_{H\cap I} = \langle\rangle$, and Definition 3.4.25, we conclude that $FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$ holds. □

By transitivity of logical implication, we obtain from Theorems 4.3.1, 4.3.2, and 4.3.6 that, under the assumption that high-level input events are always enabled, each of *PSP*, NDO^*, and *SEP* implies *IBGNI*, *GNI*, and *FC*.

4.3.4 A Taxonomy of Information Flow Properties

We are now ready to present the main technical result of this section, i.e. the taxonomy of all information flow properties that we have considered in this thesis so far.

Theorem 4.3.7 (Taxonomy of information flow properties). Let $UI \subseteq I$ be a set of user inputs. The implications depicted in Figure 4.4 are valid. ◇

Proof. Follows from Theorems 4.3.1, 4.3.2, 4.3.3, 4.3.5, and 4.3.6. □

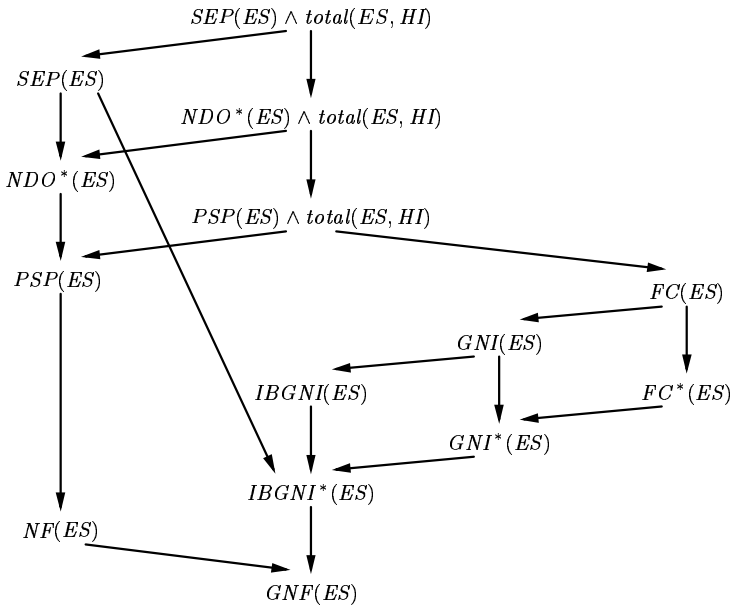


Figure 4.4: Taxonomy of information flow properties (cf. Theorem 4.4)

Remark 4.3.8. Theorem 4.3.7 subsumes the taxonomy of information flow properties in [McL94a, McL96] that only covers *GNF*, *IBGNI*, *NF*, and *SEP*. Our taxonomy also subsumes the taxonomy in [ZL97] that covers *GNF*, *IBGNI*, *NF*, *SEP*, *PSP*, and *NDO* (but not any of *FC*, *FC**, *GNI*, *GNI**, *IBGNI**). Interestingly, Zakinthinos and Lee missed that *PSP* implies *NF* and that *NDO* implies *NF*, facts pointed out by our taxonomy.[15] ◇

[15] The original definition of *NDO* is equivalent to the conjunction of two conditions $ND1 \wedge ND2$ where *ND*1 prevents a low-level user from deducing that certain high-level events have or have not occurred and *ND*2 prevents a low-level user from deducing the interleaving of high-level and low-level events (cf. [GN88] for the formal definition of *ND*1 and *ND*2). However, the definition in [ZL97] is equivalent to *ND*1 and, in general, *ND*1 does not imply *ND*2. Consequently, the definition of *NDO* in [ZL97] does not properly reflect the original definition. That the implication $NDO(ES) \Rightarrow NF(ES)$ is left out in [ZL97] might be due to this fact.

Moreover, all facts about information flow properties that we have read off their respective representation in *MAKS* are summarized in Table 4.6. Note that there are two blocks of information flow properties in the table: firstly, the properties that enforce the view $\mathcal{H}$ (having check marks in the first three columns) and, secondly, properties that enforce the view $\mathcal{HI}$ (having check marks in the first column but not in columns 2 and 3). Interestingly, all properties are assembled from a BSP in the first dimension (check mark in column 4). For the representation of two properties, namely *NF* and *GNF*, no BSP in the second dimension was necessary (no check marks in columns 5–7). However, a BSP from the second dimension is used in the representations of all other information flow properties. Hence, these properties prevent deductions about nonoccurrences of confidential events to some extent. For three properties, namely *NDO**, *NDO* and *PSP*, it is difficult to quantify more precisely what deductions about nonoccurrences of confidential events are prevented (check mark in column 7 but not in columns 5 and 6). For four properties, namely *SEP*, *FC**, *GNI** and *IBGNI**, the information that an observer can deduce about nonoccurrences of confidential events is limited to what he can already deduce from the system specification alone without observing the running system (check marks in columns 6 and 7 but not in column 5). Finally, there are three properties, namely *FC*, *GNI* and *IBGNI*, that prevent deductions about nonoccurrences of confidential events completely (check marks in columns 5–7).[16] Note also that the entries in column 9 are inverse to the entries in column 5. This means, when assembling an information flow property one has to decide whether one wants either to rule out deductions about nonoccurrences of confidential events completely (check mark in column 5) or to permit noncritical information flow (check mark in column 9). Moreover, if there is a check mark in each of the first three columns then there is also one in column 11. However, if there are no check marks in column 2 and 3 then nevertheless, there may be a check mark in column 11 (cf. the entries for *FC*). This means, if one wants to define an information flow property with a check mark in column 11 then one should use the view $\mathcal{H}$ or, alternatively, should assemble the security predicate only from backwards-strict BSPs.

4.4 Summary and Comparison to Prior Frameworks

In this chapter, we have illustrated how information flow properties can be represented in *MAKS*. This has revealed several interesting facts about the various properties that, in many cases, are not as obvious from the original definitions of these properties. The representation in *MAKS* has also provided a suitable basis for the comparison of information flow properties.

For each information flow property, we have showed whether it only prevents deductions about occurrences of confidential events (only a BSP from the first dimension) or also about nonoccurrences of confidential events (BSPs from both dimensions). We have further classified the latter properties into ones that rule out deductions about nonoccurrences of confidential events completely, ones that limit the information that can be deduced to what an observer could obtain already from the system specification, and ones that are even more liberal in this respect (depending on the particular BSP from the second dimension). We have also clarified which information flow properties consider all high-level events as confidential (view $\mathcal{H}$) and which properties only consider the high-level input events as confidential (view $\mathcal{HI}$). Moreover, we have pointed out to what extent the various information flow properties limit

[16] For *IBGNI*, this statement is softened by the missing check mark in column 11 indicating that noncausal corrections permitted by *IBGNI* could be exploited for information leakage as illustrated in Example 3.4.22.

noncritical information flow (i.e. from low to high) as an undesirable side effect of preventing critical information flow (i.e. from high to low). All these facts are summarized in Table 4.6. In this table one can also see that permitting noncritical information flow is incompatible with ruling out deductions about nonoccurrences of confidential events completely as the entries in column 5 and 9 are inverse to each other. Moreover, the comparison of information flow properties has resulted in a taxonomy of these properties (cf. Figure 4.4).

The information flow properties GNI^*, $IBGNI^*$, FC^*, and NDO^* are another outcome of our investigation. We have derived them from known properties in order to overcome certain limitations: e.g. NDO^* is applicable to systems that are not input total (unlike NDO) and FC^* is applicable to systems that are not total in the high-level input events (unlike FC).

The list of known properties that we have represented in *MAKS* includes all (non-obsolete) properties that have been represented in the prior frameworks for (trace-based) information flow properties by McLean [McL94a, McL96], Focardi/Gorrieri [FG95],[17] and Zakinthinos/Lee [Zak96, ZL97]. However, we have also represented properties in *MAKS* that have not been represented in these prior frameworks. For example, forward correctability has not been considered in [McL94a, McL96], the perfect security property has neither been considered in [McL94a, McL96] nor in [FG95], and the original variant of nondeducibility for outputs has not been represented in any of these frameworks (cf. Table 4.7 for a more detailed comparison).[18] By representing all these properties in *MAKS*, we have demonstrated that our framework is quite *expressive*.[19]

All information flow properties have been represented in the same way, i.e. by specifying a set of views and a set of BSPs (cf. Definition 3.2.6). This *uniform* representation has been helpful for the comparison of the represented properties and will also *simplify* our investigations in subsequent chapters. This means that *MAKS* combines expressiveness with a uniform representation that simplifies reasoning about information flow properties. In contrast to this, prior frameworks either emphasized expressiveness (at the cost of uniformity and simplification) or uniformity (at the cost of expressiveness). For example, the framework by Focardi and Gorrieri [FG95], in principle, is expressive enough to represent the well known information flow properties. However, this framework lacks uniform concepts that are specifically targeted at the representation of information flow properties and that would simplify the investigation of these properties. Uniformity is limited to the use of a common system model (labeled transition systems) and a common specification formalism (the process algebra SPA). In [PWK96], information flow properties are represented in a many-sorted predicate logic. Hence, the framework is quite expressive (albeit only four information flow properties actually have been considered in that article) but provides little support for reasoning about the represented properties. In contrast to this, the framework by Zakinthinos and Lee [Zak96, ZL97] provides concepts that are specifically targeted at the representation

[17] The mapping of the names used by Focardi and Gorrieri for information flow properties to the established names is as follows: *NNI* corresponds to generalized noninference, *SNNI* and *NDC* correspond to noninference, *TNDI* corresponds to nondeducibility, *lts-RES* corresponds to restrictiveness, *lts-FC* corresponds to forward correctability, and *NDCIT* corresponds to the conjunction of generalized noninference and input totality.

[18] For completeness: An early comparison of information flow properties by Bieber and Cuppens covered generalized noninference, generalized noninterference, nondeducibility, and causality [BC92]. To date, nondeducibility is an obsolete security property and it is known that causality is restricted to deterministic systems. In [PWK96], Peri, Wulf and Kienzle recast McLean's comparison in a many-sorted predicate logic, which seems not to reveal any interesting additional insights about the investigated properties. Focardi and Gorrieri investigated also nondeducibility and restrictiveness in their comparison. Both properties are obsolete to date.

[19] However, we do not know at this stage whether it is complete in any theoretical sense.

	set C in the view			BSPs from which dimensions and ρ-admissibility							strict BSPs
	information flow from high inputs to low is prevented	information flow from high internals to low is prevented	information flow from high outputs to low is prevented	deductions about occurrences of events in C prevented	deductions about nonocc. of events in C *prevented completely*	deductions about nonocc. of events in C *limited to static knowledge*	deductions about nonocc. of events in C *prevented to some extent*	totality in high inputs not required	not all sequences of events in C need to be possible	set of events on which enabledness of events in C may depend	information leakage because of noncausal corrections is avoided
SEP	√	√	√	√		√	√	√	√	H	√
NDO	√	√	√	√			√		√	$H \cup (L \cap UI)$	√
*NDO**	√	√	√	√			√	√	√	$H \cup (L \cap UI)$	√
PSP	√	√	√	√			√	√	√	$H \cup L$	√
NF	√	√	√	√				√	√	$H \cup L$	√
FC	√			√	√	√	√			$\emptyset$	√
GNI	√			√	√	√	√			$\emptyset$	√
IBGNI	√			√	√	√	√	√		$H \setminus I$	
*FC**	√			√		√	√	√	√	$H \cap I$	√
*GNI**	√			√		√	√	√	√	$H \cap I$	√
*IBGNI**	√			√		√	√	√	√	H	
GNF	√			√				√	√	$H \cup L$	

Table 4.6: Facts about information flow properties in summary

	represented in framework by McLean	represented in framework by Focardi and Gorrieri	represented in framework by Zakinthinos and Lee	represented in *MAKS*
	properties protecting occurrences and nonoccurrences of all high-level events			
SEP	√		√	√
NDO			$\times_1$	√
*NDO**				√
PSP			√	√
	properties protecting occurrences of all high-level events			
NF	√	√	√	√
	properties protecting occurrences and nonoccurrences of high-level inputs			
FC		√	√	√
*FC**				√
GNI			√	√
*GNI**				√
IBGNI	$\surd^{\times_2}$		√	√
*IBGNI**	$\surd^{\times_2}$			√
	properties protecting occurrences of high-level inputs			
GNF	√	√	√	√

$\times_1$ A variant of nondeducibility for outputs has been considered in [ZL97] but its definition is neither equivalent to the original definition nor is it a sensible security property (cf. Remark 4.3.8).

$\times_2$ In [ZL97], *IBGNI* has been introduced as natural adaptation of McLean's definition of generalized noninterference to the event-based setting. However, to us *IBGNI** seems at least as natural (cf. Remark 4.2.6).

Table 4.7: Information flow properties represented in the various frameworks

of information flow properties (low-level equivalence sets and a uniform schema for information flow properties). However, these concepts seem not to provide enough structure for general reasoning about classes of properties. Moreover, the use of these concepts is not enforced by the framework (e.g. the representation of separability and the perfect security property in [ZL97] do not comply with the schema). Among the previously proposed frameworks for information flow properties, McLean's framework of selective interleaving functions [McL94a, McL96] seems to be the only one that considerably simplifies the investigation of information flow properties. In that framework, information flow properties are represented in a uniform way based on a concept of selective interleaving functions where each selective interleaving function belongs to one or more *types*. These types can be used, e.g., to simplify the comparison of information flow properties or the derivation of compositionality results. However, selective interleaving functions are not expressive enough to represent, e.g., nondeducibility for outputs and the perfect security property [Zak96]. More specifically, closure under a set of selective interleaving functions of a particular type is equivalent to a statement of the form $R_{\mathcal{V}}(Tr) \wedge IA^{\rho}_{\mathcal{V}}(Tr)$ in *MAKS*. By introducing the concept of domain restrictions (or, alternatively, the concept of range restrictions) it becomes possible to represent additional properties but the expressiveness is still quite limited. Properties that are not of the

form $R_{\mathcal{V}}(Tr)$ or $R_{\mathcal{V}}(Tr) \wedge IA^{\rho}_{\mathcal{V}}(Tr)$ cannot be represented in McLean's framework of selective interleaving functions (in its current form).[20]

In our opinion, it is crucial for a framework for information flow properties to be expressive, to facilitate the uniform representation of information flow properties, and to simplify their investigation. *Expressiveness* is important because if an information flow property of interest cannot be expressed in a framework then it, obviously, cannot be investigated in this framework. *Uniformity* is important because if the representations of different information flow properties are completely dissimilar then their representation in the framework provides little help for improving ones understanding of these properties. Finally, to *simplify the investigation* of information flow properties, apparently, is the main objective when developing a framework for their investigation. As we have demonstrated in this chapter, *MAKS* combines these three desirable properties and, in this respect, it is an improvement over all previous frameworks for information flow properties.

[20]These statements are justified in Appendix B where we investigate the relationship between McLean's framework and *MAKS* in more detail.

Chapter 5

Verification Techniques for Information Flow Properties

5.1 Introduction

Once a system's behavior and its security requirements have been formally specified, the security of this system can be verified with mathematical rigor. Naturally, verifying that a system satisfies its requirements is often difficult and time consuming.

In this chapter, we propose verification techniques for information flow properties. Our main goal is to simplify the proof that a given system satisfies a given information flow property. We show that part of this proof can be done independently of the particular system. To this end, we follow the unwinding approach [GM84, HY87, Jac90, Rya91, GCS91, MC92, Rus92, Mil94, Pin95, Zak96, RS99]. However, we go one step further than the traditional unwinding techniques by showing that most of this proof can be constructed also independently of the particular information flow property. This simplifies the verification considerably because part of the proof that a system satisfies an information flow property can be factored out and needs to be constructed only once for all systems and all information flow properties. Our framework *MAKS* provides the uniform basis for this effort.

More specifically, we reduce the verification of an information flow property to the verification of all BSPs from which this property is assembled. Then, we divide the verification of each BSP into two parts. To this end, we introduce *unwinding conditions* that serve as an intermediate specification in this process (cf. Figure 5.1). Typical requirements imposed

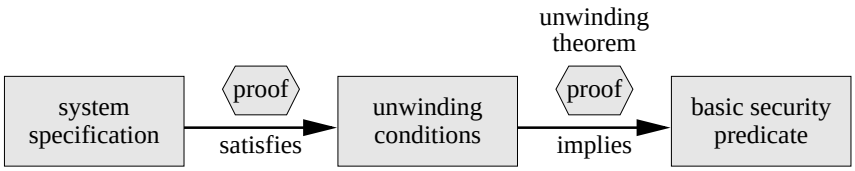


Figure 5.1: Proving a BSP by unwinding

by unwinding conditions are that the states[1] before and after an occurrence of a confidential event are related by the *unwinding relation*, a binary relation between states, or that the unwinding relation is preserved under stepwise execution. *Unwinding theorems* ensure that these

[1] The system model used in this chapter is that of state-event systems (cf. Definition 2.1.13).

unwinding conditions indeed imply the various BSPs by inductive arguments. This means, our unwinding theorems reduce the verification of BSPs (i.e. global requirements expressed in terms of sets of traces) to the verification of unwinding conditions (i.e. local requirements expressed in terms of single transitions). In other words, part of the argument that a system satisfies a given BSP is independent of the particular system specification. This proof is also independent of the particular information flow property because our unwinding theorems aim at the verification of BSPs and, hence, can be applied during the verification of all information flow properties that can be assembled from these BSPs. Consequently, the tedious derivation of specialized unwinding theorems for individual information flow properties is made obsolete by our results.

However, specialized unwinding theorems for individual information flow properties also can be derived from our unwinding theorems. We illustrate this for several properties, including some for which no such theorems were available previously (like, e.g., for nondeducibility for outputs and separability). For other properties (like, e.g., for the perfect security property), our unwinding theorems constitute improvements over the known theorems. All of these unwinding theorems are simple consequences of our unwinding theorems for BSPs. More generally, by representing an information flow property in *MAKS*, one obtains a corresponding specialized unwinding result for this property (almost) "for free".

A technical novelty of our unwinding results is that we permit arbitrary unwinding relations rather than only equivalence relations (like, e.g., in [HY87, Rya91, GCS91, Rus92, Mil94, Zak96, RS99]). We have experienced in a case study [MSK+01] how complicated and inconvenient the situation can be, if we are restricted to equivalence relations. However, our relaxation has not only practical but also some more fundamental advantages because certain information flow properties (like, e.g., noninference) can now be captured more closely by unwinding conditions.

This chapter is complemented by Appendix C where we propose further verification techniques for information flow properties as alternatives to unwinding. More specifically, we show how simulation techniques like, e.g. forward simulation or backwards simulation, can be employed for this purpose. These simulation techniques have been originally developed for verifying that a specification *refines* another specification.

Overview. In Section 5.2, we illustrate informally the derivation of unwinding results for BSPs. Unwinding theorems that cover all BSPs from Chapter 3 are presented in Section 5.3. They are the main technical results of this chapter. In Section 5.4, we formally define the various unwinding conditions and prove all cases of the unwinding theorems. Based on the unwinding theorems for BSPs, we derive several specialized unwinding theorems for information flow properties in Section 5.5. In Section 5.6, we illustrate how to verify that a given system specification satisfies our unwinding conditions and thereby demonstrate the advantages of being able to use arbitrary unwinding relations. The main results of this chapter are summarized in Section 5.7.

Notational Conventions. Let $ES = (E, I, O, Tr)$ denote an event system and $SES = (S, s_0, E, I, O, T)$ denote a state-event system. $\mathcal{V} = (V, N, C)$ denotes a view in E, ρ denotes a function from views in E to subsets of E, and $\Gamma = (\nabla, \Delta, \Upsilon)$ is a triple where $\nabla, \Delta, \Upsilon \subseteq E$. Moreover, α and β denote finite traces, i.e. $\alpha, \beta \in E^*$, and c denotes a confidential event, i.e. $c \in C$. Finally, let $\mathcal{HI} = (L, H \setminus I, H \cap I)$, $\mathcal{H} = (L, \emptyset, H)$, and $\Gamma_{FC} = (I, \emptyset, I)$.

5.2 On the Derivation of Unwinding Results

We now illustrate the derivation of unwinding results for BSPs using *BSD* as an example. For simplicity, we assume that $N = \emptyset$ holds for the view $\mathcal{V} = (V, N, C)$, a simplifying assumption made only in this section. From Section 5.3 on, we will investigate the general case.

Recall from Section 3.4 that *BSD* prevents an observer from deducing information about occurrences of confidential events. If *BSD* holds then, for every possible trace with occurrences of confidential events, there is another possible trace without confidential events that generates the same observation. This makes it impossible for an observer to tell whether a trace with confidential events or a trace without confidential events has generated a given observation. In other words, he cannot conclude that confidential events have occurred.

Under the assumption $N = \emptyset$, $BSD_{\mathcal{V}}(Tr)$ requires that if $\beta.\langle c\rangle.\alpha$ is a possible trace in Tr and α contains no confidential events then deleting the occurrence of c results in a trace, namely $\beta.\alpha$, that is also in Tr. Consequently, *BSD* is a closure property of sets of traces. Since such global conditions are not very easy to handle during verification, we would like to have more local conditions that involve only individual transitions rather than complete traces. For expressing these local conditions, it is convenient to have a notion of state. Therefore, we use the system model of state-event systems throughout this chapter.[2] For a state-event system, the requirements of *BSD* can be expressed as follows: If s is the state that results after some trace β has occurred in the initial state s_0, s' is the state that results after c has occurred in s, and s'_n is the state that results after some trace α without confidential events has occurred in s' then there must be a state s_n that results after α has occurred in s. This requirement can be viewed as follows:

$$\mathbf{s_0} \xrightarrow{\boldsymbol{\beta}}_{\mathbf{T}} \mathbf{s} \xrightarrow{\mathbf{c}}_{\mathbf{T}} \mathbf{s'} \xrightarrow{\boldsymbol{\alpha}}_{\mathbf{T}} \mathbf{s'_n} \tag{5.1}$$

$$\Rightarrow \exists s_n \in S.\quad \mathbf{s_0} \xrightarrow{\boldsymbol{\beta}}_{\mathbf{T}} \mathbf{s} \xrightarrow{\boldsymbol{\alpha}}_{\mathbf{T}} s_n$$

In (5.1), it is implicitly assumed that $\alpha, \beta \in E^*$, $c \in C$, and $s, s', s'_n \in S$ are universally quantified and that $\alpha|_C = \langle\rangle$ holds. That all states in the diagram can be reached from the initial state is expressed by the condition $s_0 \stackrel{\beta}{\Longrightarrow}_T s$, which occurs on both sides of the implication. By making it an implicit assumption that all states are reachable rather than stating it explicitly, we arrive at the following simplified diagram:

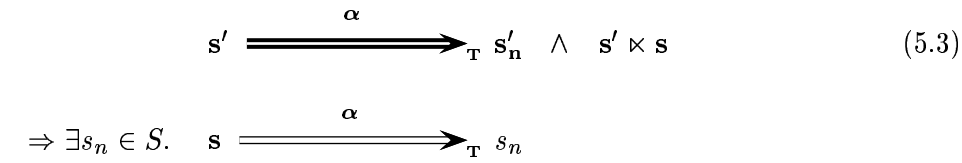
$$\mathbf{s} \xrightarrow{\mathbf{c}}_{\mathbf{T}} \mathbf{s'} \xrightarrow{\boldsymbol{\alpha}}_{\mathbf{T}} \mathbf{s'_n} \tag{5.2}$$

$$\Rightarrow \exists s_n \in S.\quad \mathbf{s} \xrightarrow{\boldsymbol{\alpha}}_{\mathbf{T}} s_n$$

Next, we split (5.2) into two conditions where one is concerned with the transition $s \xrightarrow{c}_T s'$ and the other with the transitions $s' \stackrel{\alpha}{\Longrightarrow}_T s'_n$ and $s \stackrel{\alpha}{\Longrightarrow}_T s_n$. To this end, we introduce a binary relation $\ltimes$ between states, the *unwinding relation*. The intuition is that if $s' \ltimes s$ holds then every sequence α of visible events that is enabled in state s' is also enabled in the state

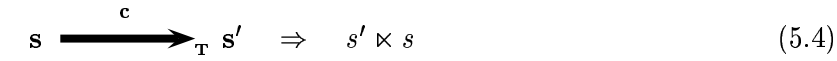
[2] Using state-event systems instead of event systems is not a major change. For each state-event system *SES* there is an event system that has the same set of possible traces, namely $ES_{SES} = (E, I, O, Tr_{SES})$ (cf. Definition 2.1.18) and for each event system $ES = (E, I, O, Tr)$ there is a corresponding state-event system, namely $SES_{ES} = (S, s_0, E, I, O, T)$ where $S = E^*$, $s_0 = \langle\rangle$, and $T = \{(s, e, s.\langle e\rangle) \mid s.\langle e\rangle \in Tr\}$.

s, which can be viewed as follows:



(5.3)

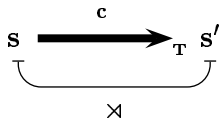
Using the unwinding relation, we can reformulate (5.2) as follows:



(5.4)

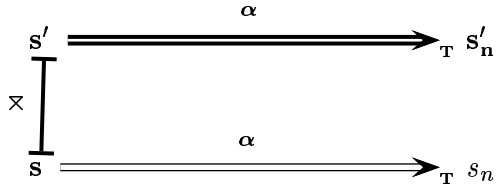
Note that (5.4) reflects the main requirement of *BSD*, namely that every sequence of visible events that is enabled after a confidential event c has occurred also must have been enabled before this occurrence.

In the diagrams, we have used boldface to mark objects that are implicitly universally quantified (e.g. s or s' in (5.3)) and we have indicated the conditions that are assumed (on the left hand side of the implication) by fat arrows and lines (e.g. $s \xrightarrow{c}_T s'$ in (5.4)). In contrast to this, existentially quantified objects have not been written in boldface (e.g. the state s_n in (5.3)) and conditions that need to be shown have been indicated by thin arrows/lines (e.g. $s \overset{\alpha}{\Longrightarrow}_T s_n$ in (5.3)). By rigorously obeying these conventions, it becomes unnecessary to state the logical connectives explicitly in the diagrams, which results in a more compact graphical notation in which (5.4) can be represented as follows:



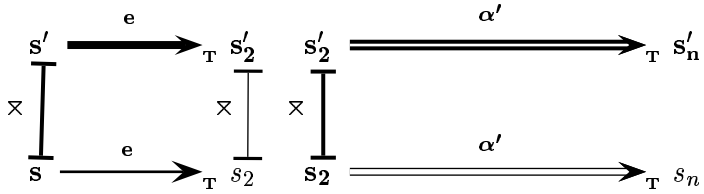
(5.5)

The compact representation of (5.3) is as follows:



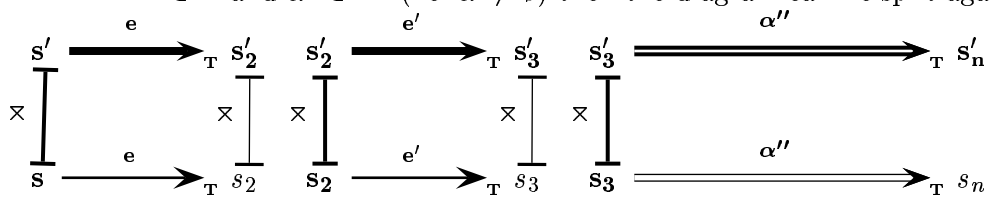
(5.6)

Note that (5.6) is trivially true if $\alpha = \langle\rangle$ holds. The interesting case is where $\alpha \neq \langle\rangle$ holds or, in other words, where there are $e \in E$ and $\alpha' \in E^*$ with $\alpha = \langle e \rangle.\alpha'$. This assumption allows us to split (5.6) into two diagrams:



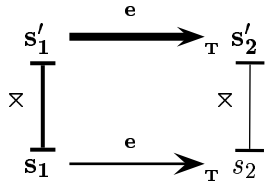
(5.7)

If $\alpha' = \langle\rangle$ holds then the condition on the right hand side of (5.7) holds trivially. If $\alpha' = \langle e' \rangle.\alpha''$ holds for some $e' \in E$ and $\alpha'' \in E^*$ (i.e. $\alpha' \neq \emptyset$) then the diagram can be split again:



(5.8)

The local conditions that result from splitting (cf. the two diagrams on the left hand side of (5.8)) are logically equivalent to each other. Hence, it does not matter how many times we have to split the diagram on the right hand side (or, in other words, what the length of α'' is). If one of the local conditions that we obtain can be proven then all of them hold. This means, it is sufficient to prove the following condition:



(5.9)

Finally, we have derived a local verification condition that implies (5.6) and that is expressed in terms of individual transitions rather than in terms of complete traces.

We are now ready to translate the graphical notation used for our two unwinding conditions, i.e. (5.5) and (5.9), into logical formulas. The following formula expresses (5.5):

$$\begin{aligned} &\forall s, s' \in S.\, \forall c \in C. \\ &((\mathit{reachable}(\mathit{SES}, s) \wedge \underline{s \xrightarrow{c}_T s'}) \Rightarrow \underline{s' \ltimes s}) \end{aligned} \tag{5.10}$$

and the following formula expresses (5.9):

$$\begin{aligned} &\forall s_1, s'_1, s'_2 \in S.\, \forall e \in E \setminus C. \\ &\left(\begin{array}{l} \mathit{reachable}(\mathit{SES}, s_1) \wedge \mathit{reachable}(\mathit{SES}, s'_1) \wedge \underline{s'_1 \xrightarrow{e}_T s'_2} \wedge \underline{s'_1 \ltimes s_1}) \\ \Rightarrow \exists s_2 \in S.\, (\underline{s_1 \xrightarrow{e}_T s_2} \wedge \underline{s'_2 \ltimes s_2}) \end{array} \right) \end{aligned} \tag{5.11}$$

In (5.10) and (5.11), we have underlined the expressions that correspond to parts of the diagrams in (5.5) and (5.9). All other expressions stem from our implicit assumptions.

If (5.10) and (5.11) can be shown then (5.1) also holds, i.e. $\mathit{BSD}_\mathcal{V}(\mathit{Tr}_{\mathit{SES}})$ holds. Hence, instead of verifying $\mathit{BSD}_\mathcal{V}(\mathit{Tr}_{\mathit{SES}})$, it suffices to prove these two unwinding conditions. The unwinding conditions are much easier to handle during verification than the BSP because they are formulated in terms of individual transitions rather than in terms of (sets of) complete traces. This is the kind of simplification we are aiming at with our verification techniques.

The purpose of this section has been to informally introduce our verification techniques and to illustrate the derivation of unwinding conditions. Our explanations are not meant as a rigorous proof that the unwinding conditions, indeed, imply $\mathit{BSD}_\mathcal{V}(\mathit{Tr})$. This is the purpose of the next sections, where we derive unwinding theorems that cover not only *BSD* but also all other BSPs.

Remark 5.2.1. In (5.10) and (5.11), the reachability of the states s, s_1, s_2 is stated explicitly by assumptions. Reasoning about the reachability of states usually involves complete traces rather than only individual transitions. However, this does not mean that it is necessary to reason about complete traces when verifying our unwinding conditions. The reachability of states can be approximated by local conditions, namely by invariants of the system.[3] In particular the assumptions about the reachability of states can be substituted by the invariant *True*, i.e. the condition that is always fulfilled. This means, the assumptions about reachability can, but need not, be exploited in the proofs of the unwinding conditions. ◇

[3] An *invariant* is a property that holds in the initial state and that is preserved under all transitions.

Remark 5.2.2. According to (5.3), if $s' \ltimes s$ holds for two states s, s' then every sequence of visible events that is enabled in s' is also enabled in s. However, this does *not* mean that $s' \ltimes s$ must hold if every sequence of visible events enabled in s' is also enabled in s because there is an implication in (5.3), not an equivalence. Consequently, although "every sequence of visible events that is enabled in $\_$ is also enabled in $\_$" defines a relation that is reflexive as well as transitive, this does not mean that $\ltimes$ also must be reflexive or transitive. Moreover, $s' \ltimes s$ obviously does not imply that $s \ltimes s'$ holds. Hence, $\ltimes$ need not be symmetric. ◇

5.3 Unwinding Theorem for Basic Security Predicates

The following theorem reduces the verification of BSPs to the verification of unwinding conditions.[4] More specifically, for each BSP, two unwinding conditions are stated that imply this BSP. The unwinding theorem states the unwinding results for all backwards-strict BSPs and for all forward-correctable BSPs. Corresponding unwinding results for strict BSPs and non-strict BSPs are presented subsequently in two theorems that are immediate consequences of the following unwinding theorem.

Theorem 5.3.1 (Unwinding theorem). Let $\ltimes \subseteq S \times S$ be an arbitrary relation. The following implications are valid:

1. $(lrf_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow BSD_{\mathcal{V}}(Tr_{SES})$
2. $(lrb_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow BSI_{\mathcal{V}}(Tr_{SES})$
3. $(lrbe^{\rho}_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow BSIA^{\rho}_{\mathcal{V}}(Tr_{SES})$
4. $(fcrf^{\Gamma}_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow FCD^{\Gamma}_{\mathcal{V}}(Tr_{SES})$
5. $(fcrb^{\Gamma}_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow FCI^{\Gamma}_{\mathcal{V}}(Tr_{SES})$
6. $(fcrbe^{\rho,\Gamma}_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr_{SES})$ ◇

Proof. Propositions 1 and 2 will be proven in Section 5.4.2, Propositions 4 and 5 will be proven in Section 5.4.3, and Propositions 3 and 6 will be proven in Section 5.4.4. □

Note that, in the unwinding theorem, the first unwinding condition differs depending on the particular BSP while the second unwinding condition is identical for all BSPs. For instance, the first unwinding condition for *BSD*, namely $lrf_{\mathcal{V}}(T, \ltimes)$, is equivalent to the condition (5.10) in Section 5.2 and the second unwinding condition for *BSD*, namely $osc_{\mathcal{V}}(T, \ltimes)$, corresponds to (5.11). We refrain from introducing the formal definitions of the various other unwinding conditions at this point. This will be done in Section 5.4 where we also present the proof of the unwinding theorem. In the remainder of the current section, let us show how to unwind other BSPs than the ones explicitly mentioned in Theorem 5.3.1.

Theorem 5.3.2 (Unwinding theorem for strict BSPs). Let $\ltimes \subseteq S \times S$ be an arbitrary relation. The following implications are valid where $\mathcal{V}' = (V \cup N, \emptyset, C)$ and $\rho'(\mathcal{V}') = \rho(\mathcal{V})$:

[4] We present the unwinding theorems in this section before defining the unwinding conditions formally in order to clarify what we are aiming for with these conditions. The unwinding conditions used in Theorems 5.3.1, 5.3.2, and 5.3.3 (like, e.g., *lrf*, *osc*, or *lrb*) will be defined in Section 5.4.

1. $(lrf_{\mathcal{V}'}(T, \ltimes) \wedge osc_{\mathcal{V}'}(T, \ltimes)) \Rightarrow SD_{\mathcal{V}}(Tr_{SES})$
2. $(lrb_{\mathcal{V}'}(T, \ltimes) \wedge osc_{\mathcal{V}'}(T, \ltimes)) \Rightarrow SI_{\mathcal{V}}(Tr_{SES})$
3. $(lrbe^{\rho'}_{\mathcal{V}'}(T, \ltimes) \wedge osc_{\mathcal{V}'}(T, \ltimes)) \Rightarrow SIA^{\rho}_{\mathcal{V}}(Tr_{SES})$
4. $(lrf_{\mathcal{V}'}(T, \ltimes) \wedge osc_{\mathcal{V}'}(T, \ltimes)) \Rightarrow SR_{\mathcal{V}}(Tr_{SES})$ ◇

Proof. From Theorem 3.5.6(2), we obtain $SD_{\mathcal{V}}(Tr) \Leftrightarrow BSD_{\mathcal{V}'}(Tr)$. From Theorem 5.3.1(1), we obtain $(lrf_{\mathcal{V}'}(T, \ltimes) \wedge osc_{\mathcal{V}'}(T, \ltimes)) \Rightarrow BSD_{\mathcal{V}'}(Tr_{SES})$. Hence, the first proposition holds. The second proposition follows from Theorems 3.5.15(1) and 5.3.1(2). The third proposition follows from Theorems 3.5.15(2) and 5.3.1(3). According to Theorem 3.5.1(4), we have $SD_{\mathcal{V}}(Tr) \Rightarrow SR_{\mathcal{V}}(Tr)$. Hence, the unwinding conditions for *SD* also imply *SR*, i.e. the fourth proposition holds. □

Theorem 5.3.3 (Unwinding theorem for non-strict BSPs). Let $\ltimes \subseteq S \times S$ be an arbitrary relation. The following implications are valid:

1. $(lrf_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow D_{\mathcal{V}}(Tr_{SES})$
2. $(lrb_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow I_{\mathcal{V}}(Tr_{SES})$
3. $(lrbe^{\rho}_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow IA^{\rho}_{\mathcal{V}}(Tr_{SES})$
4. $(lrf_{\mathcal{V}}(T, \ltimes) \wedge osc_{\mathcal{V}}(T, \ltimes)) \Rightarrow R_{\mathcal{V}}(Tr_{SES})$ ◇

Proof. According to Theorem 3.5.1(2), we have $BSD_{\mathcal{V}}(Tr) \Rightarrow D_{\mathcal{V}}(Tr)$. Hence, the unwinding conditions for *BSD* (cf. Theorem 5.3.1(1)) also imply *D*, i.e. the first proposition holds. The second proposition follows from Theorems 3.5.9(2) and 5.3.1(2). The third proposition follows from Theorems 3.5.9(4) and 5.3.1(3). According to Theorem 3.5.1(1), we have $D_{\mathcal{V}}(Tr) \Rightarrow R_{\mathcal{V}}(Tr)$. Hence, the unwinding conditions for *D* also imply *R*, i.e. the fourth proposition holds. □

Note that Theorems 5.3.1–5.3.3 cover all BSPs that we have introduced in Chapter 3. This means that it is possible to use our unwinding techniques during the verification of *every* information flow property that can be represented in *MAKS*. In particular, they can be used for verifying the information flow properties from the literature for which we have demonstrated in Chapter 4 how they can be represented in *MAKS*. They can also be used to simplify the verification of the novel properties *GNI**, *IBGNI**, *FC**, and *NDO** proposed in Chapter 4.

5.4 Unwinding Conditions and Proof of Unwinding Theorem

We shall now define the unwinding conditions that already have been mentioned in Theorems 5.3.1–5.3.3. Moreover, we complete the proof of the unwinding theorem by proving, for each unwinding condition, that the corresponding proposition in the unwinding theorem is valid. Thereby, we show that unwinding, as a proof technique, is *sound*. At the end of this section, we discuss to what extent this proof technique is *complete*, i.e. under which assumptions the unwinding conditions are also necessary conditions for the corresponding BSPs.

5.4.1 Unwinding Condition *osc*

The purpose of the unwinding condition *osc* (for output-step consistency) is to capture the intuitive idea of unwinding relations, namely that $s' \ltimes s$ implies that every observation that is possible in s' is also possible in s. Since this intuition is completely independent of the particular restrictions that a given BSP imposes on the information flow, *osc* occurs in the respective unwinding result for each BSP (cf. Theorems 5.3.1–5.3.3). The formal definition of *osc* is similar to the condition (5.11) that we have derived in Section 5.2. The difference between the two conditions stems from the fact that we made a simplifying assumption in Section 5.2, namely that $N = \emptyset$ holds for the view under consideration, and we do no longer make this assumption here.

Definition 5.4.1 (*osc*). Let $\ltimes \subseteq S \times S$ be a relation. The unwinding condition *osc* is defined as follows for $\mathcal{V}$, T, and $\ltimes$:

$$
\begin{aligned}
osc_{\mathcal{V}}(T, \ltimes) \quad \equiv \quad & \forall s_1, s_1', s_2' \in S.\, \forall e \in E \setminus C. \\
& \left(\begin{array}{l} (reachable(SES, s_1) \wedge reachable(SES, s_1') \wedge s_1' \xrightarrow{e}_T s_2' \wedge s_1' \ltimes s_1) \\ \Rightarrow \exists s_2 \in S.\, \underline{\exists \delta \in (E \setminus C)^*.\, (\delta|_V = \langle e \rangle|_V} \wedge \underline{s_1 \stackrel{\delta}{\Longrightarrow}_T s_2} \wedge s_2' \ltimes s_2) \end{array} \right) \qquad \diamond
\end{aligned}
$$

The unwinding condition *osc* is shown also in Figure 5.2 using the graphical notation that we have introduced in Section 5.2. The difference to (5.9) is that δ is used in the lower part of the diagram (instead of e). This means, if a non-confidential event e is enabled in s_1' and $s_1' \ltimes s_1$ holds then some trace δ must be enabled in s_1 that does not contain any confidential events and that yields the same observation as $\langle e \rangle$. Moreover, the resulting states, i.e. s_2' and s_2, must be related by $\ltimes$. When proving $osc_{\mathcal{V}}(T, \ltimes)$, one possible choice is to set $\delta = \langle e \rangle$. This choice trivially satisfies the conditions $\delta \in (E \setminus C)^*$ and $\delta|_V = \langle e \rangle|_V$. However, other choices for δ are possible if $N = \emptyset$ holds. Hence, $osc_{\mathcal{V}}(T, \ltimes)$ is more liberal than (5.11). To point out the differences between the two conditions, we have underlined them in Definition 5.4.1.

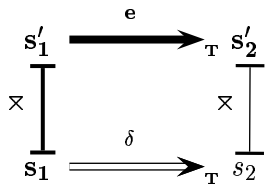


Figure 5.2: The unwinding condition *osc* (where $e \in E \setminus C$, $\delta \in (E \setminus C)^*$, $\delta|_V = \langle e \rangle|_V$)

The following lemma states that *osc* properly captures the intuition of unwinding relations. If $s_1' \ltimes s_1$ holds for two reachable states s_1, s_1' then for every trace $\alpha \in (E \setminus C)^*$ enabled in s_1' there is a trace $\alpha' \in (E \setminus C)^*$ that is enabled in s_1 and yields the same observation as α.

Lemma 5.4.2. Let $\ltimes \subseteq S \times S$ be an arbitrary relation. If $osc_{\mathcal{V}}(T, \ltimes)$ holds then the following proposition is valid:

$$
\begin{array}{l}
\forall s_1, s_1' \in S.\, \forall \alpha \in (E \setminus C)^*. \\
\left(\begin{array}{l} reachable(SES, s_1) \wedge reachable(SES, s_1')) \\ \Rightarrow \left(\begin{array}{l} (enabled(SES, s_1', \alpha) \wedge s_1' \ltimes s_1) \\ \Rightarrow \exists \alpha' \in (E \setminus C)^*.\, (\alpha'|_V = \alpha|_V \wedge enabled(SES, s_1, \alpha')) \end{array} \right) \end{array} \right) \qquad \diamond
\end{array}
$$

Proof. We prove the proposition by induction on the length of α. For $\alpha = \langle\rangle$, it holds trivially (choose $\alpha' = \langle\rangle$). In the step case, i.e. for $\alpha = \langle e_1\rangle.\alpha_1$, assume $reachable(SES, s_1)$, $reachable(SES, s_1')$, $enabled(SES, s_1', \langle e_1\rangle.\alpha_1)$, and $s_1' \ltimes s_1$. Thus, there are states $s_2', s_n' \in S$ with $s_1' \xrightarrow{e_1}_T s_2'$ and $s_2' \stackrel{\alpha_1}{\Longrightarrow}_T s_n'$. Since $e_1 \in E \setminus C$, $reachable(SES, s_1)$, $reachable(SES, s_1')$, $s_1' \xrightarrow{e_1}_T s_2'$, $s_1' \ltimes s_1$, and $osc_{\mathcal{V}}(T, \ltimes)$ hold, there are a state $s_2 \in S$ and a trace $\delta \in (E \setminus C)^*$ with $\delta|_V = \langle e_1\rangle|_V$, $s_1 \stackrel{\delta}{\Longrightarrow}_T s_2$, and $s_2' \ltimes s_2$. The induction hypothesis yields that there is a trace $\alpha'' \in (E \setminus C)^*$ with $\alpha''|_V = \alpha_1|_V$ and $enabled(SES, s_2, \alpha'')$. For $\alpha' = \delta.\alpha''$, we obtain $\alpha' \in (E \setminus C)^*$, $(\alpha')|_V = (\langle e_1\rangle.\alpha_1)|_V$, and $enabled(SES, s_1, \alpha')$. □

Remark 5.4.3. To credit a report by Rushby [Rus92] that inspired our unwinding results, we have derived the names for our unwinding conditions from the names Rushby uses for his unwinding conditions. For example, he proposes unwinding conditions with the names "*output consistency*" and "*step consistency*". The first of these conditions requires that outputs are equal if they can be generated in states that are equivalent wrt. the unwinding relation (Rushby presumes unwinding relations to be equivalence relations). The second condition requires that the unwinding relation is preserved under stepwise execution. The name "*output-step consistency*" of our unwinding condition shall indicate that it serves a similar purpose as the two unwinding conditions "output consistency" and "step consistency" in Rushby's approach. Rushby introduced a third unwinding condition with the name "*locally respects*". This condition captures the restrictions imposed by the information flow property that he investigated, namely Goguen and Meseguer's noninterference (for deterministic systems). In the following sections, we will introduce unwinding conditions with names like, e.g., "*locally-respects backwards*" or *locally-respects forwards*, that capture the requirements of the various BSPs in a way similar to how Rushby's condition "locally respects" captures the requirements of noninterference. ◇

The syntax diagram for the names of our unwinding conditions is shown in Figure 5.3.

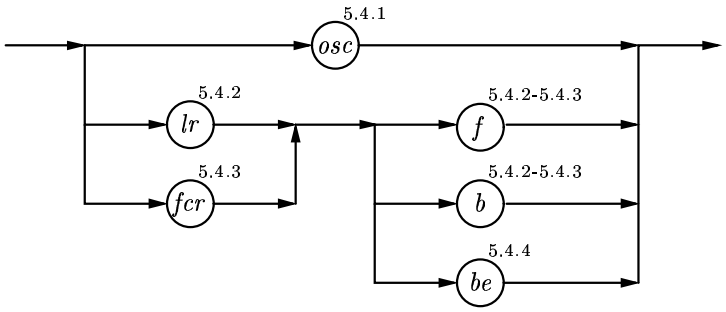


Figure 5.3: Syntax diagram for names of unwinding conditions and pointers to sections

5.4.2 Unwinding Conditions *lrf* and *lrb*

Having specified the intuition of unwinding relations with the unwinding condition *osc*, it remains to formally capture the security requirements imposed by the various BSPs. In Section 5.2, we have captured the requirements of *BSD* with the unwinding condition (5.10). This unwinding condition, although derived under the simplifying assumption $N = \emptyset$, is also appropriate for the general case. For future reference, we give this unwinding condition the name *lrf* (abbreviates *locally-respects forwards*).

In contrast to *BSD*, *BSI* prevents deductions about nonoccurrences of confidential events. Formally, it requires that the *insertion* of confidential events into possible traces results in possible traces. This means that every sequence of non-confidential events that is enabled before the occurrence of a confidential event must also be possible after the occurrence of this event. Therefore, the unwinding condition for *BSI*, i.e. *lrb* (abbreviates *locally-respects backwards*), requires that the state before the occurrence of a confidential event must be related to the state after this occurrence. Moreover, *lrb* requires that all confidential events must be enabled in all reachable states. This reflects that $BSI_\gamma(Tr_{SES})$ implies $total(ES, C)$.

The formal definitions of *lrf* and *lrb* are as follows:

Definition 5.4.4 (*lrf*, *lrb*). Let $\ltimes \subseteq S \times S$ be a relation. The unwinding conditions *lrf* (locally-respects forwards) and *lrb* (locally-respects backwards) are defined by:

$$lrf_\gamma(T, \ltimes) \equiv \forall s, s' \in S. \forall c \in C. [(reachable(SES, s) \wedge s \xrightarrow{c}_T s') \Rightarrow s' \ltimes s]$$

$$lrb_\gamma(T, \ltimes) \equiv \forall s \in S. \forall c \in C. [reachable(SES, s) \Rightarrow \exists s' \in S. (s \xrightarrow{c}_T s' \wedge s \ltimes s')] \quad \diamond$$

The unwinding conditions *lrf* and *lrb* are also shown in Figure 5.4.

Figure 5.4: The unwinding conditions *lrf* and *lrb* (where $c \in C$)

We now prove the first two propositions of Theorem 5.3.1.

Proof (of proposition 1 and 2 in Theorem 5.3.1).

1. Let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary with $\beta.\langle c \rangle.\alpha \in Tr_{SES}$ and $\alpha|_C = \langle\rangle$. There are $s_1, s_1' \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s_1$, $s_1 \xrightarrow{c}_T s_1'$, and $enabled(SES, s_1', \alpha)$. Since $lrf_\gamma(T, \ltimes)$, $reachable(SES, s_1)$, and $s_1 \xrightarrow{c}_T s_1'$ hold, we have $s_1' \ltimes s_1$. Since $enabled(SES, s_1', \alpha)$ holds, we obtain from Lemma 5.4.2 that there is a trace $\alpha' \in (E \setminus C)^*$ with $\alpha'|_V = \alpha|_V$ and $enabled(SES, s_1, \alpha')$. Consequently, $\beta.\alpha' \in Tr_{SES}$.

2. Let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary with $\beta.\alpha \in Tr_{SES}$ and $\alpha|_C = \langle\rangle$. There is a state $s_1 \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s_1$ and $enabled(SES, s_1, \alpha)$. Since $lrb_\gamma(T, \ltimes)$ and $reachable(SES, s_1)$ hold, there is a state $s_1' \in S$ such that $s_1 \xrightarrow{c}_T s_1'$ and $s_1 \ltimes s_1'$. Since $enabled(SES, s_1, \alpha)$ holds, we obtain from Lemma 5.4.2 that there is a trace $\alpha' \in (E \setminus C)^*$ with $\alpha'|_V = \alpha|_V$ and $enabled(SES, s_1', \alpha')$. Consequently, $\beta.\langle c \rangle.\alpha' \in Tr_{SES}$. $\square$

5.4.3 Unwinding Conditions *fcrf* and *fcrb*

In comparison to *BSD*, FCD^Γ requires fewer perturbations because occurrences of confidential events are deleted only if they are elements of Υ and if they occur immediately before a visible event in ∇. Moreover, FCD^Γ permits fewer corrections than *BSD* because corrections in between the position where the confidential event has been deleted and the subsequent

visible event may only be corrected in events from $N \cap \Delta$ (cf. Definition 3.4.25). Hence, the requirements imposed by FCD^Γ involve two events, namely a confidential event $c \in C \cap \Upsilon$ and a visible event $v \in N \cap \Delta$. The unwinding condition for FCD^Γ, i.e. *fcrf* (abbreviates *forward-correctably respects forwards*), also involves these two events. This unwinding condition requires that if the trace $\langle c.v \rangle$ is enabled in some reachable state s then there must be a sequence $\delta \in (N \cap \Delta)$ such that $\delta.\langle c \rangle$ is enabled in s and the resulting states are related by $\ltimes$. More specifically, if s' is the state after the occurrence of $\langle c.v \rangle$ and s'' is the state after the occurrence of $\delta.\langle c \rangle$ then $s' \ltimes s''$ must hold.

The unwinding condition for FCI^Γ, i.e. *fcrb* (abbreviates *forward-correctably respects backwards*), requires that the unwinding relation holds in the other direction. More specifically, if a visible event v is enabled in some reachable state s then for every confidential event $c \in C \cap \Upsilon$ there must be a trace $\delta \in (N \cap \Delta)$ such that $\langle c \rangle.\delta.\langle v \rangle$ is also enabled in s and the state after the occurrence of v is related to the state after the occurrence of $\langle c \rangle.\delta.\langle v \rangle$ by the unwinding relation $\ltimes$.

Like *FCD* and *FCI*, the unwinding conditions *fcrf* and *fcrb* have a triple $\Gamma = (\nabla, \Delta, \Upsilon)$ of sets as parameter. The formal definitions of these unwinding conditions are as follows:

Definition 5.4.5 (*fcrf*, *fcrb*). Let $\ltimes \subseteq S \times S$ be a relation. The unwinding conditions *fcrf* (for forward-correctably respects forwards) and $fcrb^\Gamma$ (for forward-correctably respects backwards) are defined as follows:

$$
\begin{aligned}
fcrf^\Gamma_\mathcal{V}(T, \ltimes) \quad \equiv \quad & \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla.\, \forall s, s' \in S. \\
& \left(\begin{array}{l} (reachable(SES, s) \wedge s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s') \\ \Rightarrow \exists s'' \in S.\, \exists \delta \in (N \cap \Delta)^*.\, (s \stackrel{\delta.\langle v \rangle}{\Longrightarrow}_T s'' \wedge s' \ltimes s'') \end{array} \right) \\
fcrb^\Gamma_\mathcal{V}(T, \ltimes) \quad \equiv \quad & \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla.\, \forall s, s'' \in S. \\
& \left(\begin{array}{l} (reachable(SES, s) \wedge s \stackrel{v}{\longrightarrow}_T s'') \\ \Rightarrow \exists s' \in S.\, \exists \delta \in (N \cap \Delta)^*.\, (s \stackrel{\langle c \rangle.\delta.\langle v \rangle}{\Longrightarrow}_T s' \wedge s'' \ltimes s') \end{array} \right) \qquad \diamond
\end{aligned}
$$

The unwinding conditions *fcrf* and *fcrb* are also viewed in Figure 5.5.

Figure 5.5: The unwinding conditions *fcrf* and *fcrb* (where $c \in C \cap \Upsilon$, $v \in V \cap \nabla$, $\delta \in (N \cap \Delta)^*$)

We now prove propositions 4 and 5 of Theorem 5.3.1.

Proof (of proposition 4 and 5 in Theorem 5.3.1).

1. Let $\alpha, \beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary with $\beta.\langle c.v \rangle.\alpha \in Tr_{SES}$ and $\alpha|_C = \langle\rangle$. There are $s_1, s_1' \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s_1$, $s_1 \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s_1'$, and $enabled(SES, s_1', \alpha)$.

Since $fcrf_{\mathcal{V}}^{\Gamma}(T, \ltimes)$, $reachable(SES, s_1)$, and $s_1 \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s_1'$ hold, there are a state $s_1'' \in S$ and a sequence $\delta \in (N \cap \Delta)^*$ with $s_1 \stackrel{\delta.\langle v \rangle}{\Longrightarrow}_T s_1''$ and $s_1' \ltimes s_1''$. Since $enabled(SES, s_1', \alpha)$ holds, we obtain from Lemma 5.4.2 that there is a trace $\alpha' \in (E \setminus C)^*$ with $\alpha'|_V = \alpha|_V$ and $enabled(SES, s_1'', \alpha')$. Consequently, $\beta.\delta.\langle v \rangle.\alpha' \in Tr_{SES}$.

2. Let $\alpha, \beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary with $\beta.\langle v \rangle.\alpha \in Tr_{SES}$ and $\alpha|_C = \langle\rangle$. There are states $s_1, s_1'' \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s_1$, $s_1 \stackrel{v}{\longrightarrow}_T s_1''$, and $enabled(SES, s_1'', \alpha)$. Since $fcrb_{\mathcal{V}}^{\Gamma}(T, \ltimes)$, $reachable(SES, s_1)$, and $s_1 \stackrel{v}{\longrightarrow}_T s_1''$ hold, there are a state $s_1' \in S$ and a sequence $\delta \in (N \cap \Delta)^*$ with $s_1 \stackrel{\langle c \rangle.\delta.\langle v \rangle}{\Longrightarrow}_T s_1'$ and $s_1'' \ltimes s_1'$. Since $enabled(SES, s_1'', \alpha)$ holds, we obtain from Lemma 5.4.2 that there is a trace $\alpha' \in (E \setminus C)^*$ with $\alpha'|_V = \alpha|_V$ and $enabled(SES, s_1', \alpha')$. Consequently, $\beta.\langle c \rangle.\delta.\langle v \rangle.\alpha' \in Tr_{SES}$. □

5.4.4 Unwinding Conditions *lrbe* and *fcrbe*

It remains to introduce the unwinding conditions for $BSIA^{\rho}$ and $FCIA^{\rho,\Gamma}$. Due to the similarity between $BSIA^{\rho}$ and *BSI*, the unwinding condition $lrbe^{\rho}$ is also similar to *lrb*. The only difference between these BSPs is that *BSI* requires the insertion of every confidential event at every position in a trace (where it is not followed by other confidential events) while $BSIA^{\rho}$ requires the insertion of confidential events only at positions where they are ρ-admissible. This is reflected by the assumption $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$ in the formal definition of $BSIA^{\rho}$. Hence, an unwinding condition for $BSIA^{\rho}$ could be obtained by adding $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$ as an assumption in the unwinding condition for *BSI*. Unfortunately, this straightforward solution is not very appropriate because $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$ is defined in terms of complete traces rather than in terms of states and individual transitions.

However, it is possible to derive a more appropriate condition from $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$. To derive this condition, let us recall the definition of $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$: An event e is ρ-admissible after trace β for the view $\mathcal{V}$ (denoted by $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$) if there is a trace γ such that $\gamma.\langle e \rangle \in Tr$ and $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$ hold (also cf. Definition 3.4.18). In other words, there must be some trace γ after which e is enabled and that equals β in its occurrences of $\rho(\mathcal{V})$-events. The diagram in Figure 5.6 shows how $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$ can be adopted for state-event systems.

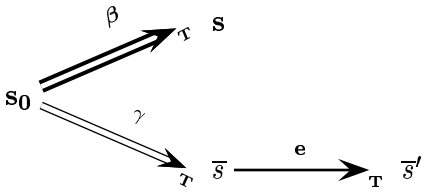


Figure 5.6: Adaptation of $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$ for state-event systems

Figure 5.6 provides a basis for reformulating $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, e)$ to a condition that is expressed in terms of states and that, hence, is more appropriate for use as an assumption in the definition of an unwinding condition. Guided by the state annotations in the above diagram, we obtain the condition $En_{\mathcal{V}}^{\rho}(T, s, e)$:

Definition 5.4.6 (ρ-enabledness). An event $e \in E$ is *ρ-enabled* in a state $s \in S$ if $En_{\mathcal{V}}^{\rho}(T, s, e)$ holds where $En_{\mathcal{V}}^{\rho}(T, s, e)$ is defined by:

$$En_{\mathcal{V}}^{\rho}(T,s,e) \equiv \exists \beta,\gamma \in E^*.\, \exists \overline{s},\overline{s}' \in S.\ (s_0 \stackrel{\beta}{\Longrightarrow}_T s) \wedge \gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})} \wedge s_0 \stackrel{\gamma}{\Longrightarrow}_T \overline{s} \wedge \overline{s} \stackrel{e}{\Longrightarrow}_T \overline{s}') \quad \diamond$$

The following theorem ensures that ρ-enabledness (for state-event systems) is equivalent to ρ-admissibility (for event systems).

Theorem 5.4.7. Let $ES_{SES} = (E, I, O, Tr_{SES})$ be the event system that is induced by *SES* and let $e \in E$ be an arbitrary event. The following propositions are valid:

1. For all states $s \in S$, if $reachable(SES, s)$ and $En_{\mathcal{V}}^{\rho}(T,s,e)$ then there is a trace $\beta \in E^*$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, e)$ hold.
2. For all traces $\beta \in E^*$, if $\beta \in Tr_{SES}$ and $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, e)$ then there is a state $s \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $En_{\mathcal{V}}^{\rho}(T,s,e)$ hold. $\diamond$

Proof. For the first proposition, assume $reachable(SES, s)$ and $En_{\mathcal{V}}^{\rho}(T,s,e)$. Since $En_{\mathcal{V}}^{\rho}(T,s,e)$, there are $\beta,\gamma \in E^*$ and $\overline{s},\overline{s}' \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s$, $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$, $s_0 \stackrel{\gamma}{\Longrightarrow}_T \overline{s}$, and $\overline{s} \stackrel{e}{\longrightarrow}_T \overline{s}'$. Thus, $s_0 \stackrel{\gamma.\langle e\rangle}{\Longrightarrow}_T \overline{s}'$. By definition of Tr_{SES}, we have $\gamma.\langle e\rangle \in Tr_{SES}$ and $\beta \in Tr_{SES}$. Since $\beta \in Tr_{SES}$, $\gamma.\langle e\rangle \in Tr_{SES}$, and $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$, we have $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, e)$.

For the second proposition, assume $\beta \in Tr_{SES}$ and $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, e)$. Since $\beta \in Tr_{SES}$ there is a state $s \in S$ for which $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ holds. $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, e)$ implies that there is a trace $\gamma \in E^*$ with $\gamma.\langle e\rangle \in Tr_{SES}$ and $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$. Since $\gamma.\langle e\rangle \in Tr_{SES}$, there are states $\overline{s},\overline{s}' \in S$ such that $s_0 \stackrel{\gamma}{\Longrightarrow}_T \overline{s}$ and $\overline{s} \stackrel{e}{\longrightarrow}_T \overline{s}'$. $s_0 \stackrel{\beta}{\Longrightarrow}_T s$, $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$, $s_0 \stackrel{\gamma}{\Longrightarrow}_T \overline{s}$, and $\overline{s} \stackrel{e}{\longrightarrow}_T \overline{s}'$ imply $En_{\mathcal{V}}^{\rho}(T,s,e)$. $\square$

By adding $En_{\mathcal{V}}^{\rho}(T,s,c)$ to the assumptions in the definitions of *lrb* and *fcrb*, we obtain unwinding conditions that are suitable for verifying $BSIA^{\rho}$ and $FCIA^{\rho,\Gamma}$.

Definition 5.4.8 (*lrbe*, *fcrbe*). Let $\ltimes \subseteq S \times S$ be a relation. The unwinding conditions $lrbe^{\rho}$ (for locally-respects backwards for enabled events) and $fcrbe^{\rho,\Gamma}$ (for forward-correctably respects backwards for enabled events) are defined as follows:

$$lrbe_{\mathcal{V}}^{\rho}(T,\ltimes) \equiv \forall s \in S.\, \forall c \in C.\ \left(\begin{array}{l} (reachable(SES,s) \wedge \underline{En_{\mathcal{V}}^{\rho}(T,s,c)}) \\ \Rightarrow \exists s' \in S.\, (s \stackrel{c}{\longrightarrow}_T s' \wedge s \ltimes s') \end{array} \right)$$

$$fcrbe_{\mathcal{V}}^{\rho,\Gamma}(T,\ltimes) \equiv \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla.\, \forall s,s'' \in S.\ \left(\begin{array}{l} (reachable(SES,s) \wedge s \stackrel{v}{\longrightarrow}_T s'' \wedge \underline{En_{\mathcal{V}}^{\rho}(T,s,c)}) \\ \Rightarrow \exists s' \in S.\, \exists \delta \in (N \cap \Delta)^*.\, (s \stackrel{\langle c\rangle.\delta.\langle v\rangle}{\Longrightarrow}_T s' \wedge s'' \ltimes s') \end{array} \right) \quad \diamond$$

The conditions $lrbe_{\mathcal{V}}^{\rho}(T,\ltimes)$ and $fcrbe_{\mathcal{V}}^{\rho,\Gamma}(T,\ltimes)$ differ from $lrb_{\mathcal{V}}(T,\ltimes)$ and $fcrb_{\mathcal{V}}^{\Gamma}(T,\ltimes)$, respectively, only in the assumption $En_{\mathcal{V}}^{\rho}(T,s,c)$. To point out this difference, we have underlined the additional assumption in Definition 5.4.8.

We now prove propositions 3 and 6 of Theorem 5.3.1.

Proof (of proposition 3 and 6 in Theorem 5.3.1).

1. Let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary with $\beta.\alpha \in Tr_{SES}$ and $\alpha|_C = \langle\rangle$. There is a state $s_1 \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s_1$ and $enabled(SES, s_1, \alpha)$. We make a case distinction depending on $En^{\rho}_{\mathcal{V}}(T, s_1, c)$.

   - Assume that $En^{\rho}_{\mathcal{V}}(T, s_1, c)$ holds. Since $lrbe^{\rho}_{\mathcal{V}}(T, \ltimes)$ and $reachable(SES, s_1)$ hold, there is a state $s_1' \in S$ with $s_1 \stackrel{c}{\longrightarrow}_T s_1'$ and $s_1 \ltimes s_1'$. Since $enabled(SES, s_1, \alpha)$ holds, we obtain from Lemma 5.4.2 that there is a trace $\alpha' \in (E \setminus C)^*$ with $\alpha'|_V = \alpha|_V$ and $enabled(SES, s_1', \alpha')$. Consequently, $\beta.\langle c\rangle.\alpha' \in Tr_{SES}$.
   - Assume that $En^{\rho}_{\mathcal{V}}(T, s_1, c)$ does not hold. From Theorem 5.4.7(2), we obtain that $Adm^{\rho}_{\mathcal{V}}(Tr_{SES}, \beta, c)$ does not hold. Hence, $BSIA^{\rho}_{\mathcal{V}}(Tr_{SES})$ is fulfilled trivially.

2. Let $\alpha, \beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary with $\beta.\langle v\rangle.\alpha \in Tr_{SES}$ and $\alpha|_C = \langle\rangle$. There are states $s_1, s_1'' \in S$ such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s_1$, $s_1 \stackrel{v}{\longrightarrow}_T s_1''$, and $enabled(SES, s_1'', \alpha)$. We make a case distinction depending on $En^{\rho}_{\mathcal{V}}(T, s_1, c)$.

   - Assume that $En^{\rho}_{\mathcal{V}}(T, s_1, c)$ holds. Since $fcrbe^{\rho,\Gamma}_{\mathcal{V}}(T, \ltimes)$, $reachable(SES, s_1)$, and $s_1 \stackrel{v}{\longrightarrow}_T s_1''$ hold, there are a state $s_1' \in S$ and a trace $\delta \in (N \cap \Delta)^*$ with $s_1 \stackrel{\langle c\rangle.\delta.\langle v\rangle}{\Longrightarrow}_T s_1'$ and $s_1'' \ltimes s_1'$. Since $enabled(SES, s_1'', \alpha)$ holds, we obtain from Lemma 5.4.2 that there is a trace $\alpha' \in (E \setminus C)^*$ such that $\alpha'|_V = \alpha|_V$ and $enabled(SES, s_1', \alpha')$ hold. Consequently, $\beta.\langle c\rangle.\delta.\langle v\rangle.\alpha' \in Tr_{SES}$.
   - Assume that $En^{\rho}_{\mathcal{V}}(T, s_1, c)$ does not hold. From Theorem 5.4.7(2) we obtain that $Adm^{\rho}_{\mathcal{V}}(Tr_{SES}, \beta, c)$ does not hold. Hence, $FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr_{SES})$ holds trivially. □

This concludes the proof of Theorem 5.3.1.

5.4.5 On Completeness

In general, backwards-strict BSPs and forward-correctable BSPs may hold although the corresponding unwinding conditions are not satisfied. Nevertheless, our unwinding conditions capture the requirements of the corresponding BSPs quite closely. Moreover, under the assumption that $N = \emptyset$ holds, our unwinding conditions also are necessary conditions for the corresponding backwards-strict BSPs and forward-correctable BSPs.

Theorem 5.4.9 (Conditional completeness).

1. If $N = \emptyset$ and $BSD_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrf_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

2. If $N = \emptyset$ and $BSI_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrb_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

3. If $N = \emptyset$ and $BSIA^{\rho}_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrbe^{\rho}_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

4. If $N = \emptyset$ and $FCD^{\Gamma}_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $fcrf^{\Gamma}_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

5. If $N = \emptyset$ and $FCI_{\mathcal{V}}^{\Gamma}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $fcrb_{\mathcal{V}}^{\Gamma}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

6. If $N = \emptyset$ and $FCIA_{\mathcal{V}}^{\rho,\Gamma}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $fcrbe_{\mathcal{V}}^{\rho,\Gamma}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold. ◇

Let us postpone the proof of Theorem 5.4.9 to a later point of this section. First, we want to present also completeness results for other BSPs. For the strict BSPs *SD*, *SI*, and *SIA*, the corresponding unwinding conditions are necessary, in general.

Theorem 5.4.10 (Completeness). Let $\mathcal{V}' = (V \cup N, \emptyset, C)$ and $\rho'(\mathcal{V}') = \rho(\mathcal{V})$.

1. If $SD_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrf_{\mathcal{V}'}(T, \ltimes)$ and $osc_{\mathcal{V}'}(T, \ltimes)$ hold.

2. If $SI_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrb_{\mathcal{V}'}(T, \ltimes)$ and $osc_{\mathcal{V}'}(T, \ltimes)$ hold.

3. If $SIA_{\mathcal{V}}^{\rho}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrbe_{\mathcal{V}'}^{\rho'}(T, \ltimes)$ and $osc_{\mathcal{V}'}(T, \ltimes)$ hold (where $\rho'(\mathcal{V}') = \rho(\mathcal{V})$). ◇

Proof. The propositions follow immediately from Theorems 5.4.9(1–3), 3.5.6, and 3.5.15. □

For the non-strict BSPs *D*, *I*, and *IA* the corresponding unwinding conditions are necessary under the assumption $N = \emptyset$.

Theorem 5.4.11 (Conditional completeness).

1. If $N = \emptyset$ and $D_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrf_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

2. If $N = \emptyset$ and $I_{\mathcal{V}}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrb_{\mathcal{V}}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold.

3. If $N = \emptyset$ and $IA_{\mathcal{V}}^{\rho}(Tr_{SES})$ then there is a relation $\ltimes \subseteq S \times S$ such that $lrbe_{\mathcal{V}}^{\rho}(T, \ltimes)$ and $osc_{\mathcal{V}}(T, \ltimes)$ hold. ◇

Proof. The propositions follow immediately from Theorems 5.4.9(1–3), 3.5.6, and 3.5.15. □

Before proving Theorem 5.4.9, let us introduce a lemma that is helpful in that proof. This lemma states conditions under which $osc_{\mathcal{V}}(T, \ltimes)$ holds.

Lemma 5.4.12. Let $\ltimes \subseteq S \times S$ be a relation. If $N = \emptyset$ and the following condition holds for all states $s_1, s_1' \in S$ that are reachable:

$$s_1' \ltimes s_1 \Leftrightarrow \forall \alpha \in (E \setminus C)^*.\, (enabled(SES, s_1', \alpha) \Rightarrow enabled(SES, s_1, \alpha))$$

then $osc_{\mathcal{V}}(T, \ltimes)$ holds. ◇

Proof. We have to prove the following formula, which results from $osc_{\mathcal{V}}(T, \ltimes)$ under the assumption $N = \emptyset$.

$$\forall s_1, s_1', s_2' \in S.\, \forall e \in E \setminus C. \qquad (5.12)$$
$$\left(\begin{array}{l} (reachable(SES, s_1) \wedge reachable(SES, s_1') \wedge s_1' \xrightarrow{e}_T s_2' \wedge s_1' \ltimes s_1) \\ \Rightarrow \exists s_2 \in S.\, (s_1 \xrightarrow{e}_T s_2 \wedge s_2' \ltimes s_2) \end{array} \right)$$

Assume $s_1, s_1', s_2' \in S$ and $e \in E \setminus C$ with $reachable(SES, s_1)$, $reachable(SES, s_1')$, $s_1' \xrightarrow{e}_T s_2'$, and $s_1' \ltimes s_1$. Let $\alpha_1 \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s_2', \alpha_1)$ holds. Therefore, $enabled(SES, s_1', \langle e \rangle.\alpha_1)$ holds and, according to our assumption about $\ltimes$, this implies $enabled(SES, s_1, \langle e \rangle.\alpha_1)$. Hence, there is a state $s_2 \in S$ with $s_1 \xrightarrow{e}_T s_2$ and $enabled(SES, s_2, \alpha_1)$. According to the construction of $\ltimes$, we obtain $s_2' \ltimes s_2$ because α_1 was chosen arbitrarily (recall that T is functional in the first two arguments). □

Now we are ready to prove Theorem 5.4.9.

Proof (of Theorem 5.4.9). We define $\ltimes$ by

$$s_1' \ltimes s_1 \Leftrightarrow \forall \alpha \in (E \setminus C)^*.\, (enabled(SES, s_1', \alpha)) \Rightarrow enabled(SES, s_1, \alpha))$$

According to Lemma 5.4.12, $osc_{\mathcal{V}}(T, \ltimes)$ holds for this definition of $\ltimes$. It remains to show, for the each BSP, that the other unwinding condition is fulfilled as well. We investigate each of the six cases separately.

1. Assume that $N = \emptyset$ and $BSD_{\mathcal{V}}(Tr_{SES})$ hold. Let $s, s' \in S$ and $c \in C$ be arbitrary such that $reachable(SES, s)$ and $s \xrightarrow{c}_T s'$ hold. Thus, there is a trace $\beta \in E^*$ with $s_0 \overset{\beta}{\Longrightarrow}_T s$. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s', \alpha)$ holds. From $BSD_{\mathcal{V}}(Tr_{SES})$ and $\beta.\langle c \rangle.\alpha \in Tr_{SES}$, we conclude that $\beta.\alpha \in Tr_{SES}$ holds. Consequently, $enabled(SES, s, \alpha)$ holds. According to our definition of $\ltimes$, this implies $s' \ltimes s$ because α was chosen arbitrarily.

2. Assume that $N = \emptyset$ and $BSI_{\mathcal{V}}(Tr_{SES})$ hold. Let $s \in S$ and $c \in C$ be arbitrary such that $reachable(SES, s)$ holds. Thus, there is a trace $\beta \in E^*$ with $s_0 \overset{\beta}{\Longrightarrow}_T s$. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s, \alpha)$ holds. From $BSI_{\mathcal{V}}(Tr_{SES})$ and $\beta.\alpha \in Tr_{SES}$, we conclude that $\beta.\langle c \rangle.\alpha \in Tr_{SES}$ holds. Consequently, there is a state $s' \in S$ with $s \xrightarrow{c}_T s'$ for which $enabled(SES, s', \alpha)$ holds. According to our definition of $\ltimes$, this implies $s \ltimes s'$.

3. Assume that $N = \emptyset$ and $BSIA^{\rho}_{\mathcal{V}}(Tr_{SES})$ hold. Let $s \in S$ and $c \in C$ be arbitrary such that $reachable(SES, s)$ holds. Thus, there is a trace $\beta \in E^*$ with $s_0 \overset{\beta}{\Longrightarrow}_T s$. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s, \alpha)$ holds. We make a case distinction:

   - Assume that $Adm^{\rho}_{\mathcal{V}}(Tr_{SES}, \beta, c)$ holds. From $BSIA^{\rho}_{\mathcal{V}}(Tr_{SES})$ and $\beta.\alpha \in Tr_{SES}$, we conclude that $\beta.\langle c \rangle.\alpha \in Tr_{SES}$ holds. Consequently, there is a state $s' \in S$ with $s \xrightarrow{c}_T s'$ for which $enabled(SES, s', \alpha)$ holds. According to our definition of $\ltimes$, this implies $s \ltimes s'$.
   - Assume that $Adm^{\rho}_{\mathcal{V}}(Tr_{SES}, \beta, c)$ does not hold. Theorem 5.4.7(1) implies that $En^{\rho}_{\mathcal{V}}(T, s, c)$ also does not hold. Hence, $lrbe^{\rho}_{\mathcal{V}}(T, \ltimes)$ is fulfilled trivially.

4. Assume that $N = \emptyset$ and $FCD_{\mathcal{V}}^{\Gamma}(Tr_{SES})$ hold. Let $s, s' \in S$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary such that $reachable(SES, s)$ and $s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s'$ hold. Thus, there is a trace $\beta \in E^*$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s', \alpha)$ holds. From $FCD_{\mathcal{V}}^{\Gamma}(Tr_{SES})$, $\beta.\langle c.v \rangle.\alpha \in Tr_{SES}$, and $N = \emptyset$, we conclude that $\beta.\langle v \rangle.\alpha \in Tr_{SES}$ holds ($\delta = \langle\rangle$ because $N = \emptyset$). Consequently, there is a state $s'' \in S$ for which $s \stackrel{v}{\longrightarrow}_T s''$ and $enabled(SES, s'', \alpha)$ holds. According to our definition of $\ltimes$, this implies $s' \ltimes s''$ because α was chosen arbitrarily and s'' is uniquely determined by $s \stackrel{v}{\longrightarrow}_T s''$.

5. Assume that $N = \emptyset$ and $FCI_{\mathcal{V}}^{\Gamma}(Tr_{SES})$ hold. Let $s, s'' \in S$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary such that $reachable(SES, s)$ and $s \stackrel{v}{\longrightarrow}_T s''$ hold. Thus, there is a trace $\beta \in E^*$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s'', \alpha)$ holds. From $FCI_{\mathcal{V}}^{\Gamma}(Tr_{SES})$, $\beta.\langle v \rangle.\alpha \in Tr_{SES}$, and $N = \emptyset$, we conclude that $\beta.\langle c.v \rangle.\alpha \in Tr_{SES}$ holds. Hence, there is a state $s' \in S$ with $s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s'$ for which $enabled(SES, s', \alpha)$ holds. According to our definition of $\ltimes$, this implies $s'' \ltimes s'$.

6. Assume that $N = \emptyset$ and $FCIA_{\mathcal{V}}^{\rho,\Gamma}(Tr_{SES})$ hold. Let $s, s'' \in S$, $c \in C \cap \Upsilon$, $v \in V \cap \nabla$ be arbitrary such that $reachable(SES, s)$ and $s \stackrel{v}{\longrightarrow}_T s''$ hold. Thus, there is a trace $\beta \in E^*$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $enabled(SES, s'', \alpha)$ holds. We make a case distinction:

   - Assume that $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, c)$ holds. From $FCIA_{\mathcal{V}}^{\rho,\Gamma}(Tr_{SES})$, $\beta.\langle v \rangle.\alpha \in Tr_{SES}$, and $N = \emptyset$, we conclude that $\beta.\langle c.v \rangle.\alpha \in Tr_{SES}$ holds. Consequently, there is a state $s' \in S$ for which $s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s'$ and $enabled(SES, s', \alpha)$ hold. According to our definition of $\ltimes$, this implies $s'' \ltimes s'$.
   - Assume that $Adm_{\mathcal{V}}^{\rho}(Tr_{SES}, \beta, c)$ does not hold. Theorem 5.4.7(1) implies that $En_{\mathcal{V}}^{\rho}(T, s'', c)$ also does not hold. Hence, $fcrbe_{\mathcal{V}}^{\rho,\Gamma}(T, \ltimes)$ is fulfilled trivially. □

Unwinding, as a proof technique, is not complete for backwards-strict BSPs, in general. This is demonstrated by the following examples. The first of these examples is concerned with the unwinding result for *BSD* and the second with the unwinding result for *BSI*.

Example 5.4.13. Let the state-event system $SES_1 = (S, s_0, E, I, O, T_1)$ be defined by

$$\begin{aligned} S &= \{s_0\} \cup \{s_j \mid 1 \leq j \leq 10\} \\ E &= \{c, n_1, n_2, v_1, v_2, v_3\} \\ T_1 &= \{(s_0, c, s_1), (s_1, v_1, s_2), (s_2, v_2, s_3), (s_2, v_3, s_4), (s_0, n_1, s_5), \\ &\quad (s_5, v_1, s_6), (s_6, v_2, s_7), (s_0, n_2, s_8), (s_8, v_1, s_9), (s_9, v_3, s_{10})\} \end{aligned}$$

and let the view $\mathcal{V} = (V, N, C)$ be defined by $V = \{v_1, v_2, v_3\}$, $N = \{n_1, n_2\}$, and $C = \{c\}$. The transition relation T_1 is also depicted on the left hand side of Figure 5.7.

In the figure, it is easy to see that SES_1 fulfills $BSD_{\mathcal{V}}(Tr_{SES_1})$. Let us now try to construct an unwinding relation $\ltimes$ for which $lrf_{\mathcal{V}}(T_1, \ltimes)$ and $osc_{\mathcal{V}}(T_1, \ltimes)$ are fulfilled. Since $s_0 \stackrel{c}{\longrightarrow}_{T_1} s_1$ holds, $s_1 \ltimes s_0$ needs to hold for the satisfaction of $lrf_{\mathcal{V}}(T_1, \ltimes)$. Since $s_1 \stackrel{v_1}{\longrightarrow}_{T_1} s_2$ and $s_1 \ltimes s_0$ hold, there must be a state $s_2' \in S$ and a sequence $\delta \in (E \setminus C)^*$ for which $\delta|_V = \langle v_1 \rangle$, $s_0 \stackrel{\delta}{\Longrightarrow}_{T_1} s_2'$, and $s_2 \ltimes s_2'$ hold in order to satisfy $osc_{\mathcal{V}}(T_1, \ltimes)$. The only two candidates for δ are $\langle n_1.v_1 \rangle$ and $\langle n_2.v_1 \rangle$ and the only two candidates for s_2' are s_6 and s_9. This means, $s_2 \ltimes s_6$

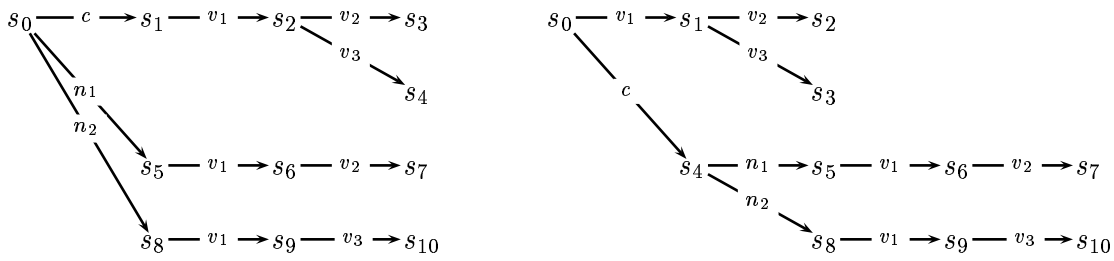


Figure 5.7: Counterexamples for completeness (cf. Examples 5.4.13 and 5.4.14)

or $s_2 \ltimes s_9$ must hold. However, while v_2 as well as v_3 are enabled in s_2, v_3 is not enabled in s_6 and v_2 is not enabled in s_9. Hence, if $s_2 \ltimes s_6$ or $s_2 \ltimes s_9$ hold then $osc_{\mathcal{V}}(T_1, \ltimes)$ is violated. Consequently, there cannot be an unwinding relation $\ltimes$ for which $lrf_{\mathcal{V}}(T_1, \ltimes)$ and $osc_{\mathcal{V}}(T_1, \ltimes)$ are fulfilled. ◇

Example 5.4.14. Let the state-event system $SES_2 = (S, s_0, E, I, O, T_2)$ be defined by

$$\begin{aligned} S &= \{s_0\} \cup \{s_j \mid 1 \leq j \leq 10\} \\ E &= \{c, n_1, n_2, v_1, v_2, v_3\} \\ T_2 &= \{(s_0, v_1, s_1), (s_1, v_2, s_2), (s_1, v_3, s_3), (s_0, c, s_4), (s_4, n_1, s_5), \\ &\quad (s_5, v_1, s_6), (s_6, v_2, s_7), (s_4, n_2, s_8), (s_8, v_1, s_9), (s_9, v_3, s_{10})\} \\ &\quad \cup \{(s_j, c, s_j) \mid 1 \leq j \leq 10\} \end{aligned}$$

and let the view $\mathcal{V} = (V, N, C)$ be defined by $V = \{v_1, v_2, v_3\}$, $N = \{n_1, n_2\}$, and $C = \{c\}$. The transition relation T_2 is also depicted on the right hand side of Figure 5.7 where the transitions (s_j, c, s_j) are omitted in the diagram for better readability.

In the diagram, it is easy to see that SES_2 fulfills $BSI_{\mathcal{V}}(Tr_{SES_2})$. Therefore, it also fulfills the weaker property $BSIA^{\rho}_{\mathcal{V}}(Tr_{SES_2})$ for all choices of the parameter ρ (cf. Theorem 3.5.9 for the ordering of these BSPs). Let us now try to construct an unwinding relation $\ltimes$ for which $lrbe^{\rho}_{\mathcal{V}}(T_2, \ltimes)$ and $osc_{\mathcal{V}}(T_2, \ltimes)$ are fulfilled. Since $s_0 \xrightarrow{c}_{T_2} s_4$ holds, $s_0 \ltimes s_4$ needs to hold for the satisfaction of $lrbe^{\rho}_{\mathcal{V}}(T_2, \ltimes)$. Since $s_0 \xrightarrow{v_1}_{T_2} s_1$ and $s_0 \ltimes s_4$ hold, there must be a sequence $\delta \in (E \setminus C)^*$ and a state s_1' for which $\delta|_V = \langle v_1 \rangle$, $s_4 \stackrel{\delta}{\Longrightarrow}_{T_2} s_1'$, and $s_1 \ltimes s_1'$ hold in order to satisfy $osc_{\mathcal{V}}(T, \ltimes)$. The only two candidates for δ are $\langle n_1.v_1 \rangle$ and $\langle n_2.v_1 \rangle$ and the only two candidates for s_1' are s_6 and s_9. This means, $s_1 \ltimes s_6$ or $s_1 \ltimes s_9$ must hold. However, while v_2 as well as v_3 are enabled in s_1, v_3 is not enabled in s_6 and v_2 is not enabled in s_9. Hence, if $s_1 \ltimes s_6$ or $s_1 \ltimes s_9$ hold then $osc_{\mathcal{V}}(T_2, \ltimes)$ is violated. Consequently, there cannot be an unwinding relation $\ltimes$ for which $lrbe^{\rho}_{\mathcal{V}}(T_2, \ltimes)$ and $osc_{\mathcal{V}}(T_2, \ltimes)$ are fulfilled. Since $lrb_{\mathcal{V}}(T_2, \ltimes)$ implies $lrbe^{\rho}_{\mathcal{V}}(T_2, \ltimes)$, there also cannot be an unwinding relation $\ltimes$ for which $lrb_{\mathcal{V}}(T_2, \ltimes)$ and $osc_{\mathcal{V}}(T_2, \ltimes)$ hold. ◇

Remark 5.4.15. The proof by unwinding fails in both examples because *osc* cannot be satisfied. This is the cause for the incompleteness of the unwinding proof technique: While *osc* implies that an unwinding relation is properly defined,[5] there are proper unwinding relations

[5] That is, if $s_1' \ltimes s_1$ holds then for every trace without confidential events that is enabled in s_1' there is a trace without confidential events that yields the same observation and that is enabled in s_1.

for which *osc* is not satisfied. This means, expressing the (global) intuition of unwinding relations by local conditions leads to the incompleteness of the proof technique. This also explains why Millen [Mil94] and Zakinthinos [Zak96] have been able to prove the completeness of their unwinding techniques, namely because they specified their unwinding relations by a global condition. The price paid for establishing completeness in this way is that a large fraction of the verification burden remains to be shown at every application of these unwinding results. In other words, the disadvantage is that more effort must be spent when verifying that a given system satisfies a given information flow property.

Nevertheless, it is possible to find (other) local conditions that result in a complete proof technique without the disadvantages of the approaches in [Mil94] and [Zak96]. In Appendix C, we will derive such local conditions by adapting well known simulation techniques. This provides the basis for a proof technique that is sound as well as complete. However, the resulting verification conditions are more complex than our unwinding conditions (as they involve relations between sets of states) and, hence, they are also more difficult to apply. ◇

5.5 Unwinding Theorems for Information Flow Properties

Our unwinding results for BSPs can now be used to simplify the verification of information flow properties based on the modular representation of those properties in *MAKS*: Using the representation theorem for a given information flow property, the task of verifying this property can be reduced to the task of verifying all BSPs from which this property is assembled. Then each of the BSPs can be proven separately with the help of our unwinding results. That is, for each BSP, one constructs an unwinding relation and proves that the unwinding conditions for that BSP are satisfied. Our unwinding results have been derived completely independently of a particular information flow property. However, specialized unwinding theorems for particular information flow properties can also be derived. In this section, we illustrate how this can be done. To this end, we derive specialized unwinding theorems for all information flow properties for which we have given a representation in *MAKS*.

5.5.1 Generalized Noninterference

Unwinding results for all variants of McCullough's generalized noninterference [McC87] can be derived from Theorems 5.3.1 and 5.3.3.

Theorem 5.5.1 (Unwinding theorem for *GNI*). If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $lrb_{\mathcal{HI}}(T, \ltimes_2)$, and $osc_{\mathcal{HI}}(T, \ltimes_2)$ hold then $GNI(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,2)), we obtain $BSD_{\mathcal{HI}}(Tr_{SES})$ and $BSI_{\mathcal{HI}}(Tr_{SES})$. From the representation theorem for *GNI* (Theorem 4.2.3), we obtain $GNI(ES_{SES})$. □

The unwinding theorem for *GNI* demands that two unwinding relations $\ltimes_1$ and $\ltimes_2$ are constructed such that instances of *lrf* and *osc* are fulfilled for $\ltimes_1$ and instances of *lrb* and *osc* are fulfilled for $\ltimes_2$. However, it is not necessary that $\ltimes_1$ and $\ltimes_2$ be *different* relations. Choosing the same relation for $\ltimes_1$ and $\ltimes_2$ has advantages, namely *osc* only needs to be proven once (rather than twice) because $osc_{\mathcal{HI}}(T, \ltimes_1)$ and $osc_{\mathcal{HI}}(T, \ltimes_2)$ are identical if $\ltimes_1 = \ltimes_2$ holds. This means, having the option of using different unwinding relations for $\ltimes_1$ and $\ltimes_2$ provides some flexibility that can, but need not, be used.

Theorem 5.5.2 (Unwinding theorem for *GNI).** If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$, and $osc_{\mathcal{HI}}(T, \ltimes_2)$ hold then $GNI^*(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,3)), we obtain $BSD_{\mathcal{HI}}(Tr_{SES})$ and $BSIA^{\rho_C}_{\mathcal{HI}}(Tr_{SES})$. From the representation of GNI^* in *MAKS* (Definition 4.2.7), we obtain $GNI^*(ES_{SES})$. □

Theorem 5.5.3 (Unwinding theorem for *IBGNI*). If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ with $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $lrb_{\mathcal{HI}}(T, \ltimes_2)$, and $osc_{\mathcal{HI}}(T, \ltimes_2)$ then $IBGNI(ES_{SES})$ holds. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.3(2,4)), we obtain $R_{\mathcal{HI}}(Tr_{SES})$ and $I_{\mathcal{HI}}(Tr_{SES})$. From the representation theorem for *IBGNI* (Theorem 4.2.5), we obtain $IBGNI(ES_{SES})$. □

Theorem 5.5.4 (Unwinding theorem for *IBGNI).** If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ with $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$, and $osc_{\mathcal{HI}}(T, \ltimes_2)$ then $IBGNI^*(ES_{SES})$ holds. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.3(3,4)), we obtain $R_{\mathcal{HI}}(Tr_{SES})$ and $IA^{\rho_C}_{\mathcal{HI}}(Tr_{SES})$. From the representation of $IBGNI^*$ in *MAKS* (Definition 4.2.8), we obtain $IBGNI^*(ES_{SES})$. □

Note that the difference between *GNI* (cf. Theorem 4.2.3 for the representation in *MAKS*) and GNI^* (cf. Definition 4.2.7 for the representation in *MAKS*) leads to a difference in the corresponding unwinding conditions. While $lrb_{\mathcal{HI}}(T, \ltimes_2)$ is an unwinding condition for *GNI*, GNI^* has the unwinding condition $lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$ instead (cf. Theorems 5.5.1 and 5.5.2). This means, unwinding GNI^*, i.e. the less restrictive information flow property, leads to weaker unwinding conditions than unwinding *GNI*. In contrast to this, the unwinding conditions for *GNI* and *IBGNI* (cf. Theorem 4.2.5 for the representation in *MAKS*) are identical (cf. Theorems 5.5.1 and 5.5.3). The reason why both information flow properties have the same unwinding conditions albeit *IBGNI* is less restrictive than *GNI* is that it is difficult to capture the noncausal corrections permitted by *IBGNI* by local verification conditions. The unwinding conditions for GNI^* and $IBGNI^*$ (cf. Definition 4.2.8 for the representation in *MAKS*) are identical for the same reasons (cf. Theorems 5.5.2 and 5.5.4). However, the difference between *IBGNI* and $IBGNI^*$ is reflected by a difference in the corresponding unwinding conditions (cf. Theorems 5.5.3 and 5.5.4).

In [Zak96], Zakinthinos also presents an unwinding theorem for *GNI*. The difference to our unwinding theorem is that the unwinding condition required by Zakinthinos's theorem is a global condition that is expressed in terms of complete traces (more specifically, the set of all traces enabled in a given state) rather than being expressed in terms of single transitions. Hence, our unwinding conditions are easier to handle during verification. Moreover, the sets of traces that occur in his unwinding condition can be regarded as the equivalence classes of an equivalence relation while we permit the use of arbitrary relations as unwinding relations.[6]

5.5.2 Forward Correctability

Using Theorem 5.3.1, unwinding results can be derived for both variants of Johnson and Thayer's forward correctability [JT88].

[6]The advantages of permitting arbitrary unwinding relations will be elaborated in detail in Section 5.6.2.

Theorem 5.5.5 (Unwinding theorem for *FC*). Let $\Gamma = \Gamma_{FC}$. If there are relations $\ltimes_1, \ltimes_2, \ltimes_3, \ltimes_4 \subseteq S \times S$ for which $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $lrb_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$, $fcrf^{\Gamma}_{\mathcal{HI}}(T, \ltimes_3)$, $osc_{\mathcal{HI}}(T, \ltimes_3)$, $fcrb^{\Gamma}_{\mathcal{HI}}(T, \ltimes_4)$, and $osc_{\mathcal{HI}}(T, \ltimes_4)$ hold then $FC(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,2,4,5)), we obtain $BSD_{\mathcal{HI}}(Tr_{SES})$, $BSI_{\mathcal{HI}}(Tr_{SES})$, $FCD^{\Gamma}_{\mathcal{HI}}(Tr_{SES})$, and $FCI^{\Gamma}_{\mathcal{HI}}(Tr_{SES})$. From the representation theorem for *FC* (Theorem 4.2.11), we obtain $FC(ES_{SES})$. □

Theorem 5.5.6 (Unwinding theorem for *FC).** Let $\Gamma = \Gamma_{FC}$. If there are relations $\ltimes_1, \ltimes_2, \ltimes_3, \ltimes_4 \subseteq S \times S$ for which $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$, $fcrf^{\Gamma}_{\mathcal{HI}}(T, \ltimes_3)$, $osc_{\mathcal{HI}}(T, \ltimes_3)$, $fcrbe^{\rho_C,\Gamma}_{\mathcal{HI}}(T, \ltimes_4)$, and $osc_{\mathcal{HI}}(T, \ltimes_4)$ hold then $FC^*(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,3,4,6)), we obtain $BSD_{\mathcal{HI}}(Tr_{SES})$, $BSIA^{\rho_C}_{\mathcal{HI}}(Tr_{SES})$, $FCD^{\Gamma}_{\mathcal{HI}}(Tr_{SES})$, and $FCIA^{\rho_C,\Gamma}_{\mathcal{HI}}(Tr_{SES})$. From the representation of FC^* in *MAKS* (Definition 4.2.13), we obtain $FC^*(ES_{SES})$. □

Note that the difference between *FC* and our novel property FC^* leads to a difference in the corresponding unwinding conditions. While $lrb_{\mathcal{HI}}(T, \ltimes_2)$ and $fcrb^{\Gamma}_{\mathcal{HI}}(T, \ltimes_4)$ are unwinding conditions for *FC*, FC^* has the less restrictive unwinding conditions $lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$ and $fcrbe^{\rho_C,\Gamma}_{\mathcal{HI}}(T, \ltimes_4)$ instead.

An unwinding theorem for *FC* have also been proposed by Millen [Mil94]. The difference to our unwinding theorem is that the unwinding relation is specified by a global condition that is formulated in terms of complete traces. Moreover, Millen's unwinding relation is an equivalence relation while we permit arbitrary unwinding relations.

5.5.3 Nondeducibility for Outputs

Based on Theorem 5.3.1, unwinding results can be derived for both variants of Guttman and Nadel's nondeducibility for outputs [GN88].

Theorem 5.5.7 (Unwinding theorem for *NDO*). If ES_{SES} is input total and there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe^{\rho_{UI}}_{\mathcal{H}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold then $NDO(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,3)), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA^{\rho_{UI}}_{\mathcal{H}}(Tr_{SES})$. From the representation theorem for *NDO* (Theorem 4.2.16), we obtain $NDO(ES_{SES})$. □

Theorem 5.5.8 (Unwinding theorem for *NDO).** If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe^{\rho_{UI}}_{\mathcal{H}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold then $NDO^*(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,3)), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA^{\rho_{UI}}_{\mathcal{H}}(Tr_{SES})$. From the representation of NDO^* in *MAKS* (Definition 4.2.18), we obtain $NDO^*(ES_{SES})$. □

Naturally, there is no difference between the unwinding conditions for *NDO* and NDO^*. Recall that we introduced NDO^* as a generalization of *NDO* that is also applicable for systems that

are not input total rather than as a property that imposes entirely different restrictions on the permitted flow of information.

To the best of our knowledge, no unwinding result for *NDO* had been published before our work. This means, using our unwinding theorems for BSPs, it has been quite simple to derive a specialized unwinding theorem for an information flow property for which no such result existed before.

Theorem 5.5.9 (Completeness of unwinding for *NDO*). If ES_{SES} is input total and $NDO(ES_{SES})$ holds then there are two relations $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe_{\mathcal{H}}^{\rho_{UI}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold. ◇

Proof. Let the assumptions of the theorem be satisfied. From the representation theorem for *NDO* (Theorem 4.2.16), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA_{\mathcal{H}}^{\rho_{UI}}(Tr_{SES})$. From our completeness theorem for unwinding BSPs (Theorem 5.4.9(1,3)), we obtain that there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ with $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe_{\mathcal{H}}^{\rho_{UI}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$. □

Theorem 5.5.10 (Completeness of unwinding for *NDO).** If $NDO(ES_{SES})$ holds then there are relations $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe_{\mathcal{H}}^{\rho_{UI}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold. ◇

Proof. Let the assumptions of the theorem be satisfied. From the representation of NDO^* in *MAKS* (Definition 4.2.18), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA_{\mathcal{H}}^{\rho_{UI}}(Tr_{SES})$. From our completeness theorem for unwinding BSPs (Theorem 5.4.9(1,3)), we obtain that there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ with $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe_{\mathcal{H}}^{\rho_{UI}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$. □

5.5.4 Noninference

Based on Theorem 5.3.3, we derive an unwinding result for O'Halloran's noninference [O'H90].

Theorem 5.5.11 (Unwinding theorem for *NF*). If there is a relation $\ltimes_1 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$ and $osc_{\mathcal{H}}(T, \ltimes_1)$ hold then $NF(ES_{SES})$. ◇

Proof. Let the assumption of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.3(4)), we obtain $R_{\mathcal{H}}(Tr_{SES})$. From the representation theorem for *NF* (Theorem 4.2.20), we obtain $NF(ES_{SES})$. □

Another unwinding theorem for *NF* is due to Zakinthinos [Zak96]. His theorem requires an unwinding condition that is global in the sense that it involves sets of traces (more specifically, the set of all traces that are enabled in a given state). Interestingly, Zakinthinos's unwinding condition for *NF* is much more similar to our definition of *BSD* than to our unwinding conditions *lrf* or *osc*.[7] The progress made by our unwinding theorem is that it requires only unwinding conditions that are local in the sense that they are expressed in terms of single transitions rather than in terms of complete traces. Hence, our unwinding conditions are easier to handle during verification.

[7] In our terminology, Zakinthinos unwinding condition can be recast as: Every trace without confidential events that is enabled after a confidential event has occurred must have been enabled already in the state before this occurrence (using the version explained in the text of Section 7.7 in [Zak96] rather than the formalization in which the subset relation is obviously required in the wrong direction). This is the same requirement as in condition (5.1) in Section 5.2, namely the condition that we took as our *starting point* for the derivation of unwinding conditions rather than the result of these derivations.

5.5.5 Generalized Noninference

We derive an unwinding result for McLean's generalized noninference [McL94a] based on Theorem 5.3.3.

Theorem 5.5.12 (Unwinding theorem for *GNF*). If there is a relation $\ltimes_1 \subseteq S \times S$ for which $lrf_{\mathcal{HI}}(T, \ltimes_1)$ and $osc_{\mathcal{HI}}(T, \ltimes_1)$ hold then $GNF(ES_{SES})$. ◇

Proof. Let the assumption of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.3(4)), we obtain $R_{\mathcal{HI}}(Tr_{SES})$. From the representation theorem for *GNF* (Theorem 4.2.22), we obtain $GNF(ES_{SES})$. □

Zakinthinos also proposed an unwinding theorem for *GNF* [Zak96]. The unwinding condition required by his theorem is, again, a global condition that involves complete traces unlike our unwinding conditions that are local conditions of individual transitions.

5.5.6 Separability

Using Theorem 5.3.1, we derive an unwinding result for McLean's separability [McL94a].

Theorem 5.5.13 (Unwinding theorem for *SEP*). If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe^{\rho_C}_{\mathcal{H}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold then $SEP(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,3)), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA^{\rho_C}_{\mathcal{H}}(Tr_{SES})$. From the representation theorem for *SEP* (Theorem 4.2.24), we obtain $SEP(ES_{SES})$. □

Note that the unwinding conditions for *SEP* are quite similar to the unwinding conditions for *NDO*. The only difference is that, for *SEP*, the parameter ρ_C is used to instantiate *lrbe* while, for *NDO*, the parameter ρ_{UI} is used instead (cf. Theorem 5.5.7). This is similar to the difference between the representations of these information flow properties in *MAKS* where ρ_C is used to instantiate *BSIA* in the representation of *SEP* while ρ_{UI} is used in the representation of *NDO* instead.

To the best of our knowledge, no unwinding result for *SEP* had been published before our work.

Theorem 5.5.14 (Completeness of unwinding for *SEP*). If $SEP(ES_{SES})$ holds then there are relations $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe^{\rho_C}_{\mathcal{H}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold. ◇

Proof. Assume $SEP(ES_{SES})$. From the representation theorem for *SEP* (Theorem 4.2.24), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA^{\rho_C}_{\mathcal{H}}(Tr_{SES})$. From our completeness theorem for unwinding BSPs (Theorem 5.4.9(1,3)), we obtain that there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ with $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe^{\rho_C}_{\mathcal{H}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$. □

5.5.7 Perfect Security Property

An unwinding result can be derived for the perfect security property [ZL97] as well.

Theorem 5.5.15 (Unwinding theorem for *PSP*). If there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe^{\rho_E}_{\mathcal{H}}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold then $PSP(ES_{SES})$. ◇

Proof. Let the assumptions of the theorem be satisfied. From our unwinding theorem for BSPs (Theorem 5.3.1(1,3)), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA_{\mathcal{H}}^{\rho_E}(Tr_{SES})$. From the representation theorem for *PSP* (Theorem 4.2.26), we obtain $PSP(ES_{SES})$. □

Zakinthinos also proposed an unwinding theorem for *PSP* together with an (incorrect) completeness result. As in his other unwinding theorems, the unwinding condition is not a local condition because it is expressed in terms of sets of complete traces rather than in terms of individual transitions. Moreover, his unwinding condition is quite restrictive because it requires that high-level events can be inserted at *every* position in a trace. This is in contrast to the definition of *PSP* that only requires that high-level events can be inserted at positions where they are ρ_E-admissible. This is why his completeness result is incorrect.[8]

Theorem 5.5.16 (Completeness of unwinding for *PSP*). If $PSP(ES_{SES})$ holds then there are relations $\ltimes_1, \ltimes_2 \subseteq S \times S$ for which $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe_{\mathcal{H}}^{\rho_E}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$ hold. ◇

Proof. Assume $PSP(ES_{SES})$. From the representation theorem for *PSP* (Theorem 4.2.26), we obtain $BSD_{\mathcal{H}}(Tr_{SES})$ and $BSIA_{\mathcal{H}}^{\rho_E}(Tr_{SES})$. From our completeness theorem for unwinding BSPs (Theorem 5.4.9(1,3)), we obtain that there are $\ltimes_1, \ltimes_2 \subseteq S \times S$ with $lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$, $lrbe_{\mathcal{H}}^{\rho_E}(T, \ltimes_2)$, and $osc_{\mathcal{H}}(T, \ltimes_2)$. □

5.6 Verifying Unwinding Conditions

We will now show how an information flow property can be verified with the help of our unwinding results.

5.6.1 Verification by Unwinding: An Example

After the unwinding conditions have been determined appropriate unwinding relations need to be constructed, and it has to be shown that the unwinding conditions hold for these relations. In this section, we illustrate these two steps. Further examples for proofs by unwinding can be found in Sections 6.7 and 7.5.

Example 5.6.1. Let the state-event system $SES = (S, s_0, E, I, O, T)$ be defined by

$$\begin{array}{rcl} S & = & \{s_k \mid 0 \leq k \leq 11\} \\ E & = & \{hi, lo_1, lo_2\} \\ I & = & \{hi\} \\ O & = & \{lo_1, lo_2\} \\ T & = & \left\{ \begin{array}{l} (s_0, lo_1, s_1), (s_0, lo_2, s_2), (s_0, hi, s_4), (s_1, lo_2, s_3), (s_2, lo_1, s_3), (s_4, lo_1, s_5), \\ (s_4, lo_2, s_6), (s_4, hi, s_8), (s_6, lo_1, s_7), (s_8, lo_2, s_{10}), (s_{10}, lo_1, s_{11}) \end{array} \right\} \end{array}$$

The transition relation T is also viewed in Figure 5.8.

We now illustrate how to prove that noninference holds for this system. According to Theorem 5.5.11, we have to prove the two unwinding conditions $lrf_{\mathcal{H}}(T, \ltimes)$ and $osc_{\mathcal{H}}(T, \ltimes)$ where the view $\mathcal{H} = (V, N, C)$ shall be defined by $V = \{lo_1, lo_2\}$, $N = \emptyset$, and $C = \{hi\}$.

[8] As a side node for the reader who is familiar with Zakinthinos's notation, a corrected version of the unwinding condition is $\forall x : H \cdot \forall q : Q \cdot \underline{q/x \neq \emptyset \Rightarrow} \pi'(q) = \pi'(q/x)$ where the underlined part s the difference to the condition in [Zak96].

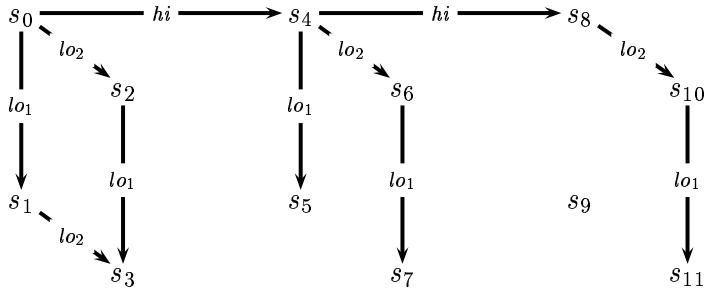


Figure 5.8: Transition relation from Example 5.6.1

Now, let us try to construct an unwinding relation $\ltimes$ for which these unwinding conditions hold. For this purpose, we start with an empty relation and iteratively add elements to this relation in order to fulfill the unwinding conditions. Firstly, in order to fulfill $lrf_{\mathcal{H}}(T, \ltimes)$, $s_0 \rtimes s_4$ and $s_4 \rtimes s_8$ need to hold (cf. Figure 5.9). Since $s_0 \rtimes s_4$ and $s_4 \xrightarrow{lo_1}_T s_5$, $osc_{\mathcal{H}}(T, \ltimes)$ requires that $s_1 \rtimes s_5$ holds. By iterating $osc_{\mathcal{H}}(T, \ltimes)$, we obtain that $s_2 \rtimes s_6$, $s_3 \rtimes s_7$, $s_6 \rtimes s_{10}$, and $s_7 \rtimes s_{11}$ need to hold as well. This results in the unwinding relation $\ltimes = \{(s_4, s_0), (s_5, s_1), (s_6, s_2), (s_7, s_3), (s_8, s_4), (s_{10}, s_6), (s_{11}, s_7)\}$, which is also viewed in Figure 5.10. With the figure, it is easy to check that $lrf_{\mathcal{H}}(T, \ltimes)$ and $osc_{\mathcal{H}}(T, \ltimes)$, indeed, hold for the unwinding relation $\ltimes$. Hence, we can conclude from our unwinding theorem for NF that $NF(ES_{SES})$ holds. $\diamond$

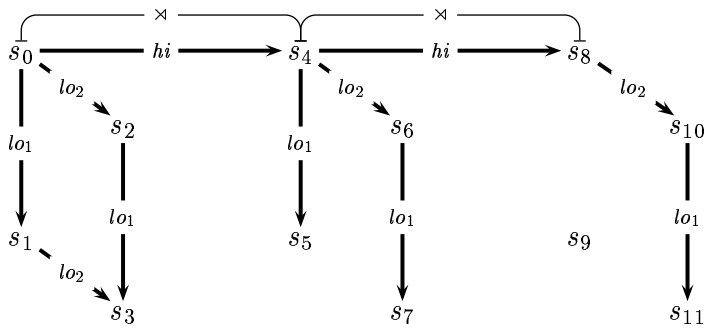


Figure 5.9: First step in construction of $\ltimes$ in Example 5.6.1

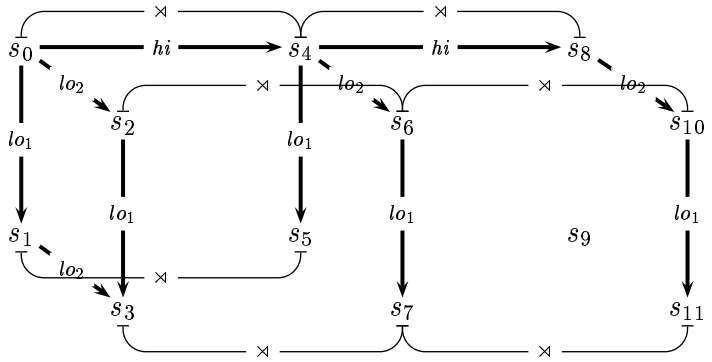


Figure 5.10: Final step in construction of $\ltimes$ in Example 5.6.1

5.6.2 Advantages of Unwinding with Arbitrary Unwinding Relations

A novelty of our approach to unwinding is that we permit arbitrary unwinding relations rather than only equivalence relations or preorders.

In a case study, we investigated a mobile device where the security requirement demanded that the current location of this device must be kept confidential [MSK+01]. We expressed this requirement by an information flow property in *MAKS* and then verified this property with the help of a previous version of our unwinding results [Man00d] that required unwinding relations to be preorders.[9] We decided that the definition of the unwinding relation should be based solely on state objects and, in particular, should not incorporate transitivity as an explicit requirement of the form $\forall s_1, s_2, s_3.\,((s_1 \ltimes s_2 \wedge s_2 \ltimes s_3) \Rightarrow s_1 \ltimes s_3)$. Our motivation for avoiding transitivity as an explicit requirement in this definition was that it would have complicated the proof of the unwinding conditions, in particular, of the condition *osc* where $\ltimes$ occurs on both sides of the implication. Since we did not require transitivity explicitly, we had to prove that our unwinding relation, indeed, was a preorder. Unfortunately, we decided to prove the unwinding conditions first, in which we succeeded, and to prove reflexivity and transitivity of the unwinding relation afterwards, in which we failed. Since the chosen relation was not transitive, all of our efforts had been wasted. In a second proof attempt (with a different unwinding relation), we succeeded to prove all conditions. However, for this it was necessary to again prove the unwinding conditions from scratch. Using the unwinding results proposed in this chapter, the initial proof attempt would have succeeded because transitivity of the unwinding relation is no longer required, i.e. the resources spent in the second proof attempt could have been saved.

However, a restriction of unwinding relations to equivalence relations is not only inconvenient in practice, but this restriction has also more fundamental disadvantages in the sense that the resulting unwinding conditions are too restrictive. For example, if unwinding relations were restricted to symmetric relations in our unwinding conditions then the distinction, e.g., between $lrf_{\mathcal{V}}(T, \ltimes)$ and $lrbe_{\mathcal{V}}^{\rho_E}(T, \ltimes)$ (or between $fcrf_{\mathcal{V}}^{\Gamma}(T, \ltimes)$ and $fcrbe_{\mathcal{V}}^{\rho_E,\Gamma}(T, \ltimes)$) would disappear, i.e. $lrf_{\mathcal{V}}(T, \ltimes)$ would hold if and only if $lrbe_{\mathcal{V}}^{\rho_E}(T, \ltimes)$ holds. This means, by using symmetric unwinding relations when unwinding noninference (unwinding conditions *lrf* and *osc*), one essentially proves the perfect security property (unwinding conditions *lrf*, $lrbe^{\rho_E}$, and *osc*). However, the perfect security property is much more restrictive than noninference. If arbitrary unwinding relations may be used then these pairs of unwinding conditions differ. In particular, this difference allows us to capture the requirements of information flow properties that are assembled from a single BSP like, e.g. noninference or generalized noninference, more closely than possible with equivalence relations.

The requirement that unwinding relations must be preorders, i.e. that they are reflexive and transitive, has been inconvenient in our case study [MSK+01]. Nevertheless, it can be shown that, at least in principle, such a restriction has no fundamental disadvantages in the sense that if there is some arbitrary unwinding relation for which a set of unwinding conditions is satisfied then there is also a preorder for which these unwinding conditions are satisfied. This result is expressed by the following theorem.

Theorem 5.6.2. Let $\mathcal{UC}$ be a set of instantiations of the unwinding conditions *lrf*, *lrb*, $lrbe^{\rho}$, $fcrf^{\Gamma}$, $fcrb^{\Gamma}$, $fcrbe^{\rho,\Gamma}$, and *osc* such that all conditions in $\mathcal{UC}$ involve the same transition relation T and the same unwinding relation $\ltimes$.

[9] A *preorder* on states is a binary relation on states that is reflexive and transitive.

If all unwinding conditions in $\mathcal{UC}$ are fulfilled for $\ltimes$ then there is a preorder $\ltimes' \subseteq S \times S$ for which all unwinding conditions in $\mathcal{UC}'$ are fulfilled where $\mathcal{UC}'$ is constructed from $\mathcal{UC}$ by replacing $\ltimes$ with $\ltimes'$ in every unwinding condition. ◇

Before proving Theorem 5.6.2, let us introduce three lemmas that are helpful in this proof.

Lemma 5.6.3. Let $\ltimes, \ltimes' \subseteq S \times S$ be binary relations between states. If $\ltimes \subseteq \ltimes'$ holds then the implications $lrf_{\mathcal{V}}(T, \ltimes) \Rightarrow lrf_{\mathcal{V}}(T, \ltimes')$, $lrb_{\mathcal{V}}(T, \ltimes) \Rightarrow lrb_{\mathcal{V}}(T, \ltimes')$, $lrbe^{\rho}_{\mathcal{V}}(T, \ltimes) \Rightarrow lrbe^{\rho}_{\mathcal{V}}(T, \ltimes')$, $fcrf^{\Gamma}_{\mathcal{V}}(T, \ltimes) \Rightarrow fcrf^{\Gamma}_{\mathcal{V}}(T, \ltimes')$, $fcrb^{\Gamma}_{\mathcal{V}}(T, \ltimes) \Rightarrow fcrb^{\Gamma}_{\mathcal{V}}(T, \ltimes')$, and $fcrbe^{\rho,\Gamma}_{\mathcal{V}}(T, \ltimes) \Rightarrow fcrbe^{\rho,\Gamma}_{\mathcal{V}}(T, \ltimes')$ hold. ◇

Proof. In the definition of the respective unwinding conditions, the unwinding relation only occurs in atomic formulas with a positive polarity (on the right hand side of the top level implication). Therefore, enlarging the unwinding relation cannot make these unwinding conditions false. □

Lemma 5.6.4. Let $\ltimes, \preceq \subseteq S \times S$ be binary relations between states. If $\preceq \subseteq S \times S$ is the restriction of $\ltimes$ to pairs of reachable states then the implications $lrf_{\mathcal{V}}(T, \ltimes) \Rightarrow lrf_{\mathcal{V}}(T, \preceq)$, $lrb_{\mathcal{V}}(T, \ltimes) \Rightarrow lrb_{\mathcal{V}}(T, \preceq)$, $lrbe^{\rho}_{\mathcal{V}}(T, \ltimes) \Rightarrow lrbe^{\rho}_{\mathcal{V}}(T, \preceq)$, $fcrf^{\Gamma}_{\mathcal{V}}(T, \ltimes) \Rightarrow fcrf^{\Gamma}_{\mathcal{V}}(T, \preceq)$, $fcrb^{\Gamma}_{\mathcal{V}}(T, \ltimes) \Rightarrow fcrb^{\Gamma}_{\mathcal{V}}(T, \preceq)$, and $fcrbe^{\rho,\Gamma}_{\mathcal{V}}(T, \ltimes) \Rightarrow fcrbe^{\rho,\Gamma}_{\mathcal{V}}(T, \preceq)$ hold. ◇

Proof. Since *lrf*, *lrb*, *lrbe*, *fcrf*, *fcrb*, and *fcrbe* only require that reachable states are related, these unwinding conditions remain valid if the unwinding relation is restricted to reachable states. □

The unwinding condition *osc* is investigated separately:

Lemma 5.6.5. Let $\ltimes, \preceq, \preceq^* \subseteq S \times S$ where $\preceq \subseteq S \times S$ is the restriction of $\ltimes$ to reachable states and $\preceq^* \subseteq S \times S$ is the reflexive and transitive closure of $\preceq$.

If $osc_{\mathcal{V}}(T, \ltimes)$ then $osc_{\mathcal{V}}(T, \preceq^*)$ holds. ◇

Proof. Assuming that $osc_{\mathcal{V}}(T, \ltimes)$ holds, we prove that for all $s_1, s_1' \in S$ the following holds

$$\begin{aligned} &\forall s_2' \in S.\, \forall \gamma' \in (E \setminus C)^*. \qquad (5.13)\\ &[reachable(SES, s_1) \wedge reachable(SES, s_1') \wedge s_1' \stackrel{\gamma'}{\Longrightarrow}_T s_2' \wedge s_1' \preceq^* s_1)\\ &\quad \Rightarrow \exists s_2 \in S.\, \exists \gamma \in (E \setminus C)^*.\, (\gamma|_V = \gamma'|_V \wedge s_1 \stackrel{\gamma}{\Longrightarrow}_T s_2 \wedge s_2' \preceq^* s_2)] \end{aligned}$$

By setting $\gamma' = \langle e \rangle$ in (5.13), one obtains $osc_{\mathcal{V}}(T, \preceq^*)$. This means that (5.13) is a generalization of $osc_{\mathcal{V}}(T, \preceq^*)$.

Since $s_1' \preceq^* s_1$ holds, there must be a finite sequence of states $\vec{u} = \langle u_1^1 \ldots u_1^n \rangle$ such that $s_1' = u_1^1$, $s_1 = u_1^n$, and $u_1^k \preceq u_1^{k+1}$ holds for all $k < n$. The induction is by the length of the shortest sequence $\vec{u}$ with the above properties. Note that $\vec{u}$ cannot be the empty sequence. The proof of (5.13) proceeds by induction on the length of $\vec{u}$.

In the *base case*, $\vec{u} = \langle u_1^1 \rangle$ holds for some state $u_1^1 \in S$. Hence, $s_1' = u_1^1 = s_1$ must hold. Since $\preceq^*$ is reflexive, (5.13) holds trivially (choose $s_2 = s_2'$ and $\gamma = \gamma'$).

In the *step case*, $\vec{u} = \langle u_1^1 . u_1^2 \rangle . \vec{u}'$ holds for some states $u_1^1, u_1^2 \in S$ and some (possible empty) sequence of states $\vec{u}' \in S^*$. Note that all states in $\vec{u}$ are reachable because of our construction of $\preceq^*$. Since $s_1' \stackrel{\gamma'}{\Longrightarrow}_T s_2'$, $s_1' = u_1^1$, $u_1^1 \preceq u_1^2$ (implies $u_1^1 \ltimes u_1^2$), and $\gamma' \in (E \setminus C)^*$ we obtain from $osc_{\mathcal{V}}(T, \ltimes)$ by an inductive argument (induction on length of γ') that there are a trace

$\delta \in (E \setminus C)^*$ and a state $u_2^2 \in S$ such that $u_1^2 \stackrel{\delta}{\Longrightarrow}_T u_2^2$, $\delta|_V = \gamma'|_V$, and $s_2' \ltimes u_2^2$ hold. Since s_2' and u_2^2 are reachable, $s_2' \preceq u_2^2$ holds. We apply the induction hypothesis on u_1^2, u_1^n ($u_1^n = s_1$), u_2^2, δ, and obtain that there are a state $s_2 \in S$ and a trace $\gamma \in (E \setminus C)^*$ with $\gamma|_V = \delta|_V$, $s_1 \stackrel{\gamma}{\Longrightarrow}_T s_2$, and $u_2^2 \preceq^* s_2$. Note that the assumptions of the induction hypothesis are all fulfilled, i.e. $\mathit{reachable}(SES, u_1^2)$, $\mathit{reachable}(SES, u_1^n)$, $u_1^2 \stackrel{\delta}{\Longrightarrow}_T u_2^2$, and $u_1^2 \preceq^* u_1^n$ hold. From $\delta|_V = \gamma'|_V$ and $\gamma|_V = \delta|_V$, we obtain $\gamma|_V = \gamma'|_V$. From $s_2' \preceq u_2^2$ and $u_2^2 \preceq^* s_2$, we obtain that $s_2' \preceq^* s_2$ holds ($\preceq^*$ is transitive). Summarizing, we have $\gamma \in (E \setminus C)^*$, $\gamma|_V = \gamma'|_V$, $s_1 \stackrel{\gamma}{\Longrightarrow}_T s_2$, and $s_2' \preceq^* s_2$. □

Using Lemmas 5.6.3, 5.6.4, and 5.6.5, we now prove Theorem 5.6.2.

Proof (of Theorem 5.6.2). Define $\ltimes'$ by $s_1 \ltimes' s_2$ iff $s_1 \preceq^* s_2$ (where $\preceq^*$ is like in Lemma 5.6.5). Hence, $\ltimes'$ is the reflexive and transitive closure of $\preceq$ where $\preceq$ is the restriction of $\ltimes$ to reachable states. From Lemmas 5.6.3 and 5.6.4, we obtain that all unwinding conditions in $\mathcal{UC}'$ involving *lrf*, *lrb*, *lrbe*, *fcrf*, *fcrb*, and *fcrbe* hold. From Lemma 5.6.5, we obtain that all unwinding conditions in $\mathcal{UC}'$ involving *osc* hold. □

Theorem 5.6.2 states that if one has proven a set of unwinding conditions for some arbitrary unwinding relation then there is also a preorder for which these unwinding conditions are satisfied. Hence, in principle, one could have used a preorder to verify the unwinding conditions. However, Theorem 5.6.2 is mainly of theoretical interest, as it may be difficult in practice to find a suitable unwinding relation that is reflexive as well as transitive and that is also appropriate for verifying the unwinding conditions. Also, it may be more difficult to prove the unwinding conditions for the preorder than for some arbitrary unwinding relation. Fortunately, our unwinding results do not impose any such restriction. This is an improvement of our earlier unwinding result [Man00d].

5.7 Summary and Comparison to Prior Frameworks

In this chapter, we have proposed novel unwinding results that simplify the verification of information flow properties. These results can be applied no matter what the particular system or information flow property is. The only prerequisites are that the system must be specified by a state-event system and that the property must be represented in *MAKS*.

Following our approach to unwinding, the task of verifying that a given system satisfies a given information flow property is reduced to the task of verifying the individual BSPs from which the property is assembled. Then, the verification of BSPs is reduced to the verification of unwinding conditions and, finally, all unwinding conditions are verified for the system under consideration. In the first of these three steps, our representation of information flow properties in *MAKS* from Chapter 4 can be exploited. The second step is completely independent of the particular information flow property and, hence, we could perform this step once and for all. For each BSP, we stated two unwinding conditions (as summarized in Table 5.1) and showed that these unwinding conditions imply the given BSP (cf. Theorems 5.3.1–5.3.3). The only part of the proof that is dependent on the particular system is the third step, i.e. the verification of the unwinding conditions. We have illustrated this step using noninference as an example (cf. Section 5.6.1). Further examples for proofs by unwinding will be presented in Sections 6.7 and 7.5.

Prior approaches to unwinding aimed at the verification of specific information flow properties (rather than being generic in the property). For example, the unwinding result in

BSP	params	view	unwinding condition 1	params	view	unwinding condition 2	view
BSD		$\mathcal{V}$	*lrf*		$\mathcal{V}$	*osc*	$\mathcal{V}$
BSI		$\mathcal{V}$	*lrb*		$\mathcal{V}$	*osc*	$\mathcal{V}$
BSIA	ρ	$\mathcal{V}$	*lrbe*	ρ	$\mathcal{V}$	*osc*	$\mathcal{V}$
FCD	Γ	$\mathcal{V}$	*fcrf*	Γ	$\mathcal{V}$	*osc*	$\mathcal{V}$
FCI	Γ	$\mathcal{V}$	*fcrb*	Γ	$\mathcal{V}$	*osc*	$\mathcal{V}$
FCIA	ρ, Γ	$\mathcal{V}$	*fcrbe*	ρ, Γ	$\mathcal{V}$	*osc*	$\mathcal{V}$
SD		$\mathcal{V}$	*lrf*		$\mathcal{V}'$	*osc*	$\mathcal{V}'$
SI		$\mathcal{V}$	*lrb*		$\mathcal{V}'$	*osc*	$\mathcal{V}'$
SIA	ρ	$\mathcal{V}$	*lrbe*	ρ'	$\mathcal{V}'$	*osc*	$\mathcal{V}'$
SR		$\mathcal{V}$	*lrf*		$\mathcal{V}'$	*osc*	$\mathcal{V}'$
D		$\mathcal{V}$	*lrf*		$\mathcal{V}$	*osc*	$\mathcal{V}$
I		$\mathcal{V}$	*lrb*		$\mathcal{V}$	*osc*	$\mathcal{V}$
IA	ρ	$\mathcal{V}$	*lrbe*	ρ	$\mathcal{V}$	*osc*	$\mathcal{V}$
R		$\mathcal{V}$	*lrf*		$\mathcal{V}$	*osc*	$\mathcal{V}$

where $\mathcal{V} = (V, N, C)$, $\mathcal{V}' = (V \cup N, \emptyset, C)$, $\rho'(\mathcal{V}') = \rho(\mathcal{V})$, and $\Gamma = (\nabla, \Delta, \Upsilon)$

Table 5.1: The unwinding conditions for each BSP

[GM84] is only applicable to the original definition of noninterference [GM82]; the result in [Mil94] aims at verifying forward correctability [JT88]; and the five unwinding results in [Zak96] aim at the verification of generalized noninterference, forward correctability, noninference, generalized noninference, and the perfect security property, respectively. In contrast, our unwinding results aim at the verification of BSPs and, hence, can be applied during the verification of every information flow property assembled from these BSPs. The advantage is that the tedious derivation of specialized unwinding results becomes unnecessary. This means, after a new information flow property has been developed (like, e.g., our novel properties *GNI**, *IBGNI**, *NDO**, and *FC**), one can immediately use this property in formal developments without having to worry about the existence of suitable unwinding techniques.

However, if desired, our unwinding theorems for BSPs can also be employed for simplifying the derivation of unwinding results for specific information flow properties. To illustrate this, we have derived unwinding theorems for all information flow properties that we have represented in *MAKS* in Chapter 4. Interestingly, we have been able to derive unwinding theorems for some properties for which no such results had been published so far, e.g. for nondeducibility for outputs, separability, and also our novel information flow properties. For other properties, our specialized unwinding theorems improve previous results. For example, our unwinding theorem for forward correctability is an improvement of a result by Millen [Mil94] in the sense that our unwinding conditions involve only single transitions while Millen specified the unwinding relation in terms of complete traces. In the same sense, our unwinding theorems for generalized noninterference, noninference, generalized noninference, and the perfect security property constitute improvements of earlier unwinding results by Zakinthinos [Zak96]. During our investigations, we have also detected a flaw in [Zak96] where it has been claimed that a given unwinding condition would be a necessary condition for the perfect

security property, which is not the case (cf. Section 5.5.7). We have corrected this flaw by presenting unwinding conditions that are, both, sufficient and necessary. All specialized unwinding results for information flow properties that we have derived in Theorems 5.5.1–5.5.8, 5.5.11–5.5.13, and 5.5.15 are listed in Table 5.2.[10]

	unwinding conditions for BSPs from 1st dimension	unwinding conditions for BSPs from 2nd dimension
$GNI(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$	$lrb_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$
$IBGNI(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$	$lrb_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$
$GNI^*(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$	$lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$
$IBGNI^*(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$	$lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$
$FC(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$ $fcrf^{\Gamma}_{\mathcal{HI}}(T, \ltimes_3)$, $osc_{\mathcal{HI}}(T, \ltimes_3)$	$lrb_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$ $fcrb^{\Gamma}_{\mathcal{HI}}(T, \ltimes_4)$, $osc_{\mathcal{HI}}(T, \ltimes_4)$
$FC^*(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$ $fcrf^{\Gamma}_{\mathcal{HI}}(T, \ltimes_3)$, $osc_{\mathcal{HI}}(T, \ltimes_3)$	$lrbe^{\rho_C}_{\mathcal{HI}}(T, \ltimes_2)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$ $fcrbe^{\rho_C,\Gamma}_{\mathcal{HI}}(T, \ltimes_4)$, $osc_{\mathcal{HI}}(T, \ltimes_4)$
$NDO(ES)$	$lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$	$lrbe^{\rho_{UI}}_{\mathcal{H}}(T, \ltimes_2)$, $osc_{\mathcal{H}}(T, \ltimes_2)$
$NDO^*(ES)$	$lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$	$lrbe^{\rho_{UI}}_{\mathcal{H}}(T, \ltimes_2)$, $osc_{\mathcal{H}}(T, \ltimes_2)$
$NF(ES)$	$lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$	
$GNF(ES)$	$lrf_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$	
$SEP(ES)$	$lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$	$lrbe^{\rho_C}_{\mathcal{H}}(T, \ltimes_2)$, $osc_{\mathcal{H}}(T, \ltimes_2)$
$PSP(ES)$	$lrf_{\mathcal{H}}(T, \ltimes_1)$, $osc_{\mathcal{H}}(T, \ltimes_1)$	$lrbe^{\rho_E}_{\mathcal{H}}(T, \ltimes_2)$, $osc_{\mathcal{H}}(T, \ltimes_2)$

where $\ltimes_1, \ltimes_2, \ltimes_3, \ltimes_4 \subseteq S \times S$ and $\Gamma = \Gamma_{FC} = (I, \emptyset, I)$

Table 5.2: Unwinding conditions for known information flow properties

Another novelty of our approach to unwinding is that we permit arbitrary unwinding relations. Other authors have required that unwinding relations must be equivalence relations (e.g. in [HY87, Rya91, GCS91, MC92, Rus92, Mil94, Zak96, RS99])[11] and our previous approach [Man00d] required that they must be preorders. The advantage of using arbitrary unwinding relations is that some information flow properties can be captured more closely. A prominent example for this is noninference because proving the unwinding conditions of noninference with an equivalence relation would amount to proving the perfect security property, i.e. a property that is much more restrictive than noninference. Another information flow property where a restriction to equivalence relations has similar disadvantages is generalized noninference. However, even if a suitable equivalence relation exists, it is sometimes difficult to find this relation, in practice (cf. Section 5.6.2). Hence, our innovation of using arbitrary

[10] Some of these theorems involve multiple unwinding relations. It is not necessary that all of these relations are defined differently. If two unwinding relations are identical then the corresponding instances of *osc* are also identical. That is, this condition needs to be proven only once.

[11] To be precise some of these approaches are based on equivalence classes rather than on equivalence relations (e.g. in [Zak96]). However, logically this is equivalent.

unwinding relations has advantages in theory as well as in practice.

The framework by Zakinthinos and Lee is the only one among the prior frameworks in which unwinding results have been derived. In his thesis [Zak96], Zakinthinos presents unwinding theorems for five information flow properties (cf. Table 5.3 for the detailed list), and he aimed at deriving each of them along the same lines. Nevertheless, he had to prove each of these theorems from scratch. In particular, there are no generic results that could be used in the proofs of the unwinding theorems for different properties. That is, the uniformity achieved is much more limited than in our work. Moreover, as pointed out before, Zakinthinos's unwinding conditions are global conditions on sets of traces and, hence, they are more difficult to handle during verification than our local unwinding conditions.

	unwinding theorem derived in framework by McLean	unwinding theorem derived in framework by Focardi and Gorrieri	unwinding theorem derived in framework by Zakinthinos and Lee	unwinding theorem derived in *MAKS*
	properties protecting occurrences and nonoccurrences of all high-level events			
SEP				√
NDO				√
*NDO**				√
PSP			√	√
	properties protecting occurrences of all high-level events			
NF			√	√
	properties protecting occurrences and nonoccurrences of high-level inputs			
FC			√	√
*FC**				√
GNI			√	√
*GNI**				√
IBGNI				√
*IBGNI**				√
	properties protecting occurrences of high-level inputs			
GNF			√	√

Table 5.3: Unwinding theorems derived in prior frameworks

The approach to verifying information flow properties proposed in this chapter exploits the uniform representation of these properties in *MAKS*. Rather than developing unwinding results for complex information flow properties, we have presented unwinding results for conceptually quite simple BSPs; rather than developing unwinding results for each property from scratch, we have presented unwinding results for basic ingredients of such properties. Hence, after representing an information flow property in *MAKS*, one obtains unwinding techniques for this property without spending any further effort. This is a very appealing feature of *MAKS* that distinguishes our framework from the ones proposed previously.

Chapter 6

Compositionality Results for Information Flow Properties

6.1 Introduction

A modular system architecture is usually chosen for the construction of large and complex systems. Thereby, a complex task, namely the development of the overall system, is reduced to a simpler development task, namely the development of the individual system components. A modular architecture can also be exploited to simplify the verification of information flow properties. Rather than verifying directly that the overall system satisfies a given property, one first verifies the property for each component and then applies a compositionality result to conclude that the overall system also satisfies this property. The existence of a suitable compositionality result is a prerequisite for applying this divide-and-conquer approach. Hence, it is unfortunate that the derivation of compositionality results is often nontrivial.

In this chapter, our main goal is to simplify the verification of compositionality results for information flow properties. For this purpose, we propose a uniform approach for deriving such results. The proposed approach is general enough for re-justifying several compositionality results from the literature as well as for deriving novel ones. It is also uniform enough for revealing similarities between compositionality results that were previously unrelated. This led to a deeper understanding of these results, which inspired the derivation of weakened forward correctability, a novel composable information flow property.

More specifically, we derive compositionality results for BSPs. These results can be applied during the derivation of compositionality theorems for more complex information flow properties according to the following observation: If all BSPs from which an information flow property is assembled are preserved under a given form of composition then the information flow property is also preserved under this form of composition. That is, deriving a compositionality theorem for a given information flow property amounts to collecting all side conditions of the compositionality results for the BSPs from which the property is assembled. Accordingly, verifying a given compositionality theorem amounts to checking that all of these side conditions are fulfilled. As we will show, this uniform approach simplifies the derivation as well as the verification of compositionality theorems considerably. This simplification does not come at the cost of the quality of the obtained results, as we will demonstrate by re-proving several results from the literature and by deriving novel compositionality results for GNI^*, FC^*, and NDO^*. Our uniform justifications of these results show that some hold

for similar reasons, which will lead to a classification of compositionality theorems.

While re-justifying the compositionality theorem for Johnson and Thayer's forward correctability [JT88], we observe that forward correctability is more restrictive than necessary for obtaining a general compositionality theorem. Based on this observation, we derive a novel information flow property, *weakened forward correctability*, that is composable although it is less restrictive than forward correctability. That is, weakened forward correctability is an improvement of forward correctability. From a historical perspective, this is a very interesting result because it constitutes after more than fifteen years a new improvement in one of the main traditions of information flow properties (incorporating forward correctability, restrictiveness, generalized noninterference, and nondeducibility).

Interestingly, we also derive our compositionality results for BSPs, which provide the basis of the above approach, in a uniform way. This uniformity is achieved by a lemma, the *generalized zipping lemma*, that is applied in the derivation of each of these compositionality results. The name of this lemma is motivated by the fact that in its proof, a possible trace of the composed system is constructed by composing traces of the components in a stepwise manner that resembles closing a zipper.

Overview. In Section 6.2, we present some basics about the composition of event-based systems. In Section 6.3, we illustrate the problem of preserving information flow properties under composition with an example. In Section 6.4, we present compositionality theorems for BSPs, introduce the generalized zipping lemma, and prove the theorems with the help of this lemma. In Section 6.5, we explain our approach to verifying compositionality theorems for more complex information flow properties and illustrate this approach with several examples. Based on these results, we present a classification of compositionality theorems in Section 6.6. In Section 6.7, we propose weakened forward correctability, a novel information flow property, and derive a compositionality theorem as well as an unwinding theorem for this property. In Section 6.8, the main results of this chapter are summarized and are put into perspective with results obtained with other frameworks.

6.2 Composition of Event Systems

In event-based system models, communication between two components is modeled by the occurrence of shared events. We employ the usual notion of composition for event systems (e.g. [JT88, ZL97]), i.e. a given communication event can occur only if it is enabled in both components and, when the event occurs, the behavior of the components is synchronized by this occurrence. When composing event systems, communication events have to be output events of one component and input events of the other component. For the composed system, communication between system components is assumed to occur internally (cf. Figure 6.1). The following definitions specify more precisely what it means to compose two event systems.

Definition 6.2.1 (Composable). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be event systems. ES_1 and ES_2 are *composable* iff $E_1 \cap E_2 \subseteq (O_1 \cap I_2) \cup (O_2 \cap I_1)$ holds. $\diamond$

Definition 6.2.2 (Composition of event systems). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be composable event systems. The *composition* of ES_1 and ES_2 is the event system $ES = (E, I, O, Tr)$ (denoted by $ES_1 \parallel ES_2$) where E, I, O, and Tr are

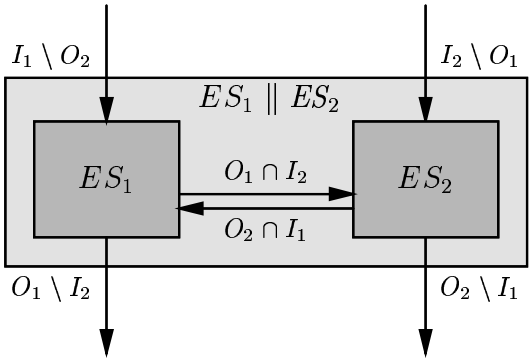


Figure 6.1: Composition of two event systems

defined as follows:

$$\begin{aligned} E &= E_1 \cup E_2 \\ I &= (I_1 \setminus O_2) \cup (I_2 \setminus O_1) \\ O &= (O_1 \setminus I_2) \cup (O_2 \setminus I_1) \\ Tr &= \{\tau \in E^* \mid \tau|_{E_1} \in Tr_1 \wedge \tau|_{E_2} \in Tr_2\} \end{aligned}$$ ◇

The operator $\parallel$ is associative as well as commutative.

Theorem 6.2.3 (Associativity). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$, $ES_2 = (E_2, I_2, O_2, Tr_2)$, and $ES_3 = (E_3, I_3, O_3, Tr_3)$ be event systems that are pairwise composable.
Then $(ES_1 \parallel ES_2) \parallel ES_3 = ES_1 \parallel (ES_2 \parallel ES_3)$ holds. ◇

Proof. Follows immediately from Definition 6.2.2. □

Theorem 6.2.4 (Commutativity). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be composable event systems. Then $ES_1 \parallel ES_2 = ES_2 \parallel ES_1$ holds. ◇

Proof. Follows immediately from Definition 6.2.2. □

Definition 6.2.2 introduces the composition of event systems in its general form. Namely, both components may communicate with each other and with the environment. In particular, feedback between components is possible, i.e. they may communicate with each other in both directions. The components have to synchronize on occurrences of events in $E_1 \cap E_2$. These events are called *communication events*. Besides general composition, we distinguish three restricted forms of composition in this chapter (as suggested in [McL94a]): product, proper cascade, and general cascade. These are formally defined in the following and are also illustrated in Figure 6.2.

Definition 6.2.5 (Product). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be composable event systems. The composition of ES_1 and ES_2 is a *product* if ES_1 and ES_2 do not communicate with each other, i.e. if $E_1 \cap E_2 = \emptyset$ holds. ◇

Definition 6.2.6 (Proper cascade). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be composable event systems. The composition of ES_1 and ES_2 is a *proper cascade* if all output events of ES_1 and all input events of ES_2 are communication events of the composed system and, moreover, neither input events of ES_1 nor output events of ES_2 are communication events, i.e. if $O_1 = E_1 \cap E_2 = I_2$ holds. ◇

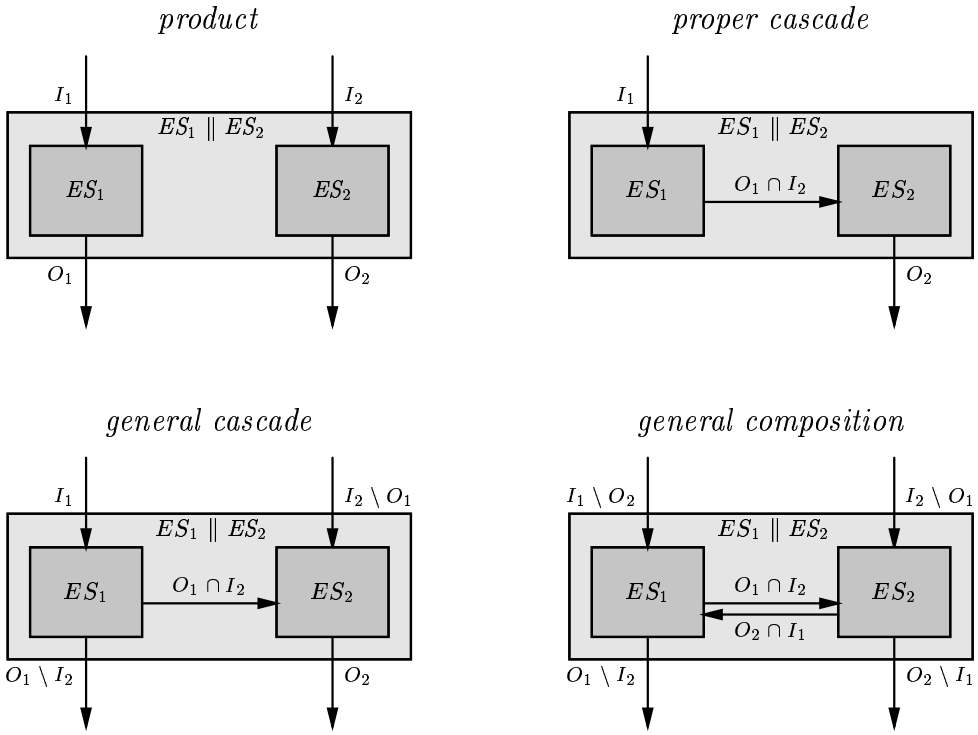


Figure 6.2: Different forms of composition

Definition 6.2.7 (General cascade). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be composable event systems. The composition of ES_1 and ES_2 is a *general cascade* if neither input events of ES_1 nor output events of ES_2 are communication events, i.e. if $E_1 \cap E_2 \subseteq O_1 \cap I_2$ holds. ◇

Theorem 6.2.8. Every product is also a general cascade and every proper cascade is also a general cascade. ◇

Proof. Follows immediately from Definitions 6.2.5–6.2.7. □

Here, composition is defined as the *binary* operation $\parallel$ on event systems. The definition of an n-ary composition operator is straightforward.

Definition 6.2.9 (n-ary Composition of event systems). Let $J = \{j \in \mathbb{N} \mid j \leq n\}$ and, for each $j \in J$, let $ES_j = (E_j, I_j, O_j, Tr_j)$ be an event system. Assume that no event belongs to more than two components and that events that are shared by two components are input events of the one component and output events of the other.

The *n-ary composition* of $(ES_j)_{j \in J}$ is the event system $ES = (E, I, O, Tr)$ (denoted by $\|_{j \in J} ES_j$) where E, I, O, and Tr are defined as follows:

$$\begin{array}{lcl} E & = & \bigcup_{j \in J} E_j \\ I & = & \bigcup_{j \in J} I_j \setminus \bigcup_{j \in J} O_j \\ O & = & \bigcup_{j \in J} O_j \setminus \bigcup_{j \in J} I_j \\ Tr & = & \{\tau \in E^* \mid \forall j \in J. \tau|_{E_j} \in Tr_j\} \end{array}$$

◇

6.3 Towards Compositionality Results

In this section, we motivate the need for compositionality results by an example that illustrates why information flow properties are not preserved under composition, in general. Based on this example, we introduce conditions under which the investigated property is preserved.

Example 6.3.1. Let the event system $ES_A = (E_A, I_A, O_A, Tr_A)$ model a system that accepts natural numbers as (confidential) input from the high-level environment (events h_n), that outputs natural numbers to the high-level environment (events h'_n), and that also outputs natural numbers to the low-level environment (events l'_n). This means, $I_A = \{h_n \mid n \in \mathbb{N}\}$, $O_A = \{h'_n, l'_n \mid n \in \mathbb{N}\}$, and $E_A = I_A \cup O_A$. In the possible traces of ES_A, occurrences of high-level events and low-level events alternate. More formally, $Tr_A \subseteq {E_A}^*$ is defined as the smallest set satisfying:

1. $\langle\rangle \in Tr_A$,
2. if $n \in \mathbb{N}$ then $\langle h_n \rangle \in Tr_A$ and $\langle h'_n \rangle \in Tr_A$, and
3. if $n \in \mathbb{N}$ and $\tau \in Tr_A$ then $\langle h_n.l'_n \rangle.\tau \in Tr_A$ and $\langle h'_n.l'_n \rangle.\tau \in Tr_A$.

This means, ES_A records its high-level input (modeled by events of the form h_n) in its low-level output where the recording is polluted by numbers that the system has generated at random (modeled by events of the form h'_n). The purpose of these random numbers is to prevent a low-level observer from deducing that certain high-level input events have occurred. Namely, if the observer receives a number n then he does not know whether this number corresponds to a (confidential) high-level input or to a (non-confidential) random number. In other words, it is impossible to tell that a particular high-level input has occurred because there is always a possible trace without this event that could have generated a given observation.[1] Obviously, $BSD_{\mathcal{V}_A}(Tr_A)$ holds for the view $\mathcal{V}_A = (V_A, N_A, C_A)$ defined by $V_A = \{l'_n \mid n \in \mathbb{N}\}$, $N_A = \{h'_n \mid n \in \mathbb{N}\}$, and $C_A = \{h_n \mid n \in \mathbb{N}\}$ because a perturbation that results from a possible trace by deleting the last occurrence of a confidential event h_n can be corrected to a possible trace by inserting an occurrence of h'_n at the position where h_n has been deleted.

Let the event system $ES_B = (E_B, I_B, O_B, Tr_B)$ model a system that accepts natural numbers as input from the high-level environment (events h'_n), that accepts natural numbers as input from the low-level environment (events l'_n), and that outputs natural numbers to the low-level environment (events l''_n). This means, $I_B = \{h'_n, l'_n \mid n \in \mathbb{N}\}$, $O_B = \{l''_n \mid n \in \mathbb{N}\}$, and $E_B = I_B \cup O_B$. System ES_B records low-level input that it receives in its low-level output. However, the low-level output need not be a complete recording of the low-level input because the system may nondeterministically swallow low-level input. Moreover, if a high-level input is received by the system then the next low-level input is not forwarded. More formally, $Tr_B \subseteq {E_B}^*$ is defined as the smallest set satisfying:

1. $\langle\rangle \in Tr_B$,
2. if $n \in N$ then $\langle h'_n \mid n \in \mathbb{N} \rangle \in Tr_B$,

[1] ES_A is very similar to the system $\mathtt{LEAK}_D$ in Example 3.4.6, i.e. the system that we introduced to illustrate the BSP D. The only essential difference between the two systems (except for differences in the names of events) is that $\mathtt{LEAK}_D$ does not record in its high-level outputs which numbers it has invented.

3. if $n \in \mathbb{N}$ and $\tau \in Tr_B$ then $\langle l'_n \rangle.\tau \in Tr_B$ and $\langle l'_n.l''_n \rangle.\tau \in Tr_B$, and

4. if $n, n' \in \mathbb{N}$ and $\tau \in Tr_B$ then $\langle h'_n.l'_{n'} \rangle.\tau \in Tr_B$.

Obviously, $BSD_{\mathcal{V}_B}(Tr_B)$ holds for the view $\mathcal{V}_B = (V_B, N_B, C_B)$ defined by $V_B = \{l'_n, l''_n \mid n \in \mathbb{N}\}$, $N_B = \emptyset$, and $C_B = \{h'_n \mid n \in \mathbb{N}\}$ because every trace that results from a possible trace by deleting the last occurrence of a confidential event h'_n is also a possible trace of the system.

Let $ES = (E, I, O, Tr)$ be defined by $ES_A \parallel ES_B$ (cf. Figure 6.3) and let the view $\mathcal{V} = (V, N, C)$ be defined by $V = V_A \cup V_B$, $N = N_A \cup N_B$, and $C = (C_A \setminus N_B) \cup (C_B \setminus N_A)$. This means that $V = \{l'_n, l''_n \mid n \in \mathbb{N}\}$, $N = \{h'_n \mid n \in \mathbb{N}\}$, and $C = \{h_n \mid n \in \mathbb{N}\}$ hold.

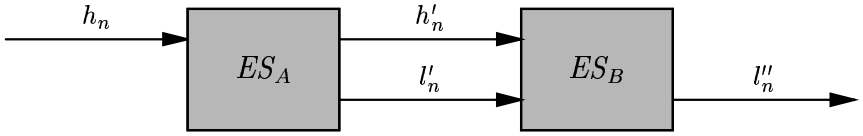


Figure 6.3: Composition of ES_A and ES_B

We show that $BSD_{\mathcal{V}}(Tr)$ does not hold by the following counterexample. The trace $\tau = \langle h_{42}.l'_{42}.l''_{42} \rangle$ is a possible trace of ES because $\tau|_{E_A} = \langle h_{42}.l'_{42} \rangle$ is a possible trace of ES_A and $\tau|_{E_B} = \langle l'_{42}.l''_{42} \rangle$ is a possible trace of ES_B. $BSD_{\mathcal{V}}(Tr)$ requires that it must be possible to correct the perturbation $t = \langle l'_{42}.l''_{42} \rangle$ that results by deleting h_{42} in τ. Let us now try to construct a correction for this perturbation: The perturbation only has an effect on the first component (because $h_{42} \notin E_B$). Namely, the perturbation $t|_{E_A} = \langle l'_{42} \rangle$ results from $\tau|_{E_A} = \langle h_{42}.l'_{42} \rangle$ by deleting h_{42}. Since $BSD_{\mathcal{V}_A}(Tr_A)$ holds, it must be possible to correct this perturbation to a possible trace of ES_A. According to the system specification, h'_{42} occurs before l'_{42} in all of these corrections and, e.g., $\tau'_1 = \langle h'_{42}.l'_{42} \rangle$ is one of these corrections. For the overall system, this means that correcting the perturbation for the first component results in the trace $t' = \langle h'_{42}.l'_{42}.l''_{42} \rangle$. For the second component, the correction for the first component leads to the trace $t'|_{E_B} = \langle h'_{42}.l'_{42}.l''_{42} \rangle$, which results from $\tau|_{E_B} = \langle l'_{42}.l''_{42} \rangle$ by inserting h'_{42}. Unfortunately, $t'|_{E_B}$ is neither a possible trace of ES_B nor can it be corrected to a possible trace (because $N_B = \emptyset$ holds). The same problem occurs with all other corrections of $t|_{E_A}$. For the overall system, this means that the perturbation t of the possible trace τ is neither a possible trace nor can it be corrected to a possible trace. Hence, $BSD_{\mathcal{V}}(Tr)$ does not hold. $\diamond$

Before we discuss in more detail the reasons why *BSD* is not preserved under composition in the above example, let us introduce the notion of a proper separation of a view.

When reducing the verification of *BSD* for a complex system $ES = (E, I, O, Tr)$ that is composed from two event systems $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ to the verification of *BSD* for each of these components, the view $\mathcal{V} = (V, N, C)$ for the overall system has to be decomposed in a suitable manner into views $\mathcal{V}_1 = (V_1, N_1, C_1)$ and $\mathcal{V}_2 = (V_2, N_2, C_2)$ for ES_1 and ES_2, respectively. Obviously, events that are confidential for the overall system must also be considered confidential when verifying *BSD* for a component (i.e. $C \cap E_1 \subseteq C_1$ and $C \cap E_2 \subseteq C_2$ must hold). Similarly, events that are visible wrt. the overall system must also be considered visible when analyzing the components (i.e. $V \cap E_1 = V_1$ and $V \cap E_2 = V_2$ must hold). Moreover, it must be ensured that changes made when correcting perturbations in one component do not reveal any secrets to the other component. This means, events that are in N_1 should be considered as confidential by ES_2 and events in N_2 should be considered

as confidential by ES_1 (i.e. $N_1 \cap N_2 = \emptyset$ must hold). Otherwise, corrections in one component would need to be synchronized with corrections in the other component, a severe restriction for compositional information flow analysis. This results in the following definition of a *proper separation* of a view.

Definition 6.3.2 (Proper separation of a view). Let E, E_1, and E_2 be sets of events such that $E = E_1 \cup E_2$ holds. Let $\mathcal{V} = (V, N, C)$, $\mathcal{V}_1 = (V_1, N_1, C_1)$, and $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E, E_1, and E_2, respectively. The views $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a *proper separation* of $\mathcal{V}$ iff $V \cap E_1 = V_1$, $V \cap E_2 = V_2$, $C \cap E_1 \subseteq C_1$, $C \cap E_2 \subseteq C_2$, and $N_1 \cap N_2 = \emptyset$ hold. $\diamond$

Example 6.3.3. The views $\mathcal{H}_1 = (L_1, \emptyset, H_1)$ and $\mathcal{H}_2 = (L_2, \emptyset, H_2)$ constitute a proper separation of $\mathcal{H} = (L, \emptyset, H)$ because $L \cap E_1 = L_1$, $L \cap E_2 = L_2$, $H \cap E_1 = H_1$, and $H \cap E_2 = H_2$. $\diamond$

Example 6.3.4. The views $\mathcal{HI}_1 = (L_1, H_1 \setminus I_1, H_1 \cap I_1)$ and $\mathcal{HI}_2 = (L_2, H_2 \setminus I_2, H_2 \cap I_2)$ constitute a proper separation of $\mathcal{HI} = (L, H \setminus I, H \cap I)$ because $L \cap E_1 = L_1$, $L \cap E_2 = L_2$, $H \cap I \cap E_1 = H_1 \cap I \cap E_1 \subseteq H_1 \cap I_1$, $H \cap I \cap E_2 = H_2 \cap I \cap E_2 \subseteq H_2 \cap I_2$, and $(H_1 \setminus I_1) \cap (H_2 \setminus I_2) \subseteq (E_1 \setminus I_1) \cap (E_2 \setminus I_2) \subseteq (E_1 \cap E_2) \setminus (I_1 \cup I_2) = \emptyset$ (recall that $E_1 \cap E_2 \subseteq (I_1 \cap O_2) \cup (I_2 \cap O_1)$ holds). $\diamond$

In the following, let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be two composable event systems. Define $ES = (E, I, O, Tr)$ by $ES = ES_1 \parallel ES_2$. Moreover, let $\mathcal{V} = (V, N, C)$, $\mathcal{V}_1 = (V_1, N_1, C_1)$, and $\mathcal{V}_2 = (V_2, N_2, C_2)$ be views in E, E_1, and E_2, respectively, such that $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a proper separation of $\mathcal{V}$.

In Example 6.3.1, we have shown that *BSD* is not preserved under composition, in general. More specifically, the preservation of *BSD* under composition is not even ensured if composition is restricted to proper cascade. However, there are many special cases where *BSD* is preserved under composition. The reason why *BSD* is not preserved in the given example is that corrections in one component cause a perturbation for the second component (because $N_A \cap E_B \neq \emptyset$) and that the second component cannot correct these perturbations.[2] These two conditions are indeed the reason why *BSD* is not preserved under composition in the example. In the following, we generalize this observation.

For simplicity let us assume for the beginning that corrections in the second component have no effect on the first component (like in the example), i.e. $N_2 \cap E_1 = \emptyset$ should hold. Note that, these assumptions do not imply that *BSD* is preserved under composition because corrections in the first component may cause perturbations in the second component and it might not be possible to correct these perturbations for the second component (as we demonstrated in Example 6.3.1). However, if

- $N_1 \cap E_2 = \emptyset$ or
- $BSIA^{\rho_2}_{\mathcal{V}_2}(Tr_2)$ and $total(ES_2, C_2 \cap N_1)$

hold then *BSD* is preserved under composition. That is, if the first of these conditions holds then corrections in ES_1 cannot have any effects on ES_2. Alternatively, if the second condition holds then the perturbations caused by ES_1 can be corrected for ES_2.

The case where $N_2 \cap E_1 = \emptyset$ does not hold but where $N_1 \cap E_2 = \emptyset$ holds instead can be handled similarly because it is symmetric. This means, if $N_1 \cap E_2 = \emptyset$, $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr_1)$, and $total(ES_1, C_1 \cap N_2)$ hold then *BSD* is preserved under composition.

[2] That $BSD_{\mathcal{V}_B}(Tr_B)$ holds does not help in the example because it is needed to correct the *insertion* of confidential events rather than their *deletion*.

However, if neither $N_1 \cap E_2 = \emptyset$ nor $N_2 \cap E_1 = \emptyset$ holds then the situation becomes more complicated. This is because corrections in the first component may cause perturbations for the second component (like in our example composition of ES_A and ES_B) and corrections in the second component may also cause perturbations for the first component (unlike in the example). Hence, one must not only ensure that the second component can deal with perturbations caused by corrections in the first component (e.g. by demanding that $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$ and $total(ES_2, C_2 \cap N_1)$ hold for some function ρ_2) but also that the first component can deal with perturbations caused by corrections in the second component (e.g. by demanding that $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ and $total(ES_1, C_1 \cap N_2)$ hold for some function ρ_1). However, this is still not sufficient because correcting a perturbation for the first component might lead to perturbations for the second component, whose corrections might lead to perturbations for the first component, and so on. This means, one must ensure that the process of correcting the initial perturbation terminates. This is what our forward correctable BSPs are good for: Instead of only demanding that instances of $BSIA^{\rho}$ hold for each component, we also require that instances of $FCIA^{\rho,\Gamma}$ hold. Intuitively, this additional requirement ensures that there is enough distance between the point in a trace where the perturbation has caused a change and the points in the trace where changes have been made in the process of correcting the perturbation.[3] More specifically, we require that there is a function ρ_1 from views in E_1 to subsets of E_1, a function ρ_2 from views in E_2 to subsets of E_2, and triples $\Gamma_1 = (\nabla_1, \Delta_1, \Upsilon_1)$ and $\Gamma_2 = (\nabla_2, \Delta_2, \Upsilon_2)$ (where $\nabla_1, \Delta_1, \Upsilon_1 \subseteq E_1$ and $\nabla_2, \Delta_2, \Upsilon_2 \subseteq E_2$) such that

- $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$, $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$, $total(ES_1, C_1 \cap N_2)$, $total(ES_2, C_2 \cap N_1)$,
- $FCIA_{\mathcal{V}_1}^{\rho_1,\Gamma_1}(Tr_1)$, $FCIA_{\mathcal{V}_2}^{\rho_2,\Gamma_2}(Tr_2)$,
- $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$,
- $C_1 \cap N_2 \subseteq \Upsilon_1$, $C_2 \cap N_1 \subseteq \Upsilon_2$, and
- $N_1 \cap \Delta_1 \cap E_2 = \emptyset$, $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ hold.

The following remark shall provide the reader with a first understanding of the above five conditions. The detailed argument why these conditions suffice for the preservation of BSPs like *BSD* will be presented in Section 6.4.

Remark 6.3.5. The set Υ_1 contains all communication events that are confidential for ES_1 but not for ES_2 (condition $C_1 \cap N_2 \subseteq \Upsilon_1$). Hence, $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ and $total(ES_1, C_1 \cap N_2)$ ensure that every perturbation that is constructed by inserting a communication event $e \in C_1 \cap N_2$ into a possible trace of ES_1 can be corrected to a possible trace of ES_1. If the event e is inserted at a position where it is immediately followed by a visible event $v \in V_1 \cap \nabla_1$ then $FCIA_{\mathcal{V}_1}^{\rho_1,\Gamma_1}(Tr_1)$ and $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ ensure that correcting the perturbation for ES_1 has no effects on ES_2 at points of the trace *before* the occurrence of v. In other words, correcting the perturbation for ES_1 can affect ES_2 only at points of the trace *after* the occurrence of v. Similarly, $C_2 \cap N_1 \subseteq \Upsilon_2$, $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$, $total(ES_2, C_2 \cap N_1)$, $FCIA_{\mathcal{V}_2}^{\rho_2,\Gamma_2}(Tr_2)$, and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ ensure that confidential communication events can be inserted into possible traces of ES_2 and

[3] Recall from Definition 3.4.25 that for $\Gamma = (\nabla, \Delta, \Upsilon)$, $FCIA^{\rho,\Gamma}$ perturbs a trace by inserting a confidential event $c \in C \cap \Upsilon$ at a position where it is ρ-admissible and where it is immediately followed by an occurrence of a visible event $v \in V \cap \nabla$. Corrections must be causal, i.e. changes may only be made after the position where c has been inserted. Moreover, changes before the occurrence of v are limited to events in $N \cap \Delta$.

that correcting this insertion, if it occurs at a position where it is immediately followed by a visible event $v \in V_2 \cap \nabla_2$, has no effects on ES_1 except for after the occurrence of v. The condition $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$ ensures that before each occurrence of a visible communication event $v \in V_1 \cap V_2$ at least one of $FCIA_{\mathcal{V}_1}^{\rho_1,\Gamma_1}(Tr_1)$ and $FCIA_{\mathcal{V}_2}^{\rho_2,\Gamma_2}(Tr_2)$ is applicable. This means, e.g., that if a communication event $e \in C_1 \cap N_2$ is inserted before a visible event $v \in V_1 \cap V_2$ that is not in ∇_1 then, although $FCIA_{\mathcal{V}_1}^{\rho_1,\Gamma_1}(Tr_1)$ cannot be applied for inserting e ($BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ must be used instead), this perturbation can cause only finitely many corrections before the occurrence of v. The reason why there is no danger of nontermination is that the changes before the occurrence of v that are caused by the corrections in ES_1 can only lead to perturbations for ES_2 that can be corrected without inserting communication events before the occurrence of v (namely by applying $FCIA_{\mathcal{V}_2}^{\rho_2,\Gamma_2}(Tr_2)$). Overall, this means that there can be only finitely many corrections before a given occurrence of a visible communication event. Since there are only finitely many visible communication events in any given trace, this implies that the process of correcting perturbations also terminates for the overall trace. ◇

In summary, the conditions for the preservation of *BSD* lead to the following definition.

Definition 6.3.6 (Well-behaved composition). Let $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ be composable event systems. Let $\mathcal{V}_1 = (V_1, N_1, C_1)$ be a view in E_1 and $\mathcal{V}_2 = (V_2, N_2, C_2)$ be a view in E_2 such that $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a proper separation of $\mathcal{V}$.

The composition of ES_1 and ES_2 is a *well-behaved composition* wrt. $\mathcal{V}_1$ and $\mathcal{V}_2$ if one of the following four conditions holds:

1. $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ hold.
2. $N_1 \cap E_2 = \emptyset$, $total(ES_1, C_1 \cap N_2)$, and $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ hold for some function ρ_1 from views in E_1 to subsets of E_1.
3. $N_2 \cap E_1 = \emptyset$, $total(ES_2, C_2 \cap N_1)$, and $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$ hold for some function ρ_2 from views in E_2 to subsets of E_2.
4. There are a function ρ_1 from views in E_1 to subsets of E_1, a function ρ_2 from views in E_2 to subsets of E_2, and triples $\Gamma_1 = (\nabla_1, \Delta_1, \Upsilon_1)$ and $\Gamma_2 = (\nabla_2, \Delta_2, \Upsilon_2)$ (where $\nabla_1, \Delta_1, \Upsilon_1 \subseteq E_1$ and $\nabla_2, \Delta_2, \Upsilon_2 \subseteq E_2$) such that
   (a) $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$, $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$, $total(ES_1, C_1 \cap N_2)$, $total(ES_2, C_2 \cap N_1)$,
   (b) $FCIA_{\mathcal{V}_1}^{\rho_1,\Gamma_1}(Tr_1)$, $FCIA_{\mathcal{V}_2}^{\rho_2,\Gamma_2}(Tr_2)$,
   (c) $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$,
   (d) $C_1 \cap N_2 \subseteq \Upsilon_1$, $C_2 \cap N_1 \subseteq \Upsilon_2$, and
   (e) $N_1 \cap \Delta_1 \cap E_2 = \emptyset$, $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ hold. ◇

Conditions 1–4 in Definition 6.3.6 can be read as a case distinction on the truth values of $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$. If $N_1 \cap E_2 = \emptyset$ holds then all communication events between the components are either visible or confidential (according to the view of the first component) and, consequently, changes made in the process of correcting a perturbation for the first component (only affect events in N_1) cannot disturb the behavior of the second component. Similarly, if $N_2 \cap E_1 = \emptyset$ holds then performing permitted corrections for the second component

cannot disturb the behavior of the first component. Hence, the four conditions in the definition differ in whether corrections in one component can have effects on the other component or not (1: no effects, 2 and 3: only in one direction, 4: in both directions).

Notational Conventions. Throughout the rest of this chapter, we assume that $ES_1 = (E_1, I_1, O_1, Tr_1)$ and $ES_2 = (E_2, I_2, O_2, Tr_2)$ denote two composable event systems. Let $ES = (E, I, O, Tr)$ denote the composition of ES_1 and ES_2, i.e. $ES = ES_1 \parallel ES_2$. Moreover, $\mathcal{V} = (V, N, C)$ denotes a view in E, ρ denotes a function from views in E to subsets of E, and Γ denotes a triple $(\nabla, \Delta, \Upsilon)$ of subsets of E. For $j \in \{1, 2\}$, $\mathcal{V}_j = (V_j, N_j, C_j)$ denotes a view in E_j, ρ_j denotes a function from views in E_j to subsets of E_j, and Γ_j denotes a triple $(\nabla_j, \Delta_j, \Upsilon_j)$ of subsets of E_j.

6.4 Compositionality of BSPs

We have shown with an example that BSPs like *BSD* are not preserved under composition, in general. However, BSPs like *BSD* are preserved under composition if certain conditions are fulfilled. We now demonstrate this by deriving compositionality theorems for our BSPs. The basis for proving these theorems is provided by a single lemma, the *generalized zipping lemma*. The use of this lemma simplifies the justification of the compositionality results that we present and also leads to uniform proofs. In our presentation, we proceed as follows: In Section 6.4.1, we present the compositionality theorems for BSPs. Then we derive the generalized zipping lemma in Section 6.4.2. Finally, we prove the compositionality theorems for BSPs with the help of this lemma in Section 6.4.3.

6.4.1 Compositionality Theorems

In the following theorem, we present compositionality results for backwards-strict BSPs and forward-correctable BSPs.

Theorem 6.4.1 (Compositionality). If $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a proper separation of $\mathcal{V}$ and the composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{V}_1$ and $\mathcal{V}_2$ then the following propositions are valid:

1. $BSD_{\mathcal{V}_1}(Tr_1) \wedge BSD_{\mathcal{V}_2}(Tr_2) \Rightarrow BSD_{\mathcal{V}}(Tr)$.
2. If $BSD_{\mathcal{V}_j}(Tr_j)$ for all $j \in \{1, 2\}$ then $BSI_{\mathcal{V}_1}(Tr_1) \wedge BSI_{\mathcal{V}_2}(Tr_2) \Rightarrow BSI_{\mathcal{V}}(Tr)$.
3. If $BSD_{\mathcal{V}_j}(Tr_j)$ and $\rho_j(\mathcal{V}_j) \subseteq \rho(\mathcal{V}) \cap E_j$ for all $j \in \{1, 2\}$ then $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr_1) \wedge BSIA^{\rho_2}_{\mathcal{V}_2}(Tr_2) \Rightarrow BSIA^{\rho}_{\mathcal{V}}(Tr)$.
4. Assume $BSD_{\mathcal{V}_j}(Tr_j)$, $\nabla \cap E_j \subseteq \nabla_j$, and $\Upsilon \cap E_j \subseteq \Upsilon_j$ hold for all $j \in \{1, 2\}$. If $\Delta \supseteq ((\Delta_1 \cap N_1) \cup (\Delta_2 \cap N_2))$, $N_1 \cap \Delta_1 \cap E_2 = \emptyset$, and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ then $FCD^{\Gamma_1}_{\mathcal{V}_1}(Tr_1) \wedge FCD^{\Gamma_2}_{\mathcal{V}_2}(Tr_2) \Rightarrow FCD^{\Gamma}_{\mathcal{V}}(Tr)$.
5. Assume $total(ES_j, C_j \cap \Upsilon_j)$, $BSD_{\mathcal{V}_j}(Tr_j)$, $BSIA^{\rho_j}_{\mathcal{V}_j}(Tr_j)$, $\nabla \cap E_j \subseteq \nabla_j$, and $\Upsilon \cap E_j \subseteq \Upsilon_j$ hold for all $j \in \{1, 2\}$. If $\Delta \supseteq ((N_1 \cap \Delta_1) \cup (N_2 \cap \Delta_2))$ and
   - $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 \subseteq \Upsilon_1$, or
   - $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ and $N_1 \cap \Delta_1 \cap E_2 \subseteq \Upsilon_2$

then $FCI^{\Gamma_1}_{\mathcal{V}_1}(Tr_1) \wedge FCI^{\Gamma_2}_{\mathcal{V}_2}(Tr_2) \Rightarrow FCI^{\Gamma}_{\mathcal{V}}(Tr)$.

6. Assume $BSD_{\mathcal{V}_j}(Tr_j)$, $\rho_j(\mathcal{V}_j) \subseteq \rho(\mathcal{V}) \cap E_j$, $BSIA^{\rho_j}_{\mathcal{V}_j}(Tr_j)$, $\nabla \cap E_j \subseteq \nabla_j$, and $\Upsilon \cap E_j \subseteq \Upsilon_j$ hold for all $j \in \{1,2\}$. If $total(ES_1, C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2)$, $total(ES_2, C_2 \cap \Upsilon_2 \cap N_1 \cap \Delta_1)$, $\Delta \supseteq ((N_1 \cap \Delta_1) \cup (N_2 \cap \Delta_2))$, and
   - $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 \subseteq \Upsilon_1$, or
   - $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ and $N_1 \cap \Delta_1 \cap E_2 \subseteq \Upsilon_2$

   then $FCIA^{\rho_1,\Gamma_1}_{\mathcal{V}_1}(Tr_1) \wedge FCIA^{\rho_2,\Gamma_2}_{\mathcal{V}_2}(Tr_2) \Rightarrow FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr)$. ◇

In the following theorem, we present a compositionality result for the non-strict BSP R.

Theorem 6.4.2 (Compositionality). If $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a proper separation of $\mathcal{V}$ and the composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{V}_1$ and $\mathcal{V}_2$ then

1. $R_{\mathcal{V}_1}(Tr_1) \wedge R_{\mathcal{V}_2}(Tr_2) \Rightarrow R_{\mathcal{V}}(Tr)$. ◇

In the following theorem, we present compositionality results for strict BSPs.

Theorem 6.4.3 (Compositionality). If $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a proper separation of $\mathcal{V}$, the composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{V}_1$ and $\mathcal{V}_2$, and $N \cap E_j = N_j$ holds for all $j \in \{1,2\}$ then the following propositions are valid:

1. $SR_{\mathcal{V}_1}(Tr_1) \wedge SR_{\mathcal{V}_2}(Tr_2) \Rightarrow SR_{\mathcal{V}}(Tr)$.
2. $SD_{\mathcal{V}_1}(Tr_1) \wedge SD_{\mathcal{V}_2}(Tr_2) \Rightarrow SD_{\mathcal{V}}(Tr)$.
3. If $SD_{\mathcal{V}_j}(Tr_j)$ for all $j \in \{1,2\}$ then $SI_{\mathcal{V}_1}(Tr_1) \wedge SI_{\mathcal{V}_2}(Tr_2) \Rightarrow SI_{\mathcal{V}}(Tr)$.
4. If $SD_{\mathcal{V}_j}(Tr_j)$ and $\rho_j(\mathcal{V}_j) \subseteq \rho(\mathcal{V}) \cap E_j$ for all $j \in \{1,2\}$ then $SIA^{\rho_1}_{\mathcal{V}_1}(Tr_1) \wedge SIA^{\rho_2}_{\mathcal{V}_2}(Tr_2) \Rightarrow SIA^{\rho}_{\mathcal{V}}(Tr)$. ◇

6.4.2 Generalized Zipping Lemma

Johnson and Thayer pioneered the technique of proving compositionality results with the help of a zipping lemma [JT88]. They used this technique to prove that forward correctability is preserved under arbitrary compositions. For this purpose, they introduced a zipping lemma that is tailored to forward correctability and, hence, cannot be used for proving compositionality results for other information flow properties. However, the idea underlying this proof technique is not limited to forward correctability. In the following, we will introduce a *generalized zipping lemma* that can be used to prove compositionality results for a variety of information flow properties (including, but not limited to, forward correctability).

Our generalized zipping lemma ensures that possible traces of system components can be merged to possible traces of the overall system. If one views the visible events and the confidential events in the component traces as the teeth of a zipper then merging the component traces (i.e. the two sides of the zipper) corresponds to closing the zipper. For example, if $\tau \in Tr$ is a possible trace of the composed system and $\tau|_{E_1}.t_1 \in Tr_1$, $\tau|_{E_2}.t_2 \in Tr_2$ are possible traces of the components then τ can be viewed as the part of a zipper that has been closed already while t_1 and t_2 can be viewed as the two sides of the part that is still open. The generalized zipping lemma states sufficient conditions for merging t_1 and t_2 to a trace $t \in E^*$ such that $\tau.t \in Tr$ holds where $\tau.t$ can be viewed as the completely closed zipper.

Lemma 6.4.4 (Generalized zipping lemma). Let $\tau \in E^*$, $\lambda \in V^*$, $t_1 \in E_1^*$, and $t_2 \in E_2^*$.

If $\tau|_{E_1}.t_1 \in Tr_1$, $\tau|_{E_2}.t_2 \in Tr_2$, $\lambda|_{E_1} = t_1|_V$, $\lambda|_{E_2} = t_2|_V$, $t_1|_{C_1} = \langle\rangle$, $t_2|_{C_2} = \langle\rangle$, $\mathcal{V}_1$ and $\mathcal{V}_2$ constitute a proper separation of $\mathcal{V}$, and the composition of ES_1 and ES_2 is well behaved then there is a trace $t \in E^*$ with $\tau.t \in Tr$, $t|_V = \lambda$, and $t|_C = \langle\rangle$. ◇

We only sketch the proof of the generalized zipping lemma, here. The detailed proof can be found in Appendix D.1.

Proof Sketch (of Lemma 6.4.4). The proof starts with a case distinction on the four conditions in the definition of a well-behaved composition (Definition 6.3.6).

Let condition 1 in Definition 6.3.6 be true. In this case, the two component traces t_1 and t_2 have only visible events in common, i.e. $t_1|_{E_2} = \lambda|_{E_1 \cap E_2} = t_2|_{E_1}$. Hence, it is possible to construct a trace $t \in E^*$ by merging t_1 and t_2 such that $t|_{E_1} = t_1$, $t|_{E_2} = t_2$, $t|_V = \lambda$, and $t|_C = \langle\rangle$. It follows from the definition of the composition operator that $\tau.t \in Tr$ holds.

Let condition 2 in Definition 6.3.6 be true. In this case, the common subsequences of the two component traces, i.e. $t_1|_{E_2}$ and $t_2|_{E_1}$, differ only in events from $N_2 \cap E_1$. We construct a trace t_1' by inserting the sequence $t_2|_{N_2 \cap C_1}$ into t_1 in a stepwise manner from left to right such that $t_1'|_{E_2} = t_2|_{E_1}$ and $\tau|_{E_1}.t_1' \in Tr_1$ (applying $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ and $total(ES_1, C_1 \cap N_2)$ in the process). Now, a trace $t \in E^*$ can be constructed by merging t_1' and t_2 such that $t|_{E_1} = t_1'$, $t|_{E_2} = t_2$, $t|_V = \lambda$, and $t|_C = \langle\rangle$ hold. It follows that $\tau.t \in Tr$ also holds.

Let condition 3 in Definition 6.3.6 be true. The proof proceeds like in the previous case.

Let condition 4 in Definition 6.3.6 be true. In this case, the common subsequences of the two component traces, i.e. $t_1|_{E_2}$ and $t_2|_{E_1}$, might differ in events from $(N_2 \cap E_1) \cup (N_1 \cap E_2)$. The proof proceeds by induction on the length of λ where the base case follows immediately from the assumptions. The step case is proved by a case distinction on the first event v in λ:

1. $v \in V_1 \cap V_2 \cap \nabla_1$ 2. $v \in V_1 \cap V_2 \cap \nabla_2$ 3. $v \in V_1 \setminus E_2$ 4. $v \in V_2 \setminus E_1$

In the first case, $FCIA_{\mathcal{V}_1}^{\rho_1,\Gamma_1}(Tr_1)$, $total(ES_1, C_1 \cap N_2)$, $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$, and $total(ES_2, C_2 \cap N_1)$ are applied in order to construct two component traces that can be merged to a trace $t \in E^*$ with $\tau.t \in Tr$, $t|_V = \lambda$, and $t|_C = \langle\rangle$. The second case is symmetric to the first case. In the third case, $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$ and $total(ES_2, C_2 \cap N_1)$ are applied in order to construct a trace $t \in E^*$ with $\tau.t \in Tr$, $t|_V = \lambda$, and $t|_C = \langle\rangle$. The fourth case is symmetric to the third case. □

The generalized zipping lemma provides a means to construct a possible trace of a composed system from possible traces of its components. The construction leaves visible events and confidential events (i.e. the teeth of the zipper) intact. In the lemma the trace $\lambda \in V^*$ can be viewed as a specification of how visible events shall be ordered after the zipper has been closed completely. Naturally, this specification must comply with the ordering of visible events and confidential events in the component traces (expressed by the assumptions $\lambda|_{E_1} = t_1|_V$ and $\lambda|_{E_2} = t_2|_V$). The conclusion $\tau.t \in Tr$ expresses that the zipper can, indeed, be closed completely and the conclusion $t|_V = \lambda$ ensures that the ordering of visible events in t is as prescribed by λ. That the teeth of the zipper (and their ordering) remain intact is ensured by the conditions $t|_V = \lambda$, $\lambda|_{E_1} = t_1|_V$, $\lambda|_{E_2} = t_2|_V$, $t|_C = \langle\rangle$, $t_1|_{C_1} = \langle\rangle$, and $t_2|_{C_2} = \langle\rangle$. Overall, this means t can be viewed as the result of "closing" t_1 and t_2 without damaging the zipper.

6.4.3 Proof of the Compositionality Theorems for BSPs

We are now ready to prove our compositionality results for BSPs with the help of the generalized zipping lemma. Here, most of the proofs are only sketched. The corresponding detailed proofs can be found in Appendix D.2.

We first sketch the proofs of the compositionality results for backwards strict BSPs.

Proof Sketch (of Theorem 6.4.1). Each proposition is proved with the help of Lemma 6.4.4.

Proposition 1: Let $\beta.\langle c\rangle.\alpha \in Tr$ be arbitrary. *BSD* is applied repeatedly on each component trace in order to delete all confidential events in $\alpha|_{E_i}$ (from right to left) and c. Then, the generalized zipping lemma is applied to construct from these component traces a possible trace $\beta.\alpha'$ of the overall system that constitutes a correction of the perturbed trace $\beta.\alpha$.

Proposition 2: Let $c \in C$ and $\beta.\alpha \in Tr$ be arbitrary. *BSD* is applied repeatedly on each component trace in order to make *BSI* applicable (deleting all confidential events in $\alpha|_{E_i}$). Then, *BSI* is applied on the resulting component traces in order to insert c. Finally, the generalized zipping lemma is applied to construct from these component traces a possible trace $\beta.\langle c\rangle.\alpha'$ of the overall system that constitutes a correction of the perturbed trace $\beta.\langle c\rangle.\alpha$.

Proposition 3: The proof proceeds like for proposition 2 where *BSIA* instead of *BSI* is used to insert c and the ρ-admissibility of c is shown before applying *BSIA*.

Proposition 4: Let $\beta.\langle c\rangle.\langle v\rangle.\alpha \in Tr$ be arbitrary. *BSD* is applied repeatedly on each component trace until all occurrences of confidential events on the right of v are removed. If c is an event of the respective component then either *BSD* or *FCD* is applied (depending on whether v is an event of the component) in order to delete c. Finally, the generalized zipping lemma is applied to construct from these component traces a possible trace $\beta.\delta.\langle v\rangle.\alpha'$ of the overall system that constitutes a correction of the perturbed trace $\beta.\langle v\rangle.\alpha$.

Proposition 5: Let $c \in C$ and $\beta.\langle v\rangle.\alpha \in Tr$ be arbitrary. *BSD* is applied repeatedly on each component trace until all occurrences of confidential events on the right of v are removed. If c is an event of the respective component then either *BSI* or *FCI* is applied (depending on whether v is an event of the component) in order to insert c. Finally, the generalized zipping lemma is applied to construct from these component traces a possible trace $\beta.\langle c\rangle.\delta.\langle v\rangle.\alpha'$ of the overall system that constitutes a correction of the perturbed trace $\beta.\langle v\rangle.\alpha$.

Proposition 6: The proof proceeds like for proposition 5 where *BSIA* (*FCIA*) instead of *BSI* (*FCI*) is used to insert c and it is explicitly shown that c is ρ-admissible. □

Next, we sketch the proof of the compositionality result for the non-strict BSP R.

Proof Sketch (of Theorem 6.4.2). Let $\tau \in Tr$ be arbitrary. R is applied on each component trace in order to remove all confidential events. Then, the generalized zipping lemma is applied to construct from these component traces a possible trace τ' of the overall system that constitutes a correction of the perturbed trace $\tau|_{V\cup N}$. □

Finally, we prove the compositionality results for strict BSPs in detail.

Proof (of Theorem 6.4.3). We prove each of the four propositions:

Proposition 1: Assume $SR_{\mathcal{V}_j}(Tr_j)$ holds for $j \in \{1,2\}$. From Theorem 3.5.6(1), we obtain that $R_{V_j\cup N_j,\emptyset,C_j}(Tr_j)$ holds. From Theorem 6.4.2(1), we obtain that $R_{V\cup N,\emptyset,C}(Tr)$ holds. The preconditions of the theorem are satisfied because $(V\cup N)\cap E_j = (V\cap E_j)\cup(N\cap E_j) = V_j\cup N_j$ holds. From Theorem 3.5.6(1), we conclude that $SR_{\mathcal{V}}(Tr)$ holds.

Proposition 2: Assume $SD_{\mathcal{V}_j}(Tr_j)$ holds for $j \in \{1,2\}$. From Theorem 3.5.6(2), we obtain that $BSD_{V_j\cup N_j,\emptyset,C_j}(Tr_j)$ holds for all $j \in \{1,2\}$. From Theorem 6.4.1(1), we obtain that $BSD_{V\cup N,\emptyset,C}(Tr)$ holds. From Theorem 3.5.6(2), we conclude that $SD_{\mathcal{V}}(Tr)$ holds.

Proposition 3: Assume that $SD_{\mathcal{V}_j}(Tr_j)$ and $SI_{\mathcal{V}_j}(Tr_j)$ hold for $j \in \{1,2\}$. From Theorems 3.5.6(2) and 3.5.15(1), we obtain that $BSD_{V_j\cup N_j,\emptyset,C_j}(Tr_j)$ and $BSI_{V_j\cup N_j,\emptyset,C_j}(Tr_j)$ hold. From Theorem 6.4.1(2), we obtain that $BSI_{V\cup N,\emptyset,C}(Tr)$ holds. From Theorem 3.5.15(1), we conclude that $SI_{\mathcal{V}}(Tr)$ holds.

Proposition 4: Assume $SD_{\mathcal{V}_j}(Tr_j)$ and $SIA^{\rho_j}_{\mathcal{V}_j}(Tr_j)$ hold for $j \in \{1,2\}$. From Theorems 3.5.6(2) and 3.5.15(2), we obtain $BSD_{V_j \cup N_j, \emptyset, C_j}(Tr_j)$ and $BSIA^{\rho'_j}_{V_j \cup N_j, \emptyset, C_j}(Tr_j)$ if $\rho'_j(V_j \cup N_j, \emptyset, C_j) = \rho_j(\mathcal{V}_j)$ holds. From Theorem 6.4.1(3), we obtain $BSIA^{\rho'}_{V \cup N, \emptyset, C}(Tr)$ if $\rho'(V \cup N, \emptyset, C) = \rho(\mathcal{V})$ holds. Note that $\rho'_j(V_j \cup N_j, \emptyset, C_j) = \rho_j(\mathcal{V}_j) \subseteq \rho(\mathcal{V}) \cap E_j = \rho'(V \cup N, \emptyset, C) \cap E_j$ holds for $j \in \{1,2\}$. From Theorem 3.5.15(2), we conclude that $SIA^{\rho}_{\mathcal{V}}(Tr)$ holds. □

6.5 Compositionality of Information Flow Properties

Information flow properties are not preserved under composition, in general [McC87]. However, some are preserved under certain conditions. To know precisely which properties are preserved under which conditions is a prerequisite for a modular information flow analysis of complex systems. To this end, we present several compositionality theorems that each state sufficient conditions for the preservation of some information flow property.

Based on the modular representation of information flow properties in *MAKS* (from Chapter 4) and the compositionality theorems for BSPs (from Section 6.4), the *verification* of compositionality theorems for more complex information flow properties becomes a straightforward task. Firstly, one proves that the views for the components constitute a proper separation of the view for the overall system (cf. Definition 6.3.2). Secondly, one verifies that the composition of the components is well behaved wrt. these views (cf. Definition 6.3.6). Thirdly, one verifies that the preconditions of the compositionality results for all BSPs from which the information flow property is assembled are satisfied. Finally, one applies the compositionality results in order to conclude that all BSPs are preserved. Since all BSPs from which the information flow property is assembled are preserved under composition, the property itself is also preserved. This approach considerably simplifies the, otherwise, often quite involved verification of compositionality results. We illustrate this simplicity by re-proving several known compositionality results from the literature in this section.

From a different perspective, the representation of information flow properties in *MAKS* and our compositionality theorems for BSPs, together, provide a basis for the *derivation* of compositionality theorems for more complex information flow properties. To this end, one collects the preconditions of the compositionality results for all BSPs from which the given information flow property is assembled. Then one eliminates all conditions that follow from the BSPs and the chosen form of composition. The remaining conditions become preconditions of the derived compositionality theorem. We illustrate this possibility by deriving some novel compositionality theorems in this section.

Proving known as well as novel compositionality results with our uniform approach also leads to a classification of these results to be presented in Section 6.6.

Notational Conventions. In the rest of this chapter, we assume that L, L_1, and L_2 denote the set of low-level events in E, E_1, and E_2, respectively ($L \cap E_1 = L_1$ and $L \cap E_2 = L_2$ hold). We assume that H, H_1, and H_2 denote the set of high-level events in E, E_1, and E_2, respectively ($H \cap E_1 = H_1$ and $H \cap E_2 = H_2$ hold). Moreover, $\mathcal{H} = (L, \emptyset, H)$, $\mathcal{H}_1 = (L_1, \emptyset, H_1)$, $\mathcal{H}_2 = (L_2, \emptyset, H_2)$, $\mathcal{HI} = (L, H \setminus I, H \cap I)$, $\mathcal{HI}_1 = (L_1, H_1 \setminus I_1, H_1 \cap I_1)$, $\mathcal{HI}_2 = (L_2, H_2 \setminus I_2, H_2 \cap I_2)$, and $\Gamma_{FC} = (I, \emptyset, I)$ shall hold.

6.5.1 Generalized Noninterference

Zakinthinos and Lee showed that, albeit *GNI* is not preserved under composition in general [McC87], it is preserved if one restricts composition to a general cascade [ZL95].[4] Let us now illustrate how easily this result can be verified with the help of *MAKS* and our compositionality results for BSPs.

Theorem 6.5.1. If $GNI(ES_1)$, $GNI(ES_2)$, and ES_1 is composed with ES_2 by a general cascade then $GNI(ES)$ holds. ◇

Proof. Let $N_i = H_i \setminus I_i$ and $C_i = H_i \cap I_i$ for $i \in \{1,2\}$. Assume $GNI(ES_1)$, $GNI(ES_2)$, and $E_1 \cap E_2 \subseteq O_1 \cap I_2$ (general cascade). From the representation theorem for *GNI* (Theorem 4.2.3), we obtain $BSD_{\mathcal{HI}_i}(Tr_i)$ and $BSI_{\mathcal{HI}_i}(Tr_i)$ for all $i \in \{1,2\}$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 3 in Definition 6.3.6) because

- $N_2 \cap E_1 = \emptyset$ follows from $E_1 \cap E_2 \subseteq O_1 \cap I_2$ and $N_2 \cap I_2 = \emptyset$,
- $total(ES_2, C_2 \cap N_1)$ follows from $BSI_{\mathcal{HI}_2}(Tr_2)$, and
- $BSIA^{\rho_2}_{\mathcal{HI}_2}(Tr_2)$ follows from $BSI_{\mathcal{HI}_2}(Tr_2)$ and Theorem 3.5.9

for an arbitrary function ρ_2 from views in E_2 to subsets of E_2. Hence our compositionality theorem for BSPs (Theorem 6.4.1(1,2)) is applicable, and we obtain $BSD_{\mathcal{HI}}(Tr)$ and $BSI_{\mathcal{HI}}(Tr)$. From the representation theorem for *GNI* (Theorem 4.2.3), we obtain $GNI(ES)$. □

Inspired by the compositionality theorem for *GNI*, we now derive a corresponding compositionality theorem for our novel property *GNI** that we introduced in order to avoid a restriction to systems that are total in the set of high-level input events. Recall from Definition 4.2.7 that the difference between *GNI** and *GNI* is that *GNI** is assembled from *BSD* and $BSIA^{\rho_C}$ while *GNI* is assembled from *BSD* and *BSI*. Due to this difference, the condition $total(ES_2, C_2 \cap N_1)$ cannot be eliminated like in the proof of Theorem 6.5.1. Therefore, we add the condition $total(ES_2, I_2 \cap H_1)$ (equivalent to $total(ES_2, C_2 \cap N_1)$) as a precondition to the derived theorem. This means, the second component in the cascade must be total in the set of all high-level inputs that are also communication events. Thereby, we arrive at the following novel compositionality theorem for *GNI**.

Theorem 6.5.2. If $GNI^*(ES_1)$, $GNI^*(ES_2)$, $total(ES_2, I_2 \cap H_1)$, and ES_1 is composed with ES_2 by a general cascade then $GNI^*(ES)$ holds. ◇

Proof. Let $N_i = H_i \setminus I_i$ and $C_i = H_i \cap I_i$ for $i \in \{1,2\}$. Assume $GNI^*(ES_1)$, $GNI^*(ES_2)$, $total(ES_2, I_2 \cap H_1)$, and $E_1 \cap E_2 \subseteq O_1 \cap I_2$ (general cascade). From the representation of *GNI** in *MAKS* (Definition 4.2.7), we obtain that $BSD_{\mathcal{HI}_i}(Tr_i)$ and $BSIA^{\rho_C}_{\mathcal{HI}_i}(Tr_i)$ hold for all $i \in \{1,2\}$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 3 in Definition 6.3.6) because

- $N_2 \cap E_1 = \emptyset$ follows from $E_1 \cap E_2 \subseteq O_1 \cap I_2$ and $N_2 \cap I_2 = \emptyset$,

[4] For a state-based system model and an interleaving-based variant of generalized noninterference, this has been shown in [McL94a].

- $total(ES_2, C_2 \cap N_1)$ holds by assumption, and
- $BSIA^{\rho_C}_{\mathcal{HI}_2}(Tr_2)$.

Hence, our compositionality theorem for BSPs (Theorem 6.4.1(1,3)) is applicable, and we obtain $BSD_{\mathcal{HI}}(Tr)$ and $BSIA^{\rho_C}_{\mathcal{HI}}(Tr)$. From the representation of GNI^* in $MAKS$ (Definition 4.2.7), we obtain $GNI^*(ES)$. □

The following two corollaries are immediate consequences of Theorem 6.5.2.

Corollary 6.5.3. If $GNI^*(ES_1)$, $GNI^*(ES_2)$, ES_2 is input total, and ES_1 is composed with ES_2 by a general cascade then $GNI^*(ES)$ holds. ◇

Corollary 6.5.4. If $GNI^*(ES_1)$, $GNI(ES_2)$, and ES_1 is composed with ES_2 by a general cascade then $GNI^*(ES)$ holds. ◇

6.5.2 Forward Correctability

Johnson and Thayer showed that FC is preserved under composition, in general [JT88].[5] Restrictiveness [McC87], which historically was the first composable information flow property, thereby became obsolete because forward correctability is strictly superior to restrictiveness in the sense that it is composable, logically less restrictive, and also easier to apply than restrictiveness (cf. Section 4.2.2). We now demonstrate how this compositionality result can be verified using our approach.

Theorem 6.5.5. If $FC(ES_1)$ and $FC(ES_2)$ then $FC(ES)$ holds. ◇

Proof. For $i \in \{1,2\}$, let $V_i = L_i$, $N_i = H_i \setminus I_i$, $C_i = H_i \cap I_i$, and $\Gamma_i = (\nabla_i, \Delta_i, \Upsilon_i)$ where $\nabla_i = I_i$, $\Delta_i = \emptyset$ and $\Upsilon_i = I_i$. Assume $FC(ES_1)$ and $FC(ES_2)$. From the representation theorem for FC (Theorem 4.2.11), we obtain that $BSD_{\mathcal{HI}_i}(Tr_i)$, $BSI_{\mathcal{HI}_i}(Tr_i)$, $FCD^{\Gamma_i}_{\mathcal{HI}_i}(Tr_i)$, and $FCI^{\Gamma_i}_{\mathcal{HI}_i}(Tr_i)$ hold for all $i \in \{1,2\}$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 4 in Definition 6.3.6) because

- $BSIA^{\rho_1}_{\mathcal{HI}_1}(Tr_1)$ and $BSIA^{\rho_2}_{\mathcal{HI}_2}(Tr_2)$ follow from $BSI_{\mathcal{HI}_1}(Tr_1)$, $BSI_{\mathcal{HI}_2}(Tr_2)$, Theorem 3.5.9;
- $total(ES_1, C_1 \cap N_2)$ and $total(ES_2, C_2 \cap N_1)$ follow from $BSI_{\mathcal{HI}_1}(Tr_1)$ and $BSI_{\mathcal{HI}_2}(Tr_2)$;
- $FCIA^{\rho_1,\Gamma_1}_{\mathcal{HI}_1}(Tr_1)$ and $FCIA^{\rho_2,\Gamma_2}_{\mathcal{HI}_2}(Tr_2)$ follow from $FCI^{\Gamma_1}_{\mathcal{HI}_1}(Tr_1)$, $FCI^{\Gamma_2}_{\mathcal{HI}_2}(Tr_2)$ and Theorem 3.5.13;
- $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$ follows from $E_1 \cap E_2 \subseteq (I_1 \cap O_2) \cup (I_2 \cap O_1)$, $\nabla_1 = I_1$ and $\nabla_2 = I_2$;
- $C_1 \cap N_2 \subseteq \Upsilon_1$ and $C_2 \cap N_1 \subseteq \Upsilon_2$ follow from $C_1 \subseteq I_1 = \Upsilon_1$ and $C_2 \subseteq I_2 = \Upsilon_2$; and
- $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ follow from $\Delta_1 = \emptyset$ and $\Delta_2 = \emptyset$

for arbitrary functions ρ_1 and ρ_2 from views in E_1 to subsets of E_1 and from views in E_2 to subsets of E_2, respectively. The remaining side conditions of our compositionality theorem for BSPs (Theorem 6.4.1(4,5)) are also fulfilled, and we obtain $BSD_{\mathcal{HI}}(Tr)$, $BSI_{\mathcal{HI}}(Tr)$, $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$, and $FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$. From the representation theorem for FC (Theorem 4.2.11), we obtain $FC(ES)$. □

[5] In [FG95], Focardi and Gorrieri show that lts-FC, i.e. their variant of forward correctability, is composable.

Inspired by the compositionality theorem for FC, let us derive a corresponding compositionality theorem for our novel property FC^*. Recall that FC^* is more liberal than FC in the sense that it does not require that systems must be input total. In the representation of FC^*, this is reflected by the use of $BSIA^{\rho_C}$ and $FCIA^{\rho_C,\Gamma_{FC}}$ instead of BSI and $FCI^{\Gamma_{FC}}$, respectively. Due to this difference in the representation, there are two conditions that cannot be eliminated like in the proof of Theorem 6.5.5, namely $total(ES_1, C_1 \cap N_2)$ and $total(ES_2, C_2 \cap N_1)$. Therefore, we add the conditions $total(ES_1, I_1 \cap H_2)$ and $total(ES_2, I_2 \cap H_1)$ as preconditions to the derived theorem. We arrive at the following novel compositionality theorem.

Theorem 6.5.6. If $FC^*(ES_1)$, $FC^*(ES_2)$, $total(ES_1, I_1 \cap H_2)$, and $total(ES_2, I_2 \cap H_1)$ then $FC^*(ES)$ holds. ◇

Proof. For $i \in \{1,2\}$, let $V_i = L_i$, $N_i = H_i \setminus I_i$, and $C_i = H_i \cap I_i$, and $\Gamma_i = (\nabla_i, \Delta_i, \Upsilon_i)$ where $\nabla_i = I_i$, $\Delta_i = \emptyset$ and $\Upsilon_i = I_i$. Assume $FC^*(ES_1)$, $FC^*(ES_2)$, $total(ES_1, I_1 \cap H_2)$, and $total(ES_2, I_2 \cap H_1)$. From the representation of FC^* in *MAKS* (Definition 4.2.13), we obtain that $BSD_{\mathcal{HI}_i}(Tr_i)$, $BSIA^{\rho_C}_{\mathcal{HI}_i}(Tr_i)$, $FCD^{\Gamma_i}_{\mathcal{HI}_i}(Tr_i)$, and $FCIA^{\rho_C,\Gamma_i}_{\mathcal{HI}_i}(Tr_i)$ hold for all $i \in \{1,2\}$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is also well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 4 in Definition 6.3.6) because

- $BSIA^{\rho_C}_{\mathcal{HI}_1}(Tr_1)$ and $BSIA^{\rho_C}_{\mathcal{HI}_2}(Tr_2)$ hold;
- $total(ES_1, C_1 \cap N_2)$ and $total(ES_2, C_2 \cap N_1)$ hold by assumption;
- $FCIA^{\rho_C,\Gamma_1}_{\mathcal{HI}_1}(Tr_1)$ and $FCIA^{\rho_C,\Gamma_2}_{\mathcal{HI}_2}(Tr_2)$ hold;
- $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$ follows from $E_1 \cap E_2 \subseteq (I_1 \cap O_2) \cup (I_2 \cap O_1)$, $\nabla_1 = I_1$ and $\nabla_2 = I_2$;
- $C_1 \cap N_2 \subseteq \Upsilon_1$ and $C_2 \cap N_1 \subseteq \Upsilon_2$ follow from $C_1 \subseteq I_1 = \Upsilon_1$ and $C_2 \subseteq I_2 = \Upsilon_2$; and
- $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ follow from $\Delta_1 = \emptyset$ and $\Delta_2 = \emptyset$.

The remaining side conditions of our compositionality theorem for BSPs (Theorem 6.4.1(4,6)) are also fulfilled, and we obtain $BSD_{\mathcal{HI}}(Tr)$, $BSIA^{\rho_C}_{\mathcal{HI}}(Tr)$, $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$ and $FCIA^{\rho_C,\Gamma_{FC}}_{\mathcal{HI}}(Tr)$. From the representation of FC^* in *MAKS* (Definition 4.2.13), we obtain $FC^*(ES)$. □

6.5.3 Nondeducibility for Outputs

Guttman and Nadel showed that NDO is preserved under composition if low-level user inputs are not connected [GN88]. Since NDO considers all high-level events as confidential (rather than only high-level inputs), the (original) proof of its compositionality result is of a different flavor than the ones for generalized noninterference or for forward correctability. We now show how this theorem can be verified using our uniform approach.

Theorem 6.5.7. Let $UI \subseteq I$, $UI_1 \subseteq I_1$, and $UI_2 \subseteq I_2$ be the sets of user inputs for ES, ES_1, and ES_2, respectively ($UI = UI_1 \cup UI_2$ holds). Moreover, let ES, ES_1, and ES_2 be input total.

If $NDO(ES_1)$, $NDO(ES_2)$, $UI_1 \cap E_2 = \emptyset$, and $UI_2 \cap E_1 = \emptyset$ then $NDO(ES)$ holds. ◇

Proof. Let $N_i = \emptyset$ for $i \in \{1,2\}$. Assume $NDO(ES_1)$, $NDO(ES_2)$, $UI_1 \cap E_2 = \emptyset$, and $UI_2 \cap E_1 = \emptyset$. From the representation theorem for NDO (Theorem 4.2.16), we obtain that $BSD_{\mathcal{H}_i}(Tr_i)$ and $BSIA_{\mathcal{H}_i}^{\rho_{UI_i}}(Tr_i)$ hold for all $i \in \{1,2\}$. From Example 6.3.3, we know that the views $\mathcal{H}_1$ and $\mathcal{H}_2$ constitute a proper separation of the view $\mathcal{H}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{H}_1$ and $\mathcal{H}_2$ (condition 1 in Definition 6.3.6) because $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ follow from $N_1 = \emptyset$ and $N_2 = \emptyset$. The side conditions of Theorem 6.4.1(3) are also fulfilled: $\rho_{UI_1}(\mathcal{H}_1) = H_1 \cup (L_1 \cap UI_1) = (H \cap E_1) \cup (L \cap UI \cap E_1) = (H \cup (L \cap UI)) \cap E_1 = \rho_{UI}(\mathcal{H}) \cap E_1$ (where $UI \cap E_1 = UI_1$ follows from the assumption $UI_2 \cap E_1 = \emptyset$) and $\rho_{UI_2}(\mathcal{H}_2) = H_2 \cup (L_2 \cap UI_2) = (H \cap E_2) \cup (L \cap UI \cap E_2) = (H \cup (L \cap UI)) \cap E_2 = \rho_{UI}(\mathcal{H}) \cap E_2$ (where $UI \cap E_2 = UI_2$ follows from the assumption $UI_1 \cap E_2 = \emptyset$). Hence, our compositionality theorem for BSPs (Theorem 6.4.1(1,3)) is applicable, and we obtain $BSD_{\mathcal{H}}(Tr)$ and $BSIA_{\mathcal{H}}^{\rho_{UI}}(Tr)$. From the representation theorem for NDO (Theorem 4.2.16), we obtain $NDO(ES)$. □

The derivation of a corresponding compositionality theorem for our novel property NDO^* is quite straightforward. Recall that the representation of NDO^* in $MAKS$ is identical to the one for NDO. The only difference between the two properties is that NDO requires that systems are input total while NDO^* does not. Since the input totality assumption is not used in the proof of Theorem 6.5.7, this proof also justifies the following theorem.

Theorem 6.5.8. Let $UI \subseteq I$, $UI_1 \subseteq I_1$, and $UI_2 \subseteq I_2$ be the sets of user inputs for ES, ES_1, and ES_2, respectively ($UI = UI_1 \cup UI_2$ holds).

If $NDO^*(ES_1)$, $NDO^*(ES_2)$, $UI_1 \cap E_2 = \emptyset$, and $UI_2 \cap E_2 = \emptyset$ then $NDO^*(ES)$ holds. ◇

Proof. This proof is along the same lines as the proof of Theorem 6.5.7. □

6.5.4 Noninference

For a state-based system model, McLean showed that noninference is preserved under composition [McL94a].[6] The following theorem recasts this result for the system model of event systems. We verify the theorem based on the representation of NF in $MAKS$ and our compositionality theorems for BSPs.

Theorem 6.5.9. If $NF(ES_1)$ and $NF(ES_2)$ then $NF(ES)$ holds. ◇

Proof. Let $N_i = \emptyset$ for $i \in \{1,2\}$. Assume $NF(ES_1)$ and $NF(ES_2)$. From the representation theorem for NF (Theorem 4.2.20), we obtain that $R_{\mathcal{H}_i}(Tr_i)$ holds for all $i \in \{1,2\}$. From Example 6.3.3, we know that the views $\mathcal{H}_1$ and $\mathcal{H}_2$ constitute a proper separation of the view $\mathcal{H}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{H}_1$ and $\mathcal{H}_2$ (condition 1 in Definition 6.3.6) because $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ follow from $N_1 = \emptyset$ and $N_2 = \emptyset$. We obtain from our compositionality theorem for BSPs (Theorem 6.4.2(1)) that $R_{\mathcal{H}}(Tr)$ holds. From the representation theorem for NF (Theorem 4.2.20), we obtain $NF(ES)$. □

[6] O'Halloran already presented a compositionality result for noninference [O'H90]. However, his definition of noninference differs from the one used to date (cf. Section 4.2.4). Moreover, given that nondeducibility on strategies, $SNNI$, and may-noninterference are equivalent to noninference (cf. Section 4.2.4), Millen's compositionality result for nondeducibility on strategies [Mil90], Focardi and Gorrieri's compositionality result for $SNNI$, and Schneider's compositionality result for may-noninterference [Sch01] are also related.

6.5.5 Generalized Noninference

For a state-based system model, McLean derived a theorem that implies that generalized noninference is preserved under product [McL94a, Theorem 3.2]. The following theorem recasts this result for the system model of event systems.

Theorem 6.5.10. If $GNF(ES_1)$, $GNF(ES_2)$, and ES_1 is composed with ES_2 by a product then $GNF(ES)$ holds. ◇

Proof. Assume $GNF(ES_1)$, $GNF(ES_2)$, and $E_1 \cap E_2 = \emptyset$ (product). From the representation theorem for GNF (Theorem 4.2.22), we obtain that $R_{\mathcal{HI}_i}(Tr_i)$ holds for all $i \in \{1,2\}$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 1 in Definition 6.3.6) because $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ follow from $E_1 \cap E_2 = \emptyset$. We obtain from our compositionality theorem for BSPs (Theorem 6.4.2(1)) that $R_{\mathcal{HI}}(Tr)$ holds. From the representation theorem for GNF (Theorem 4.2.22), we obtain $GNF(ES)$. □

A limitation to product (as in Theorem 6.5.10 above) is a severe restriction because the components cannot communicate with each other. McLean proposed two alternative compositionality results for generalized noninference that are more liberal wrt. inter-component communication, namely that generalized noninference is preserved under cascade if the first component satisfies noninference and that generalized noninference is preserved under cascade if the second component satisfies generalized noninterference. The following two theorems recast these results for event systems.

Theorem 6.5.11. If $GNF(ES_1)$, $GNF(ES_2)$, $NF(ES_1)$, and ES_1 is composed with ES_2 by a general cascade then $GNF(ES)$ holds. ◇

Proof. Let $V_1 = L_1$, $N_1 = \emptyset$, $C_1 = H_1$, $V_2 = L_2$, $N_2 = H_2 \setminus I_2$, and $C_2 = H_2 \cap I_2$. Assume $GNF(ES_1)$, $GNF(ES_2)$, $NF(ES_1)$, and $E_1 \cap E_2 \subseteq O_1 \cap I_2$ (general cascade). From the representation theorems for NF (Theorem 4.2.20) and GNF (Theorem 4.2.22), we obtain $R_{\mathcal{H}_1}(Tr_1)$ and $R_{\mathcal{HI}_2}(Tr_2)$. The views $\mathcal{H}_1$ and $\mathcal{HI}_2$ constitute a proper separation of $\mathcal{HI}$ because $L \cap E_1 = L_1$, $L \cap E_2 = L_2$, $H \cap I \cap E_1 = H_1 \cap I \subseteq H_1$, $H \cap I \cap E_2 = H_2 \cap I \cap E_2 \subseteq H_2 \cap I_2$, and $\emptyset \cap (H_2 \setminus I_2) = \emptyset$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{H}_1$ and $\mathcal{HI}_2$ (condition 1 in Definition 6.3.6) because $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ follow from $N_1 = \emptyset$ and $E_1 \cap E_2 \subseteq O_1 \cap I_2$. We obtain from our compositionality theorem for BSPs (Theorem 6.4.2(1)) that $R_{\mathcal{HI}}(Tr)$ holds. From the representation theorem for GNF (Theorem 4.2.22), we obtain $GNF(ES)$. □

Theorem 6.5.12. If $GNF(ES_1)$, $GNF(ES_2)$, $GNI(ES_2)$, and ES_1 is composed with ES_2 by a general cascade then $GNF(ES)$ holds. ◇

Proof. Let $N_i = H_i \setminus I_i$ and $C_i = H_i \cap I_i$ for $i \in \{1,2\}$. Assume $GNF(ES_1)$, $GNF(ES_2)$, $GNI(ES_2)$, and $E_1 \cap E_2 \subseteq O_1 \cap I_2$ (general cascade). From the representation theorems for GNF (Theorem 4.2.22) and GNI (Theorem 4.2.3), we obtain $R_{\mathcal{HI}_1}(Tr_1)$, $R_{\mathcal{HI}_2}(Tr_2)$, $BSD_{\mathcal{HI}_2}(Tr_2)$, and $BSI_{\mathcal{HI}_2}(Tr_2)$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 3 in Definition 6.3.6) because

- $N_2 \cap E_1 = \emptyset$ follows from $E_1 \cap E_2 \subseteq O_1 \cap I_2$ and $N_2 \cap I_2 = \emptyset$,

- $total(ES_2, C_2 \cap N_1)$ follows from $BSI_{\mathcal{HI}_2}(Tr_2)$, and
- $BSIA^{\rho_C}_{\mathcal{HI}_2}(Tr_2)$ follows from $BSI_{\mathcal{HI}_2}(Tr_2)$ and Theorem 3.5.9

for an arbitrary function ρ_2 from views in E_2 to subsets of E_2. Hence, our compositionality theorem for BSPs (Theorem 6.4.2(1)) is applicable, and we obtain $R_{\mathcal{HI}}(Tr)$. From the representation theorem for *GNF* (Theorem 4.2.22), we obtain $GNF(ES)$. □

Note that Theorem 6.5.11 holds for similar reasons as Theorem 6.5.10, namely, corrections in one component cannot have an effect on the other component and vice versa (i.e. $N_1 \cap E_2 = \emptyset = N_2 \cap E_1$ holds). However, this is ensured by different means: by restricting composition to product (in Theorem 6.5.10) and by restricting composition to general cascade in combination with requiring *NF* to hold for the first component (in Theorem 6.5.11). In contrast to this, Theorem 6.5.12 holds for a different reason. Namely, corrections in the first component of a cascade can cause perturbations for the second component (i.e. $N_1 \cap E_2 \neq \emptyset$) and it is possible to correct these perturbations (i.e. $N_2 \cap E_1 = \emptyset$, $total(ES_2, C_2 \cap N_1)$, and $BSIA^{\rho_2}_{\mathcal{HI}_2}(Tr_2)$ hold). Differences and similarities of this kind will be investigated in more detail in Section 6.6, where we also present a classification of compositionality results.

6.5.6 Separability

McLean showed that separability is preserved under composition, in general [McL94a]. Separability is a pretty restrictive property because it prevents information flow between the high level and the low level in both directions. In other words, it logically simulates a physical air-gap between the two levels (cf. Section 4.2.6). Hence, it is quite natural that this property is preserved under composition. Let us recast this result for the system model of event systems and let us demonstrate how it can be verified with our approach.

Theorem 6.5.13. If $SEP(ES_1)$ and $SEP(ES_2)$ then $SEP(ES)$ holds. ◇

Proof. Let $N_i = \emptyset$ for $i \in \{1, 2\}$. Assume $SEP(ES_1)$, and $SEP(ES_2)$. From the representation theorem for *SEP* (Theorem 4.2.24), we obtain that $BSD_{\mathcal{H}_i}(Tr_i)$ and $BSIA^{\rho_C}_{\mathcal{H}_i}(Tr_i)$ hold for all $i \in \{1, 2\}$. From Example 6.3.3, we know that the views $\mathcal{H}_1$ and $\mathcal{H}_2$ constitute a proper separation of the view $\mathcal{H}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{H}_1$ and $\mathcal{H}_2$ (condition 1 in Definition 6.3.6) because $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ follow from $N_1 = \emptyset$ and $N_2 = \emptyset$. The side conditions of Theorem 6.4.1(3) are also fulfilled: $\rho_C(\mathcal{H}_1) = H_1 = H \cap E_1 = \rho_C(\mathcal{H}) \cap E_1$ and $\rho_C(\mathcal{H}_2) = H_2 = H \cap E_2 = \rho_C(\mathcal{H}) \cap E_2$. Hence, our compositionality theorem for BSPs (Theorem 6.4.1(1,3)) is applicable, and we obtain $BSD_{\mathcal{H}}(Tr)$ and $BSIA^{\rho_C}_{\mathcal{H}}(Tr)$. From the representation theorem for *SEP* (Theorem 4.2.24), we obtain $SEP(ES)$. □

6.5.7 Perfect Security Property

Zakinthinos and Lee [ZL97] introduced the perfect security property as an alternative to separability that is less restrictive wrt. non-critical information flow from the low level to the high level (cf. Section 4.2.7). They also showed that the perfect security property is a composable property. We now show how easily this result can be verified using our approach.

Theorem 6.5.14. If $PSP(ES_1)$ and $PSP(ES_2)$ then $PSP(ES)$ holds. ◇

Proof. Let $N_i = \emptyset$ for $i \in \{1,2\}$. Assume $PSP(ES_1)$, and $PSP(ES_2)$. From the representation theorem for *PSP* (Theorem 4.2.26), we obtain that $BSD_{\mathcal{H}_i}(Tr_i)$ and $BSIA^{\rho_E}_{\mathcal{H}_i}(Tr_i)$ hold for all $i \in \{1,2\}$. From Example 6.3.3, we know that the views $\mathcal{H}_1$ and $\mathcal{H}_2$ constitute a proper separation of the view $\mathcal{H}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{H}_1$ and $\mathcal{H}_2$ (condition 1 in Definition 6.3.6) because $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ follow from $N_1 = \emptyset$ and $N_2 = \emptyset$. The side conditions of Theorem 6.4.1(3) are also fulfilled: $\rho_E(\mathcal{H}_1) = E_1 = E \cap E_1 = \rho_E(\mathcal{H}) \cap E_1$ and $\rho_E(\mathcal{H}_2) = E_2 = E \cap E_2 = \rho_E(\mathcal{H}) \cap E_2$. Hence, our compositionality theorem for BSPs (Theorem 6.4.1(1,3)) is applicable, and we obtain $BSD_{\mathcal{H}}(Tr)$ and $BSIA^{\rho_E}_{\mathcal{H}}(Tr)$. From the representation theorem for *PSP* (Theorem 4.2.26), we obtain $PSP(ES)$. □

This concludes the illustration of our approach to verifying compositionality theorems for information flow properties. We are now ready to elaborate a more structured perspective on the derived results.

6.6 A Classification of Compositionality Results

Verifying a given information flow property means showing that every modification of a possible trace resulting from a perturbation can be corrected to a possible trace as permitted by this property. During the analysis of complex systems, perturbations of the overall traces are reduced to perturbations of the component traces, these local perturbations are corrected independently for each component, and a possible trace of the overall system is constructed from the corrected component traces such that it constitutes a correction of the initial perturbation. The key problem in this process is that local corrections of a perturbation for one component may have an effect on other components. This can make it difficult, or even impossible, to construct a valid trace of the overall system from the component traces. For example, if two systems ES_1 and ES_2 that each satisfy generalized noninference are composed by a proper cascade then correcting the removal of all high-level inputs in the trace of the first component may involve the insertion of high-level outputs. Since high-level outputs of ES_1 are high-level inputs of ES_2, this means that the trace of ES_2 is perturbed by *inserting* high-level inputs and it is not ensured that this perturbation can be corrected because generalized noninference ensures only that the *removal* of high-level inputs can be corrected. Hence, generalized noninference is not amenable to a modular information flow analysis.[7]

There are three solutions to the problem that inter-component communication might interfere with a modular information flow analysis. The first solution is to ensure for each component that local corrections do not cause perturbations for the respective other component or, in other words, that both components are *polite* to each other. This means that the local corrections in each component are restricted to events not used for inter-component communication or, more formally, that $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ hold. The second solution is to tackle the problem of inter-component communication in the local information flow analysis of a single component. More specifically, this means that corrections in this component do not cause perturbations for the other component (like in the first solution) and that this component can correct all perturbations that are caused by the other component (unlike in the first solution). In other words, there is a component that is *polite* as well as *tolerant* to the other component. Formally, this means that either $N_1 \cap E_2 = \emptyset$ or $N_2 \cap E_1 = \emptyset$ must hold in addition to conditions that ensure the possibility to locally correct perturbations caused by

[7] In fact, generalized noninference is not preserved under cascade, in general [Zak96].

the other component. The third solution is to not constrain the local corrections permitted in the components (neither $N_1 \cap E_2 = \emptyset$ nor $N_2 \cap E_1 = \emptyset$ holds) but rather to ensure that each component can correct all perturbations caused by the respective other component. In other words, each component is *tolerant* wrt. the effects of corrections in the other component.

Interestingly, the justification for each compositionality theorem that we have presented in Section 6.5 is based on one of these three solutions. This leads to three classes of compositionality theorems where theorems in the same class can be justified with the same argument, which means that they hold for a similar reason. In the remainder of this section, we classify all compositionality theorems from Section 6.5 and discuss the similarities and differences between the theorems within each class.

Theorems for the Composition of Polite Components (p/p). Compositionality theorems in this class avoid interferences of inter-component communication with a modular information flow analysis by restricting the permitted corrections in both components. That is, both components are forced to be polite to each other wrt. their connections. Preventing local corrections in one component from affecting the other component ensures that the local information flow analyses of the components do not interfere with each other.

We have identified three approaches to ensure that $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ hold.

The first approach is to restrict composition to product. Since $E_1 \cap E_2 = \emptyset$ holds for composition by product, $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ also hold. An example for a theorem based on this approach is the preservation of *GNF* under product (cf. Theorem 6.5.10).

The second approach is to choose local views for the components that consider each event either as visible or as confidential (i.e. $N_1 = \emptyset = N_2$ holds). For a two-level security policy, this means that *all* high-level events are confidential (and not, e.g., only high-level input events) or, in other words, the view $\mathcal{H} = (L, \emptyset, H)$ must be used. The compositionality results for *NDO* (cf. Theorem 6.5.7), for *NDO** (cf. Theorem 6.5.8), for *NF* (cf. Theorem 6.5.9), for *SEP* (cf. Theorem 6.5.13), and for *PSP* (cf. Theorem 6.5.14) all follow this approach.

The third approach is to ensure one of $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ by restricting composition and to ensure the other condition by restricting the view of one component. For example, the result that *GNF* is preserved under general cascade if the first component satisfies *NF* (cf. Theorem 6.5.11) follows this approach. Namely, the restriction to general cascade ensures that $N_2 \cap E_1 = \emptyset$ holds (follows from $E_1 \cap E_2 \subseteq O_1 \cap I_2$ and $\mathcal{H}\mathcal{I}_2 = (L_2, H_2 \backslash I_2, H_2 \cap I_2)$) and $NF(ES_1)$ ensures that $N_1 \cap E_2 = \emptyset$ holds (follows from $\mathcal{H}_1 = (L_1, \emptyset, H_1)$).

Compositionality Theorems for a Component that is Polite and Tolerant (pt). Compositionality theorems that belong to this class reduce the global problem that inter-component communication might interfere with a modular information flow analysis to the following local problem of a single component: firstly, corrections in this component may not cause perturbations for the other component and, secondly, it must be possible to correct all perturbations that are caused by the other component, i.e. $N_1 \cap E_2 = \emptyset$ or $N_2 \cap E_1 = \emptyset$ must hold. That is, this component must be polite to the other component wrt. its own corrections and, at the same time, it must be tolerant wrt. the effects of corrections in the other component. To simplify the presentation, let us focus on the case $N_2 \cap E_1 = \emptyset$.

The condition $N_2 \cap E_1 = \emptyset$ can be established by restricting composition to general cascade (implies $E_1 \cap E_2 \subseteq O_1 \cap I_2$) and to choose a view for the second component such that N_2 contains no input events, i.e. $I_2 \cap N_2 = \emptyset$. That it is possible to correct all perturbations in ES_2

that are caused by corrections in ES_1 is ensured by requiring an appropriate information flow property for ES_2. Since corrections in ES_1 may involve the deletion of occurrences of events in N_1 as well as their insertion, this information flow property must ensure that it is possible to correct the insertion as well as the deletion of events in $N_1 \cap C_2$ for ES_2. This is the case if $BSD_{\mathcal{V}_2}(Tr_2)$, $total(ES_2, C_2 \cap N_1)$, and $BSIA^{\rho_2}_{\mathcal{V}_2}(Tr_2)$ hold. Examples for compositionality theorems that are based on this approach are the preservation of *GNI* under general cascade (cf. Theorem 6.5.1) and the preservation of *GNI** under general cascade (cf. Theorem 6.5.2). Another compositionality result based on this approach is that *GNF* is preserved under general cascade if $GNI(ES_2)$ holds (cf. Theorem 6.5.12).

In analogy to the first class of compositionality theorems, a natural alternative to satisfy $N_2 \cap E_1 = \emptyset$ would be to limit the view of ES_2 such that $N_2 = \emptyset$ holds. In our presentation, this possibility is quite obvious but we are not aware of any known compositionality results that follow these lines. However, using our approach, the derivation of a compositionality theorem that can be justified along these lines becomes a straightforward task.

Theorem 6.6.1. If $GNI(ES_1)$, $SEP(ES_2)$, and $total(ES_2, H_2)$ then $GNI(ES)$ holds. ◇

Proof. Let $N_1 = H_1 \setminus I_1$, $C_1 = H_1 \cap I_1$, $N_2 = \emptyset$, and $C_2 = H_2$. Assume $GNI(ES_1)$, $SEP(ES_2)$, and $total(ES_2, H_2)$. From the representation theorems for *GNI* (Theorem 4.2.3) and *SEP* (Theorem 4.2.24), we obtain $BSD_{\mathcal{HI}_1}(Tr_1)$, $BSI_{\mathcal{HI}_1}(Tr_1)$, $BSD_{\mathcal{H}_2}(Tr_2)$, and $BSIA^{\rho_C}_{\mathcal{H}_2}(Tr_2)$. From the proof of Theorem 6.5.11, we know that the views $\mathcal{HI}_1$ and $\mathcal{H}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{H}_2$ (condition 3 in Definition 6.3.6) because

- $N_2 \cap E_1 = \emptyset$ follows from $N_2 = \emptyset$,
- $total(ES_2, C_2 \cap N_1)$ holds by assumption, and
- $BSIA^{\rho_C}_{\mathcal{HI}_2}(Tr_2)$ follows from $BSD_{\mathcal{H}_2}(Tr_2)$ and $BSIA^{\rho_C}_{\mathcal{H}_2}(Tr_2)$.

$BSD_{\mathcal{H}_2}(Tr_2)$ implies $BSD_{\mathcal{HI}_2}(Tr_2)$ (cf. Theorem 3.5.2) and $BSIA^{\rho_C}_{\mathcal{HI}_2}(Tr_2)$ and $total(ES_2, H_2)$ imply $BSI_{\mathcal{HI}_2}(Tr_2)$. Hence, our compositionality theorem for BSPs (Theorem 6.4.1(1,2)) is applicable, and we obtain $BSD_{\mathcal{HI}}(Tr)$ and $BSI_{\mathcal{HI}}(Tr)$. From the representation theorem for *GNI* (Theorem 4.2.3), we obtain $GNI(ES)$. □

Theorems for the Composition of Tolerant Components (t/t). Compositionality theorems in this class require that it is possible to correct for each component the perturbation caused by the respective other component. That is, each component must be tolerant wrt. the effects of corrections in the other component. For a two-level security policy, this means, on the one hand, that it must be possible to locally correct the insertion and deletion of high-level communication events in traces of ES_2 that are caused by local corrections in ES_1 and, on the other hand, that it must be possible to locally correct the insertion and deletion of high-level communication events in traces of ES_1 that are caused by local corrections in ES_2. Moreover, it must be ensured that the overall process of locally correcting perturbations terminates.

This can be achieved by requiring suitable instances of *BSD*, $BSIA^{\rho}$, and $FCIA^{\rho,\Gamma}$ for each component. The compositionality theorems for *FC* (cf. Theorem 6.5.5) and for *FC** (cf. Theorem 6.5.6) are based on this approach.

Insights and Implications. Note that compositionality theorems may belong to the same class although they impose different restrictions on the form of composition. For example, the class (p/p) contains compositionality theorems that restrict composition to product, theorems that restrict composition to general cascade, and theorems that permit general composition. Our classification differs from previous ones (e.g. the one in [McL94a]) as we do not classify merely according to restrictions on the form of composition but rather according to the *effect of all constraints* (including restricted forms of composition, the choice of views with $N = \emptyset$, and the use of restrictive information flow properties). This reveals similarities between previously unrelated compositionality results: for example, *GNF* is preserved under product (cf. Theorem 6.5.10) for reasons similar to why the perfect security property is preserved under general composition (cf. Theorem 6.5.14).

Our classification of compositionality theorems is summarized in Table 6.1.

class	*preserved property*	*side conditions*	*theorem*
(p/p)	*NDO*	low-level user inputs must not be connected	6.5.7
(p/p)	*NDO**	low-level user inputs must not be connected	6.5.8
(p/p)	*NF*	none	6.5.9
(p/p)	*GNF*	composition restricted to product	6.5.10
(p/p)	*GNF*	composition restricted to general cascade, ES_1 satisfies *NF*	6.5.11
(p/p)	*SEP*	none	6.5.13
(p/p)	*PSP*	none	6.5.14
(pt)	*GNI*	composition restricted to general cascade	6.5.1
(pt)	*GNI*	ES_2 satisfies *SEP*, $total(ES_2, C_2)$	6.6.1
(pt)	*GNI**	composition restricted to general cascade, $total(ES_2, I_2 \cap H_1)$	6.5.2
(pt)	*GNF*	composition restricted to general cascade, ES_2 satisfies *GNI*	6.5.12
(t/t)	*FC*	none	6.5.5
(t/t)	*FC**	$total(ES_1, I_1 \cap H_2)$, $total(ES_2, I_2 \cap H_1)$	6.5.6

Table 6.1: Classification of known compositionality results

Alternative Approaches to Derive Compositionality Theorems. In Section 6.5, we have verified compositionality theorems for complex information flow properties with the help of the modular representations of these properties in *MAKS* and our compositionality theorems for BSPs. Based on these theorems, further compositionality theorems can be derived by exploiting the taxonomy of information flow properties (cf. Section 4.3). For example, since *GNI* is preserved under general cascade and since $SEP(ES_2)$ and $total(ES_2, H_2 \cap I_2)$ together imply $GNI(ES_2)$, we obtain the following corollary.[8]

Corollary 6.6.2. If $GNI(ES_1)$, $SEP(ES_2)$, and $total(ES_2, H_2 \cap I_2)$ hold and ES_1 and ES_2 are composed by general cascade then $GNI(ES)$ holds. ◇

Obviously, numerous further compositionality theorems could be obtained along these lines.

[8] A similar result has also been presented in [McL94a].

Another possibility is to replace the form of composition in a compositionality theorem by a more restrictive form of composition. Since every product is also a general cascade (cf. Theorem 6.2.8), we obtain, for example, the following corollary.

Corollary 6.6.3. If $GNI(ES_1)$, $SEP(ES_2)$, and $total(ES_2, H_2 \cap I_2)$ hold and ES_1 and ES_2 are composed by product then $GNI(ES)$ holds. ◇

Given the results of the current chapter and of Chapter 4, the above results are not very surprising. In the next section, we obtain a more interesting novel compositionality theorem.

6.7 A Novel Composable Information Flow Property

In this section, we show that there is an information flow property that implies generalized noninterference, that is less restrictive than Johnson and Thayer's forward correctability [JT88], and that is preserved under arbitrary compositions. The existence of such an information flow property comes somewhat at a surprise because the possibility to further improve this line of composable information flow properties had remained unnoticed since the advent of forward correctability[9] and also because a statement by Zakinthinos and Lee suggested that no further improvement in this line of properties would be possible [ZL96] (cf. Remark 6.7.7).

6.7.1 Weakened Forward Correctability

Let us recall the representation of forward correctability in *MAKS* from Theorem 4.2.11:

$$FC(ES) \Leftrightarrow BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr) \wedge FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \wedge FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \ .$$

Based on this modular representation, we analyze how the restrictions of FC can be relaxed while retaining compositionality as well as the implication of *GNI*. Since *GNI* implies $BSD_{\mathcal{HI}}(Tr)$ and $BSI_{\mathcal{HI}}(Tr)$ (cf. Theorem 4.2.5), the first two conjuncts in the representation of FC cannot be discarded. However, the conjunction $BSD_{\mathcal{HI}}(Tr)$ and $BSI_{\mathcal{HI}}(Tr)$ does not constitute an information flow property that is composable in general and, therefore, additional BSPs must be added. Let us re-investigate the proof of Theorem 6.5.5 (compositionality of FC) in order to determine which other BSPs are needed. In this proof, the fact that the third conjunct in the representation of FC holds for each of the components (i.e. $FCD^{\Gamma_1}_{\mathcal{HI}_1}(Tr_1)$ and $FCD^{\Gamma_2}_{\mathcal{HI}_2}(Tr_2)$ holds for $\Gamma_1 = (I_1, \emptyset, I_1)$ and $\Gamma_2 = (I_2, \emptyset, I_2)$) is only used in the argument that $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$ holds for the overall system (where $\Gamma_{FC} = (I, \emptyset, I)$). This means, the third conjunct is not needed for the preservation of *BSD*, *BSI*, or $FCI^{\Gamma_{FC}}$ and, consequently, it can be discarded without loosing compositionality. In contrast to this, the fourth conjunct (namely $FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$) cannot be discarded. This conjunct is used in the argument that the composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (cf. Definition 6.3.6). However, it is possible to relax the parameters of $FCI^{\Gamma_{FC}}$ without loosing compositionality. More specifically, $\Delta = \emptyset$ can safely be relaxed to $\Delta = E \setminus (I \cup O)$. This is because condition 5(e) in the definition of a well-behaved composition does not imply that

[9] Recall that forward correctability itself is an improvement of McCullough's restrictiveness [McC87] in the sense that it is less restrictive, that restrictiveness is an improvement of generalized noninterference [McC87] in the sense that it is composable, and that generalized noninterference is an improvement of Sutherland's nondeducibility [Sut86] in the sense that it is a better security property (also cf. Sections 4.2.1 and 4.2.2).

the sets Δ_1 and Δ_2 must be empty. Rather, only $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ need to hold. These conditions are satisfied, e.g., if Δ_1 and Δ_2 contain only internal events, i.e. if $\Delta_1 = E_1 \setminus (I_1 \cup O_1)$ and $\Delta_2 = E_2 \setminus (I_2 \cup O_2)$ hold. This means $FCI_{\mathcal{HI}}^{\Gamma_{FC}}(Tr))$ can be replaced by $FCI_{\mathcal{HI}}^{\Gamma_{WFC}}(Tr))$ (where $\Gamma_{WFC} = (I, E \setminus (I \cup O), I)$) without loosing compositionality.[10]

This results in the following definition of *weakened forward correctability*.

Definition 6.7.1 (Weakened forward correctability). Let $\Gamma_{WFC} = (I, E \setminus (I \cup O), I)$.

$$WFC(ES) \equiv BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr) \wedge FCI_{\mathcal{HI}}^{\Gamma_{WFC}}(Tr) \ . \qquad \diamond$$

6.7.2 Integration into Taxonomy

In comparison to forward correctability, weakened forward correctability does not require any instance of *FCD*. Moreover, it requires $FCI_{\mathcal{HI}}^{\Gamma_{WFC}}(Tr)$ instead of $FCI_{\mathcal{HI}}^{\Gamma_{FC}}(Tr)$.

The following theorem shows that weakened forward correctability implies generalized noninterference and that weakened forward correctability is implied by forward correctability.

Theorem 6.7.2 (Ordering information flow properties). The following implications are valid for every event system *ES*:

- $WFC(ES) \Rightarrow GNI(ES)$
- $FC(ES) \Rightarrow WFC(ES)$ $\diamond$

Proof. The two implications follow immediately from Definition 6.7.1 and Theorems 4.2.3, 4.2.11, and 3.5.14. $\square$

The following example demonstrates that weakened forward correctability is indeed strictly less restrictive than forward correctability in the sense that there are systems for which weakened forward correctability holds but for which forward correctability does not hold.

Example 6.7.3. Let $SES_{WFC} = (S_{WFC}, s_0, E_{WFC}, I_{WFC}, O_{WFC}, T_{WFC})$ be the state-event system with $S_{WFC} = \{s_0, s_1, s_2, s_3, s_4\}$, $I_{WFC} = \{hi, li\}$, $O_{WFC} = \{ho, lo\}$, $E_{WFC} = I_{WFC} \cup O_{WFC}$, and

$$T_{WFC} = \left\{ \begin{array}{l} (s_0, hi, s_1), (s_0, ho, s_1), (s_0, li, s_4), \\ (s_1, hi, s_1), (s_1, ho, s_1), (s_1, li, s_2), \\ (s_2, hi, s_2), (s_2, ho, s_2), (s_2, li, s_2), (s_2, lo, s_3), \\ (s_3, hi, s_3), (s_3, li, s_3), \\ (s_4, hi, s_4), (s_4, ho, s_4), (s_4, li, s_4) \end{array} \right\}$$

The transition relation T_{WFC} is also viewed in Figure 6.4. For notational convenience we use *ES* and *Tr* as abbreviations of $ES_{SES_{WFC}}$ and $Tr_{SES_{WFC}}$, respectively (i.e. the event system and the set of traces induced by SES_{WFC}).

Let $H = \{hi, ho\}$ and $L = \{li, lo\}$. For $\mathcal{HI} = (\{li, lo\}, \{ho\}, \{hi\})$, we have $BSD_{\mathcal{HI}}(Tr)$, $BSI_{\mathcal{HI}}(Tr)$ and $FCI_{\mathcal{HI}}^{\Gamma_{WFC}}(Tr)$ (as will be demonstrated in Example 6.7.5). Hence, $WFC(ES)$ holds. However, we do not have $FC(ES)$ because $FCD_{\mathcal{HI}}^{\Gamma_{FC}}(Tr)$ does not hold (where $\Gamma_{FC} = (I, \emptyset, I)$): For $\tau = \langle hi.li.lo \rangle$, we have $\tau \in Tr$ but, after deleting *hi* in τ, the only possible correction to a possible trace in *Tr* that yields the same observation is $\langle ho.li.lo \rangle$. However, this correction requires the insertion of *ho* before *li*, a correction that does not comply with the requirements of $FCD_{\mathcal{HI}}^{\Gamma_{FC}}(Tr)$. $\diamond$

[10] Recall from Theorem 3.5.14(1) that $FCI_{\mathcal{HI}}^{\Gamma_{FC}}(Tr)$ implies $FCI_{\mathcal{HI}}^{\Gamma_{WFC}}(Tr)$.

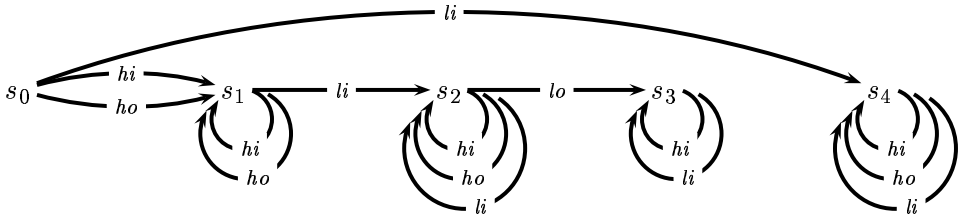


Figure 6.4: Transition relation from Example 6.7.3

6.7.3 Unwinding Theorem

Since less restrictive properties are easier to verify, weakened forward correctability seems to be, in general, preferable to forward correctability. Verification techniques for weakened forward correctability can be easily obtained by following the approach proposed in Chapter 5.

Theorem 6.7.4 (Unwinding theorem). Let $SES = (S, s_0, E, I, O, T)$ and $\ltimes_1, \ltimes_2, \ltimes_3 \subseteq S \times S$. If $lrf_{\mathcal{HI}}(T, \ltimes_1)$, $lrb_{\mathcal{HI}}(T, \ltimes_2)$, $fcrb_{\mathcal{HI}}^{\Gamma_{WFC}}(T, \ltimes_3)$, $osc_{\mathcal{HI}}(T, \ltimes_1)$, $osc_{\mathcal{HI}}(T, \ltimes_2)$, and $osc_{\mathcal{HI}}(T, \ltimes_3)$ hold then $WFC(ES_{SES})$ also holds. ◇

Proof. The theorem follows from Definition 6.7.1 and Theorem 5.3.1(1,2,5). □

Unlike in the unwinding theorem for forward correctability (cf. Theorem 5.5.5), the unwinding condition *fcrf* is not required to hold and, moreover, instead of $fcrb_{\mathcal{HI}}^{\Gamma_{FC}}(T, \ltimes_4)$, the less restrictive condition $fcrb_{\mathcal{HI}}^{\Gamma_{WFC}}(T, \ltimes_3)$ is required. Consequently, weakened forward correctability is not only a strictly less restrictive information flow property than forward correctability but also has less restrictive unwinding conditions.

Example 6.7.5. Based on Theorem 6.7.4, we are now ready to prove our claim from Example 6.7.3, i.e. that $WFC(ES)$ holds. For notational convenience, we abbreviate S_{WFC}, E_{WFC}, I_{WFC}, O_{WFC}, and T_{WFC} in this example by S, E, I, O, and T, respectively.

Let $\ltimes, \ltimes' \subseteq S \times S$ be two relations that are defined as the smallest reflexive relations satisfying $s_1 \ltimes s_0$, $s_0 \ltimes' s_1$, and $s_4 \ltimes' s_2$. For these relations, $lrf_{\mathcal{HI}}(T, \ltimes)$, $lrb_{\mathcal{HI}}(T, \ltimes')$, $fcrb_{\mathcal{HI}}^{\Gamma_{WFC}}(T, \ltimes')$, $osc_{\mathcal{HI}}(T, \ltimes)$, as well as $osc_{\mathcal{HI}}(T, \ltimes')$ hold.

To prove that $lrf_{\mathcal{HI}}(T, \ltimes)$ and $lrb_{\mathcal{HI}}(T, \ltimes')$ hold is a straightforward task because *hi* is the only confidential event and, except for $(s_0, hi, s_1) \in T$, occurrences of *hi* leave the state unchanged ($\ltimes, \ltimes'$ are reflexive). For $fcrb_{\mathcal{HI}}^{\Gamma_{WFC}}(T, \ltimes')$, only two transitions in T require special attention, namely $s_0 \xrightarrow{li} s_4$ and $s_1 \xrightarrow{li} s_2$. However, since $s_0 \overset{\langle hi.li \rangle}{\Longrightarrow} s_2$, $s_4 \ltimes' s_2$, $s_1 \overset{\langle hi.li \rangle}{\Longrightarrow} s_2$, and $s_2 \ltimes' s_2$ hold, the condition $fcrb_{\mathcal{HI}}^{\Gamma_{WFC}}(T, \ltimes')$ also holds. For proving $osc_{\mathcal{HI}}(T, \ltimes)$ and $osc_{\mathcal{HI}}(T, \ltimes')$, special attention is only required for $s_1 \ltimes s_0$, $s_0 \ltimes' s_1$, and $s_4 \ltimes' s_2$. The only transition involving a visible event that is possible in s_1 is $s_1 \xrightarrow{li} s_2$. The requirements of *osc* are satisfied for $s_1 \ltimes s_0$ because $s_0 \overset{\langle ho.li \rangle}{\Longrightarrow} s_2$ and $s_2 \ltimes s_2$. The only transition involving a visible event that is possible in s_0 is $s_0 \xrightarrow{li} s_4$. The requirements of *osc* are satisfied for $s_0 \ltimes' s_1$ because $s_1 \overset{\langle li \rangle}{\Longrightarrow} s_2$ and $s_4 \ltimes' s_2$. The only transition involving a visible event that is possible in s_4 is $s_4 \xrightarrow{li} s_4$. The requirements of *osc* are satisfied for $s_4 \ltimes' s_2$ because $s_2 \overset{\langle li \rangle}{\Longrightarrow} s_2$ and $s_4 \ltimes' s_2$. From Theorem 6.7.4, we obtain that $WFC(ES)$ holds. ◇

6.7.4 Compositionality Theorem

We now show that weakened forward correctability is preserved under general composition.

Theorem 6.7.6 (Compositionality of weakened forward correctability). If ES_1 and ES_2, both, satisfy weakened forward correctability then $ES = ES_1 \parallel ES_2$ also satisfies weakened forward correctability. ◇

Proof. For $i \in \{1,2\}$, let $\mathcal{HI}_i = (V_i, N_i, C_i)$ where $V_i = L_i$, $N_i = H_i \setminus I_i$, and $C_i = H_i \cap I_i$. Let $\Gamma_{WFC} = (\Delta, \nabla, \Upsilon)$ where $\nabla = I$, $\Delta = E \setminus (I \cup O)$, and $\Upsilon = I$. Moreover, let $\Gamma_i = (\nabla_i, \Delta_i, \Upsilon_i)$ where $\nabla_i = I_i$, $\Delta_i = E_i \setminus (I_i \cup O_i)$, and $\Upsilon_i = I_i$ for $i \in \{1,2\}$. Assume $WFC(ES_1)$ and $WFC(ES_2)$. From the representation of WFC in $MAKS$ (Definition 6.7.1), we obtain that $BSD_{\mathcal{HI}_i}(Tr_i)$, $BSI_{\mathcal{HI}_i}(Tr_i)$, and $FCI^{\Gamma_i}_{\mathcal{HI}_i}(Tr_i)$ hold for all $i \in \{1,2\}$. From Example 6.3.4, we know that the views $\mathcal{HI}_1$ and $\mathcal{HI}_2$ constitute a proper separation of the view $\mathcal{HI}$. The composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{HI}_1$ and $\mathcal{HI}_2$ (condition 4 in Definition 6.3.6) because

- $BSIA^{\rho_1}_{\mathcal{HI}_1}(Tr_1)$ and $BSIA^{\rho_2}_{\mathcal{HI}_2}(Tr_2)$ follow from $BSI_{\mathcal{HI}_1}(Tr_1)$, $BSI_{\mathcal{HI}_2}(Tr_2)$, Theorem 3.5.9;
- $total(ES_1, C_1 \cap N_2)$ and $total(ES_2, C_2 \cap N_1)$ follow from $BSI_{\mathcal{HI}_1}(Tr_1)$ and $BSI_{\mathcal{HI}_2}(Tr_2)$;
- $FCIA^{\rho_1,\Gamma_1}_{\mathcal{HI}_1}(Tr_1)$ and $FCIA^{\rho_2,\Gamma_2}_{\mathcal{HI}_2}(Tr_2)$ follow from $FCI^{\Gamma_1}_{\mathcal{HI}_1}(Tr_1)$, $FCI^{\Gamma_2}_{\mathcal{HI}_2}(Tr_2)$ and Theorem 3.5.13;
- $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$ follows from $E_1 \cap E_2 \subseteq (I_1 \cap O_2) \cup (I_2 \cap O_1)$, $\nabla_1 = I_1$ and $\nabla_2 = I_2$;
- $C_1 \cap N_2 \subseteq \Upsilon_1$ and $C_2 \cap N_1 \subseteq \Upsilon_2$ follow from $C_1 \subseteq I_1 = \Upsilon_1$ and $C_2 \subseteq I_2 = \Upsilon_2$; and
- $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ follow from $\Delta_1 = E_1 \setminus (I_1 \cup O_1)$, $\Delta_2 = E_2 \setminus (I_2 \cup O_2)$

for arbitrary functions ρ_1 and ρ_2 from views in E_1 to subsets of E_1 and from views in E_2 to subsets of E_2, respectively. Moreover, $(N_1 \cap \Delta_1) \cup (N_2 \cap \Delta_2) \subseteq \Delta$ follows from $N_i \cap \Delta_i = (H_i \setminus I_i) \cap (E_i \setminus (I_i \cup O_i)) \subseteq H_i \setminus (I \cup O)$. The remaining side conditions of our compositionality theorem for BSPs (Theorem 6.4.1(1,2,5)) are also fulfilled, and we obtain $BSD_{\mathcal{HI}}(Tr)$, $BSI_{\mathcal{HI}}(Tr)$, and $FCI^{\Gamma_{WFC}}_{\mathcal{HI}}(Tr)$. From the representation of WFC in $MAKS$ (Definition 6.7.1), we obtain $WFC(ES)$. □

Hence, weakened forward correctability is a composable security property that is strictly less restrictive than forward correctability.

Remark 6.7.7. Our results are in contrast to a statement by Zakinthinos and Lee that suggests that it would be impossible to relax the requirements of forward correctability without loosing compositionality [ZL96]. The precise statement is: "forward correctability is the weakest condition of any [composable] property that solely eliminates the possible [sic!] of there being a condition on a low-level input event" [ZL96, page 100]. Note that weakened forward correctability satisfies this conditions because it only eliminates the possibility of a condition on a low-level input event (in Zakinthinos and Lee's terminology) because $\nabla \cap V \subseteq I$ holds. ◇

6.8 Summary and Comparison to Prior Frameworks

In this chapter, we have proposed a uniform approach to deriving compositionality theorems for information flow properties. This approach simplifies the derivation of compositionality

theorems considerably, and the only prerequisite for applying it is that the information flow property under consideration must be represented in *MAKS*.

Rather than verifying compositionality theorems for every information flow property separately from scratch, our approach reduces the proof that a given information flow property is preserved under some form of composition to the proof that each basic ingredient of this property (i.e. each BSP from which it is assembled) is preserved under this form of composition. Using our approach, a given compositionality theorem is verified in four steps: firstly, one shows that the views for the components are a proper separation of the view for the overall system; secondly, one shows that the composition of the components is well behaved wrt. these views; thirdly, one proves the preconditions of the compositionality theorems for all BSPs from which the property is assembled; and, fourthly, one shows that the BSPs are preserved by applying these compositionality theorems. Given the representation of the information flow property in *MAKS*, this implies that the property is also preserved. The backbone of our approach to deriving compositionality theorems is the collection of compositionality results for BSPs that we have derived with the help of our generalized zipping lemma (cf. Section 6.4).

Using our uniform approach, we have derived several compositionality theorems for information flow properties. In particular, we have re-justified various known compositionality theorems from the literature. Thereby, we have demonstrated the generality of our approach. That the derived theorems are no more restrictive than the original ones means that the uniformity and generality of our approach does not come at the cost of the quality of the derived compositionality theorems. In addition to the re-justification of already known theorems, we have also derived novel compositionality theorems, namely for GNI^*, FC^*, and NDO^*. Moreover, we have discovered weakened forward correctability, a novel composable information flow property that constitutes an improvement of Johnson and Thayer's forward correctability [JT88], which itself is an improvement of McCullough's restrictiveness [McC87]. This contribution is not only interesting because the possibility to further improve this line of information flow properties has remained undetected for some time and because a statement by Zakinthinos and Lee suggested that such an improvement would not be possible (cf. Remark 6.7.7) but also because it provides evidence that our uniform approach to deriving compositionality theorems offers a new perspective on such results that is helpful in exploring new directions.[11]

Another outcome of our investigations is a classification of compositionality theorems. This classification is based on the observation that there are three approaches to avoiding inter-component communication from interfering with a modular information flow analysis. Namely, one ensures either that no such interferences can be caused by the local information flow analysis of any component, or one tackles the problem of inter-component interaction in the local information flow analysis of a single component, or one ensures that each component can cope with the interferences caused by the respective other component. The justification of each compositionality theorem that we have presented in this chapter follows one of these three approaches. This led us to three classes of compositionality theorems: (p/p) i.e. both components are polite, (pt) i.e. one component is polite as well as tolerant, and (t/t) i.e. both components are tolerant. Interestingly, compositionality theorems that impose different restrictions on the form of composition may belong to the same class because we regard re-

[11] Recall from Section 6.7 that the development of weakened forward correctability was motivated by our re-justification of Johnson and Thayer's compositionality theorem for forward correctability.

stricting the form of composition only as *one* option for satisfying the conditions essential for the preservation of information flow properties under composition.[12] This distinguishes our classification from McLean's classification that grouped compositionality theorems depending on which form of composition is assumed [McL94a, McL96]. The two classifications complement each other because McLean's classification clarifies better which compositionality theorems are applicable for a given system architecture while our classification clarifies better the reasons why a given compositionality theorem is valid.

Compositionality theorems have also been derived in the frameworks by McLean, by Focardi and Gorrieri, and by Zakinthinos and Lee. Focardi and Gorrieri derived compositionality theorems for noninference and forward correctability [FG95].[13] During the verification of these results, Focardi and Gorrieri applied rules of the process algebra SPA but, otherwise, could not exploit their representation of the investigated properties. The reason for this is that their framework lacks uniform concepts that are specifically targeted at representing information flow properties (cf. Section 4.4). In his thesis, Zakinthinos presents compositionality theorems for interleaving-based generalized noninterference, for noninference, generalized noninference, and the perfect security property [Zak96]. Unfortunately, the representation of these properties in the framework provides little support for simplifying the verification of these theorems. Except, for a simple (and somewhat straightforward) lemma that is used in the proofs of three compositionality theorems, each proof is constructed from scratch. In contrast to this, McLean's framework provides much more support for verifying compositionality theorems [McL94a, McL96]. The uniform concepts for representing information flow properties in his framework (selective interleaving functions) provide a basis for expressing general compositionality theorems that, once proven, can be instantiated for individual information flow properties. In this respect, McLean's approach to deriving compositionality theorems is closest to ours. However, technically it differs substantially. The main restriction of McLean's approach stems from the limited expressiveness of his framework (cf. Section 4.4 and Appendix B), which restricted him to the derivation of compositionality theorems for noninference, separability, interleaving-based generalized noninterference, and generalized noninference. Table 6.2 summarizes the compositionality theorems that have been derived in each of these frameworks. The entries in the table show that we have derived compositionality theorems for more properties than this has been done based on any other framework.[14]

Summarizing, the approach to deriving compositionality results proposed in this chapter exploits the uniform representation of information flow properties in *MAKS*. Rather than verifying compositionality results for technically complex information flow properties directly, we verify them with the help of compositionality theorems for BSPs, which simplifies the

[12] Other options are to use a restrictive view during the local information flow analysis of a component or to use a restrictive security predicate during the local information flow analysis.

[13] Here, we refer to the compositionality theorems for the operator $\|$ of SPA, which corresponds to our notion of composition. For completeness: Focardi and Gorrieri also presented a compositionality theorem for restrictiveness, i.e. an information flow property that is obsolete by now. In Focardi and Gorrieri's terminology, noninference, forward correctability, and restrictiveness are called strong nondeterministic noninterference, lts-forward correctability, and lts-restrictiveness, respectively.

[14] We have not derived compositionality results for *IBGNI* and *IBGNI** because these properties are assembled from the non-strict BSPs D, I, and IA^{ρ_C}. Our generalized zipping lemma does not apply to these BSPs because it is tailored for causal BSPs (the definition of a well-behaved composition involves instances of $BSIA^{\rho}$ and $FCIA^{\rho,\Gamma}$). However, we are confident that compositionality results for these non-strict BSPs can be derived (using a different proof technique) such that compositionality results for *IBGNI* and *IBGNI** (corresponding to the ones in [McL94a]) could also be re-justified based on *MAKS*, if desired.

	compositionality theorem derived in framework by McLean	compositionality theorem derived in framework by Focardi and Gorrieri	compositionality theorem derived in framework by Zakinthinos and Lee	compositionality theorem derived in *MAKS*
	properties protecting occurrences and nonoccurrences of all high-level events			
SEP	√			√
NDO				√
*NDO**				√
PSP			√	√
	properties protecting occurrences of all high-level events			
NF	√	√	√	√
	properties protecting occurrences and nonoccurrences of high-level inputs			
FC		√		√
*FC**				√
GNI				√
*GNI**				√
IBGNI	√		√	
*IBGNI**	√			
WFC				√
	properties protecting occurrences of high-level inputs			
GNF	√		√	√

Table 6.2: Compositionality theorems derived in prior frameworks

verification considerably. We have demonstrated the applicability of our approach by verifying compositionality results for several information flow properties. The only prerequisite for applying this approach is that the information flow property under consideration must be represented in *MAKS*. This general applicability is an appealing feature of our approach.

The compositionality results presented in this chapter and the unwinding results in Chapter 5, together, provide a basis for the modular verification of secure systems: Primitive system components are explicitly verified to be secure with the help of an unwinding theorem (from Chapter 5 and Section 6.7.3) and, when these components are composed, the security of the resulting system is ensured by applying a compositionality result (from the current chapter). How this approach can be applied during the verification of a complex system will be illustrated in detail in the case study to be presented in the following chapter.

Chapter 7

Case Study

7.1 Introduction

So far, we have applied our framework *MAKS* for deriving results about information flow properties: In Chapter 4, we have analyzed well known information flow properties from the literature and have compared them based on their representation in *MAKS*; in Chapter 5, we have developed unwinding techniques that exploit the modular representation of information flow properties in *MAKS*; and in Chapter 6, we have derived compositionality theorems also by exploiting the modularity of the representation. In this chapter, we want to go a step further by illustrating how our results can be used in concrete applications.

In the following sections, we explain how the various tasks during the construction of a formal security model can be performed. More specifically, we illustrate how a system's behavior can be specified in an event-based setting, how a system's security requirements can be expressed by information flow properties, how these security requirements can be verified for system components using our unwinding techniques, and how the security of the overall system can be verified with the help of our compositionality results. For each of these tasks, we briefly outline which subtasks must be addressed and then demonstrate how this can be done with a running example.

Our running example comes from the area of language-based security. A typical application scenario for language-based security techniques has been described by Sabelfeld and Sands [SS00]: *Given you have confidential data as well as public data on your computer and you want to run an untrusted program (e.g. obtained from the Internet). The application (e.g. a spreadsheet) might require legitimate access to the confidential data in order to perform its task as well as legitimate communication with the supplier of the code (e.g. a registration process for all users). Nevertheless, you want to be sure that the program does not reveal any confidential data to the supplier of the code.* After the program has been checked using some program analysis technique you know that it is "secure" (in some sense) to run this program. The objective of language-based security techniques is to check whether it is secure to execute a *given* program. These techniques do not aim at supporting the software engineer during the development of secure systems because they can only be applied to a system after it has been implemented. In this respect, program analysis techniques differ from the specification-based techniques considered in Chapters 3–6. To embed a particular program analysis technique into a more general specification-based framework is the objective of our case study.

Historically, the use of program analysis to rule out information leakage has been pioneered

by Denning [Den76, DD77]. More recent research in this direction has resulted in security type systems for Denning-style program analysis [VSI96].[1] Security type systems can be used to mechanically check whether untrusted programs are "secure" *in some sense*. In other words, security analysis for a given program boils down to a fully automatic type check. Lately, it has become common practice that security type systems are accompanied by a formal soundness proof. These proofs increase the confidence in a type system by showing that if a program passes the type check then it also satisfies some more abstract security definition. This means, a soundness proof exemplifies *in what sense* a type-correct program is "secure". Given that early program analysis techniques did not feature *formal* soundness proofs, this is a desirable movement. However, one shortcoming is that the underlying security definitions lack a formal comparison to more established security properties and, therefore, often appear somewhat ad hoc. In particular, the relation of language-based security techniques and information flow properties in the spirit of noninterference had not been formally elaborated (although such a connection has often been conjectured [SV98, SS00]). To establish such a rigorous connection was the main motivation for performing our case study.

In this case study, we investigate an extension of Sabelfeld and Sands's approach to language-based security [SS00] for a programming language that incorporates features for multi-threaded and distributed programming. We presume (from [SS00, SM02, MS03a]): the multi-threaded and distributed programming language DMWL, the language-based security definition strong security, a security type system, and a proof that all type-correct programs are also strongly secure. Using event systems, we model the behavior of DMWL processes and express their security requirements by an information flow property in *MAKS*. Finally, we show that this information flow property is implied by strong security. By relating strong security to a more established class of security definitions, our result adds to the understanding of strong security. Thereby, we further increase the confidence in this particular language-based security definition and the corresponding program analysis techniques (i.e. the security type system) for checking it.

Although the relation established by our case study is interesting in its own right, the focus in the current chapter is not on this original contribution. Rather, we use this case study to illustrate the various steps necessary in the construction of a formal security model and to point out how the results presented in previous chapters can be applied in the process.

Besides the domain of language-based security, i.e. the domain of the case study presented in the rest of this chapter, many other application domains for information flow properties can benefit from our results. Examples are the analysis of access control models [Rus92], the analysis of security protocols [FGG97, FGM99, FGM00, DFG00], the construction of formal security models for operating systems [SRS+00], the security analysis of mobile devices [MSK+01], and the specification of security requirements of multi-agent systems [HMS03, Sch03].

Overview. Section 7.2 provides some preliminaries on the language-based security techniques. A specification of DMWL processes with state-event systems is presented in Section 7.3. This specification models the semantics of DMWL. In Section 7.4, we demonstrate how the security requirements for a DMWL process can be expressed by an information flow property that we define in *MAKS*. In Section 7.5, we illustrate how our verification techniques

[1] For recent work on static analysis for security also cf. [HR98, SV98, Aga00, SS00, Lau01, ZM02, BN02, PHW02]. We refer to [SM03] to an up-to-date overview on language-based security.

can be exploited in a soundness proof for strong security and the security type system (for individual processes). The application of our compositionality results is demonstrated in Section 7.6 where we lift the soundness result to distributed programs (collections of processes). The case study presented in this chapter has also been performed in slightly different settings (in [MS01, SM02, MS03a]) that we review in Section 7.7. The main results of this Chapter are summarized in Section 7.8. This chapter is accompanied by an appendix that contains further details of the case study (cf. Appendix E).

7.2 Preliminaries on the Running Example

Sabelfeld and Sands's investigated language-based security in the context of a simple multi-threaded programming language, the MWL language [SS00]. The MWL language is an imperative programming language that incorporates commands for assignments ($var := Exp$), conditional branching (if B then C_1 else C_2), while loops (while B do C), and dynamic thread creation (fork($C\ \vec{D}$)).[2] Concurrent threads communicate with each other via shared memory, i.e. via variables. Variables are classified either as *high* or *low* with the intuitive understanding that high variables initially contain private data while low variables only contain public data. The contents of high variables is not observable by adversaries (the ones who must not learn your private data) while the contents of low variables is (potentially) observable during program execution. In summary, the security requirement is: *the initial value of high variables must not be leaked into low variables.*

Before presenting more details, let us illustrate with examples the subtle possibilities for leaking information. For simplicity we assume in the following that there are only two variables: a high variable h and a low variable l.[3]

$l := h$ The value of h is simply copied into l. This is an example of a so called *explicit* flow.

if $h = 1$ **then** $l := 1$ **else** $l := 0$ Depending on the value of h either the first or the second branch of the conditional is chosen and this fact is recorded in l. In particular, if $l = 1$ holds after program execution then $h = 1$ did hold before program execution. This exemplifies a so called *implicit* flow.

$l := 0; (\textbf{while}\ h \neq 1\ \textbf{do skip}); l := 1$ Depending on the value of h, the program either terminates or not. If $l = 1$ holds then the program has terminated and the adversary knows that $h = 1$ did hold before program execution. This is an example for an information leak *via termination behavior.*

$l := 0;$ **if** $h = 1$ **then sleep** $10000; l := 1$ An attacker with a stop-watch can learn whether $h = 1$ or $h \neq 1$ holds by measuring the duration between the two assignments to l. This is an example for an information leak *via externally observable timing.* Sabelfeld and Sands do not try to detect this kind of information leakage.[4] However, they are interested in detecting the following more subtle possibility for information leakage.

[2] The command fork($C\ \vec{D}$) creates a vector $\vec{D}$ of threads that run concurrently with the current thread that continues by executing the command C.

[3] A restriction to only two variables h and l has also been made in [SS00]. However, this is only assumed for clarity of the presentation. It is not a principal limitation of the approach.

[4] Information leakage via externally observable timing can be prevented, for example, by denying untrusted programs access to high precision clocks.

$$\textbf{fork}\left(\begin{array}{c} l := 1 \\ (\textbf{if } h = 1 \textbf{ then while } h < 10000 \textbf{ do } h := h + 1); l := 0 \end{array}\right)$$

After execution of the fork-command, two threads run concurrently (i.e. $l := 1$ and if $h = 1$ then $\ldots; l := 0$). The first thread sets l to 1 and the second thread sets l to 0 (possibly after consuming some time in the while loop). Apparently, the duration for executing the overall program depends on the initial value of h. Hence, there is a danger of information leakage via externally observable timing. However, unlike in the previous example, information leakage is possible even if no clock is available. Obviously, the final value of l depends on whether $l := 1$ is executed before or after $l := 0$. With many schedulers (the likelihood of) the order in which these two commands are executed will depend on whether the loop is executed or not. If the loop is executed then it becomes more likely that the first thread executes its only command before the second thread executes its final command. Consequently, if $h = 1$ holds initially then it is more likely that $l = 0$ holds after program execution in comparison to the case where $h \neq 1$ holds initially. This is an example for an information leak *via internally observable timing.*

Note that the danger of information leakage via internally observable timing heavily depends on the particular scheduler. For example, with a round-robin scheduler that deterministically selects the first thread after execution of a fork-command there would be no information leakage ($l := 1$ is always executed first). However, information leakage is possible, for instance, if the scheduler is a uniform probabilistic scheduler or a round-robin scheduler that deterministically selects the second thread after execution of a fork-command. However, the scheduler is typically not specified by the language definition (and therefore may vary from one implementation to another). In particular, the precise scheduler might not be known at the time of program analysis. In a worst-case scenario, the scheduler may even be chosen by the adversary. Therefore, it is desirable to have a security condition that does not depend on a particular scheduler. The *strong security* definition proposed by Sabelfeld and Sands's is such a condition, i.e. it is scheduler independent wrt. a large class of schedulers including deterministic schedulers (e.g. round robin) as well as probabilistic schedulers (e.g. uniform). Although the class of schedulers covered by strong security includes probabilistic schedulers, the strong security condition is a possibilistic property. This has been elaborated in [SS00], and a security type system for MWL has also been proposed. This type system is sound wrt. strong security in the sense that if a program passes a type check then it also satisfies strong security. A soundness result wrt. a different security condition that stems from a more established class of security properties, namely that of noninterference-like information flow properties has been elaborated in joint work by Sabelfeld and myself [MS01].

Extension of MWL to DMWL. Sabelfeld and Sands's results for MWL (including the definition of strong security, the security type system and the soundness proof) have been generalized in joint work by Sabelfeld and myself to the language DMWL [MS03a, SM02]. DMWL is an extension of MWL with (message-passing) communication primitives and a DMWL program may execute as multiple processes (each of which may have multiple threads). The (extended) program analysis techniques for DMWL constitute the starting point of our case study and are briefly introduced in the following.

7.2.1 Syntax and Semantics of DMWL

The language DMWL results from extending MWL with a non-blocking send command ($\mathsf{send}(cid, Exp)$) and two receive commands, a blocking version ($\mathsf{receive}(cid, var)$) and a non-blocking one ($\mathsf{if\text{-}receive}(cid, var, C_1, C_2)$). The syntax of DMWL commands is specified by the grammar in Figure 7.1. As usual, boolean expressions B range over $BOOL$ and arithmetic expressions Exp range over EXP. Let C, D (possibly with indices and primes) range over commands CMD (the DMWL-threads), and $\vec{C}, \vec{D}$ range over vectors of commands $\vec{CMD} = \bigcup_{n \in \mathbb{N}} CMD^n$. VAR denotes the set of variables (for simplicity $VAR = \{h, l\}$), VAL denotes a set of (not further specified) values, and CID denotes a set of channel identifiers. Moreover, var denotes a variable in VAR and cid denotes a channel identifier in CID.

$$C ::= \mathsf{skip} \mid var := Exp \mid C_1; C_2 \mid \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \mid \mathsf{while}\ B\ \mathsf{do}\ C \mid \mathsf{fork}(C\ \vec{D})$$
$$\mid \mathsf{send}(cid, Exp) \mid \mathsf{receive}(cid, var) \mid \mathsf{if\text{-}receive}(cid, var, C_1, C_2)$$

Figure 7.1: Command syntax

The operational semantics of DMWL processes is defined in terms of local configurations.

Definition 7.2.1 (Local configuration). A *local configuration* for a process is a triple $\langle \vec{C}, mem, \sigma \rangle$ where $\vec{C} \in \vec{CMD}$, $mem : VAR \to VAL$, and $\sigma : CID \to VAL^*$. ◇

The command vector $\vec{C}$ in a local configuration models the state of all threads of the process. For $\vec{C} = C_1 C_2 \ldots C_n$,[5] the command C_i is to be executed by the ith thread of the process. The memory-status function mem models the state of the process, i.e. the values of all local variables. For example, $mem(l)$ denotes the value of the variable l. The threads of a process communicate with each other via these variables. However, variables are not intended for inter-process communication. Rather, inter-process communication follows the message-passing paradigm, i.e. messages are exchanged between processes via communication channels that can be thought of as links between processes. The *channel-status function* σ models the state of all communication channels of the overall system. For example $\sigma(cid)$ denotes the sequence of messages that have been sent on the channel with identifier cid but that have not yet been delivered.

Example 7.2.2. Let $mem(h) = 1$, $mem(l) = 0$, $\sigma(cid) = \langle\rangle$ (for all $cid \in CID$), and

$$\vec{C} = \begin{pmatrix} l := 1 \\ (\mathsf{if}\ h = 1\ \mathsf{then}\ \mathsf{while}\ h < 10000\ \mathsf{do}\ h := h + 1); l := 0 \end{pmatrix}$$

Then $\langle \vec{C}, mem, \sigma \rangle$ is a local configuration that incorporates two concurrent threads, the memory $(1, 0)$, and a bunch of empty channels. ◇

The deterministic part of the semantics is defined by transition rules between local configurations in Figure 7.2. Arithmetic and boolean expressions are executed atomically by $\downarrow$ transitions. $Exp \downarrow^{mem} val$ denotes that $Exp \in EXP$ evaluates to $val \in VAL$ where the memory mem in the index is only important if Exp contains variables. Similarly, $B \downarrow^{mem} tt$ and $B \downarrow^{mem} ff$ denote that $B \in BOOL$ evaluates to "true" and "false", respectively.

[5] For better readability, we usually omit angle brackets for command vectors.

commands with local effects only

[Skip] $\langle \mathsf{skip}, mem, \sigma \rangle \rightarrowtriangle \langle \langle\rangle, mem, \sigma \rangle$

[Assign] $$\frac{Exp \downarrow^{mem} val}{\langle var := Exp, mem, \sigma \rangle \rightarrowtriangle \langle \langle\rangle, mem[var \mapsto val], \sigma \rangle}$$

[If$_{tt}$] $$\frac{B \downarrow^{mem} tt}{\langle \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, mem, \sigma \rangle \rightarrowtriangle \langle C_1, mem, \sigma \rangle}$$

[If$_{ff}$] $$\frac{B \downarrow^{mem} ff}{\langle \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, mem, \sigma \rangle \rightarrowtriangle \langle C_2, mem, \sigma \rangle}$$

[While$_{tt}$] $$\frac{B \downarrow^{mem} tt}{\langle \mathsf{while}\ B\ \mathsf{do}\ C, mem, \sigma \rangle \rightarrowtriangle \langle C; \mathsf{while}\ B\ \mathsf{do}\ C, mem, \sigma \rangle}$$

[While$_{ff}$] $$\frac{B \downarrow^{mem} ff}{\langle \mathsf{while}\ B\ \mathsf{do}\ C, mem, \sigma \rangle \rightarrowtriangle \langle \langle\rangle, mem, \sigma \rangle}$$

[Fork] $\langle \mathsf{fork}(C\vec{D}), mem, \sigma \rangle \rightarrowtriangle \langle C\vec{D}, mem, \sigma \rangle$

communication commands

[Send] $$\frac{Exp \downarrow^{mem} val}{\langle \mathsf{send}(cid, Exp), mem, \sigma \rangle \rightarrowtriangle \langle \langle\rangle, mem, \sigma[cid \mapsto \langle val \rangle.\sigma(cid)] \rangle}$$

[Receive] $$\frac{\sigma(cid) = vals.\langle val \rangle}{\langle \mathsf{receive}(cid, var), mem, \sigma \rangle \rightarrowtriangle \langle \langle\rangle, mem[var \mapsto val], \sigma[cid \mapsto vals] \rangle}$$

[IfRcv$_{ff}$] $$\frac{\sigma(cid) = \langle\rangle}{\langle \mathsf{if\text{-}receive}(cid, var, C_1, C_2), mem, \sigma \rangle \rightarrowtriangle \langle C_2, mem, \sigma \rangle}$$

[IfRcv$_{tt}$] $$\frac{\sigma(cid) = vals.\langle val \rangle}{\langle \mathsf{if\text{-}receive}(cid, var, C_1, C_2), mem, \sigma \rangle \rightarrowtriangle \langle C_1, mem[var \mapsto val], \sigma[cid \mapsto vals] \rangle}$$

sequencing

[Seq$_1$] $$\frac{\langle C_1, mem, \sigma \rangle \rightarrowtriangle \langle \langle\rangle, mem', \sigma' \rangle}{\langle C_1; C_2, mem, \sigma \rangle \rightarrowtriangle \langle C_2, mem', \sigma' \rangle}$$

[Seq$_2$] $$\frac{\langle C_1, mem, \sigma \rangle \rightarrowtriangle \langle C_1'\vec{D}, mem', \sigma' \rangle}{\langle C_1; C_2, mem, \sigma \rangle \rightarrowtriangle \langle (C_1'; C_2)\vec{D}, mem', \sigma' \rangle}$$

Figure 7.2: Small-step deterministic semantics of commands

A $\twoheadrightarrow$-transition is deterministic and either has the form $\langle C, mem, \sigma\rangle \twoheadrightarrow \langle\langle\rangle, mem', \sigma'\rangle$, which means termination of the thread with the final memory mem' and channel status σ', or $\langle C, mem, \sigma\rangle \twoheadrightarrow \langle C'\vec{D}, mem', \sigma'\rangle$, where $\vec{D}$ is possibly empty. Here, one step of computation starting with command C in a memory mem and with channel status σ gives a new main thread C', a (possibly empty) vector $\vec{D}$ of spawned threads, a new memory mem', and a new channel status σ'. For example, the rule [Assign] in Figure 7.2 applies to configurations that incorporate a single command $var := Exp$. According to this rule, the assignment command is removed from the configuration after its execution, the memory is updated ($mem[var \mapsto val]$ maps var to val and maps all other variables like mem), and the channel status remains unchanged. If a configuration incorporates a single command $\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ then either the [If$_{tt}$]- or the [If$_{ff}$]-rule applies. Which of the two rules is applicable depends on whether B evaluates to tt or ff in the current memory. The rule for the command $\mathsf{fork}(C\vec{D})$, where $\vec{D}$ is required to be nonempty, dynamically creates a new vector $\vec{D}$ of threads that, afterwards, run in parallel with the main thread C (sequentially composed with the commands following the fork-command according to rule [Seq$_2$]). This has the effect of adding $\vec{D}$ to the configuration. The rules [Send], [Receive], [IfRcv$_{ff}$], and [IfRcv$_{tt}$] specify the semantics of commands for inter-process communication. Executing a send-command adds a message to a communication channel ($\sigma[cid \mapsto \langle val\rangle.\sigma(cid)]$ maps cid to $\langle val\rangle.\sigma(cid)$ and maps all other channels like σ) and executing a receive-command removes a message. A receive-command is only enabled if the given channel is nonempty. Besides this blocking version, which waits until there is a value on the channel, there also is a non-blocking version, the if-receive-command, which is always enabled. Depending on whether the given channel is empty or not, either the rule [IfRcv$_{ff}$] or the rule [IfRcv$_{tt}$] applies for this command. We assume that communication channels obey the FIFO-principle, i.e. messages sent first are received first.[6]

The rule [Pick] in Figure 7.3 defines the concurrent semantics of DMWL. Whenever the scheduler (nondeterministically) picks a thread C_i in a process for execution, then a $\rightarrow$-transition takes place updating the command, the memory, and the channel status according to a (small) computation step of C_i (i.e. a $\twoheadrightarrow$-transition). It is assumed that the scheduler can decide to schedule a new thread after the execution of every single command.

$$[\text{Pick}] \qquad \frac{\langle C_i, mem, \sigma\rangle \twoheadrightarrow \langle\vec{C}, mem', \sigma'\rangle}{\langle C_1 \ldots C_n, mem, \sigma\rangle \rightarrow \langle C_1 \ldots C_{i-1}\vec{C}C_{i+1} \ldots C_n, mem', \sigma'\rangle}$$

Figure 7.3: Local concurrent semantics of programs

A *distributed program* is a collection $\vec{C}_1, \ldots, \vec{C}_n$ of programs. The operational semantics of distributed DMWL programs is defined in terms of global configurations. A global configuration incorporates a pair $(\vec{C}_i, mem_i)$ for each process. However, there is only a single channel status function for all processes.

Definition 7.2.3 (Global configuration). A *global configuration* is a tuple

$$\lhd(\vec{C}_1, mem_1), \ldots, (\vec{C}_n, mem_n); \sigma\rhd$$

where $(\vec{C}_1, mem_1), \ldots, (\vec{C}_n, mem_n)$ is a finite sequence of command vector/memory pairs and $\sigma : CID \rightarrow VAL^*$ is a finite mapping from channel identifiers to sequences of values. $\diamond$

[6] This assumption is not essential. For example, in [SM02] we did not assume the FIFO-principle.

The rule [Step] in Figure 7.4, defines the global semantics of DMWL programs. This rule ensures that a global transition (a $\twoheadrightarrow$-transition) takes place whenever a local transition (a $\rightarrow$-transition) occurs in some process.

$$[\text{Step}] \qquad \frac{\langle \vec{C}_k, mem_k, \sigma\rangle \rightarrow \langle \vec{C}'_k, mem'_k, \sigma'\rangle}{\begin{array}{l}\lhd(\vec{C}_1, mem_1), \ldots, (\vec{C}_k, mem_k), \ldots, (\vec{C}_n, mem_n); \sigma \rhd \\ \twoheadrightarrow \lhd(\vec{C}_1, mem_1), \ldots, (\vec{C}'_k, mem'_k), \ldots, (\vec{C}_n, mem_n); \sigma' \rhd\end{array}}$$

Figure 7.4: Global concurrent semantics of programs

The reflexive and transitive closures of $\rightarrow$ and $\twoheadrightarrow$ are denoted by $\rightarrow^*$ and $\twoheadrightarrow^*$, respectively.

Remark 7.2.4. Note that the rule [Pick] is possibilistic (involving no probabilities). Although, it is, in general, important to explicitly model the scheduler for addressing flows that result from scheduling policies (possibly probabilistic), there is no reason to introduce explicit schedulers in the semantics here. This is because the security condition that we consider is scheduler independent (demonstrated in [SS00]). $\diamond$

7.2.2 Security Condition for DMWL

The definition of *strong security* is based on the notion of *low-bisimilarity*. A *low-bisimulation* $\sim_L$ is a binary relation between DMWL commands. Intuitively, $C \sim_L D$ expresses that, for the adversary, the behavior of C is indistinguishable from the behavior of D.

Two states are indistinguishable for an adversary if they are equal in their low parts, i.e. if the values of low variables and low channels are identical in both states. In the following definitions, let $level : (VAR \cup CID) \rightarrow \{high, low\}$ be a function that associates security levels with variables and channel identifiers such that $level(h) = high$ and $level(l) = low$. The intuition behind this association is that the values of high variables and the contents of high channels are not visible to the attacker while the values of low variables and the contents of low channels are fully observable.

Definition 7.2.5 (Low-equality on memory). Two mappings $mem_1, mem_2 : VAR \rightarrow VAL$ are *low-equal*, denoted by $mem_1 =_L mem_2$, iff $mem_1(var) = mem_2(var)$ holds for all $var \in VAR$ with $level(var) = low$ (i.e. if $mem_1(l) = mem_2(l)$ holds). $\diamond$

Definition 7.2.6 (Low-equality on channel status). Two mappings $\sigma_1, \sigma_2 : CID \rightarrow VAL^*$ are *low-equal*, denoted by $\sigma_1 =_L \sigma_2$, iff $\sigma_1(cid) = \sigma_2(cid)$ holds for all $cid \in CID$ with $level(cid) = low$. $\diamond$

Two possible behaviors of (nondeterministic) programs are indistinguishable for a low-level observer if all corresponding states in the two behaviors are pairwise low-equal.

Definition 7.2.7 (Low-bisimulation). A *low-bisimulation* $\sim_L \subseteq \vec{CMD} \times \vec{CMD}$ is a symmetric relation between command vectors of equal size such that whenever $\langle C_1 \ldots C_n\rangle \sim_L$

$\langle D_1 \ldots D_n \rangle$ then

$$\begin{aligned}
&\forall mem_1, mem_2, \sigma_1, \sigma_2. \forall \vec{C}', mem_1', \sigma_1'. \forall i \in \{1, \ldots, n\}. \\
&(\langle\!| C_i, mem_1, \sigma_1 |\!\rangle \twoheadrightarrow \langle\!| \vec{C}', mem_1', \sigma_1' |\!\rangle \wedge \ mem_1 =_L mem_2 \wedge \ \sigma_1 =_L \sigma_2) \\
&\Rightarrow \exists \vec{D}', mem_2', \sigma_2'. \\
&\quad (\langle\!| D_i, mem_2, \sigma_2 |\!\rangle \twoheadrightarrow \langle\!| \vec{D}', mem_2', \sigma_2' |\!\rangle \wedge \ mem_1' =_L mem_2' \wedge \sigma_1' =_L \sigma_2' \wedge \vec{C}' \sim_L \vec{D}') \quad \diamond
\end{aligned}$$

Low-bisimilarity requires not only the low-observable parts of corresponding states to be equal but also individual $\twoheadrightarrow$-transitions to correspond. This is the basis for a timing-sensitive and scheduler-independent security condition.

Definition 7.2.8 (Strong low-bisimulation). Define *strong low-bisimulation* $\cong_L \subseteq \vec{CMD} \times \vec{CMD}$ to be the union of all low-bisimulations. $\diamond$

Note that $\cong_L$ is not a reflexive relation. In particular, a program that shows differences in its low-observable behavior for different low-equal starting states is not strongly low-bisimilar to itself. If a program shows differences in its low-observable behavior for states that differ only in values of high variables and high channels then there is a danger of information leakage. This means, a program that is not strongly low-bisimilar to itself may be insecure. This is the intuition underlying the definition of strong security.

Definition 7.2.9 (Strong security). A DMWL program $\vec{C}$ is *strongly secure* if and only if $\vec{C} \cong_L \vec{C}$ holds. $\diamond$

Let us illustrate how strong security can be used to detect dangerous information leakage. In the following example, let σ be some arbitrary channel status function.

Example 7.2.10 (Explicit flow). The program $l := h$ is *not* strongly secure. Choose $mem_1 = (0,0)$ (denotes $mem_1(h) = 0$, $mem_1(l) = 0$) and $mem_2 = (1,0)$ (i.e. $mem_2(h) = 1$, $mem_2(l) = 0$). Since $\langle\!| l := h, (0,0), \sigma |\!\rangle \twoheadrightarrow \langle\!| \langle\rangle, (0,0), \sigma |\!\rangle$ and $\langle\!| l := h, (1,0), \sigma |\!\rangle \twoheadrightarrow \langle\!| \langle\rangle, (1,1), \sigma |\!\rangle$ hold, the resulting memories are not low-equal, i.e. $(0,0) \neq_L (1,1)$ (although the initial memories are low-equal, i.e. $(0,0) =_L (1,0)$). $\diamond$

More subtle examples of programs that are insecure according to Definition 7.2.9 can be found in Appendix E.2. The following example is an instance of a strongly secure program.

Example 7.2.11 (Strongly secure program). The program $\mathsf{if}\ h = 1\ \mathsf{then}\ h := h{+}1\ \mathsf{else}\ \mathsf{skip}$ is strongly secure. Indeed, the timing behavior is independent of the value of h, as well as the low variable l.[7] A suitable symmetric relation that makes this program low-bisimilar to itself is, e.g., the relation

$$\left\{ \begin{array}{l} (\mathsf{if}\ h = 1\ \mathsf{then}\ h := h + 1\ \mathsf{else}\ \mathsf{skip}, \mathsf{if}\ h = 1\ \mathsf{then}\ h := h + 1\ \mathsf{else}\ \mathsf{skip}), \\ (h := h + 1, \mathsf{skip}), \quad (\mathsf{skip}, h := h + 1), \quad (h := h + 1, h := h + 1), \\ (\mathsf{skip}, \mathsf{skip}), \quad (\langle\rangle, \langle\rangle) \end{array} \right\} \quad \diamond$$

For further examples, including algorithms for searching, sorting, and a simple file server, we refer to [SS00, AS01, SM02].

Definition 7.2.9 introduces strong security for programs consisting of a single vector of commands. Such programs are executed in a single (multi-threaded) process. The definition of strong security is extended to the distributed case in a straightforward way.

[7] An implicit assumption underlying the definition of strong security is that differences in the duration of $\twoheadrightarrow$-transitions do not have any impact on the scheduling behavior. This is, e.g., the case if there are no differences in the duration of different $\twoheadrightarrow$-transitions.

Definition 7.2.12 (Strong security for distributed programs). A distributed program, i.e. a collection $\vec{C}_1, \ldots, \vec{C}_n$ of programs, is *strongly secure* if for all $i \in \{1, \ldots, n\}$ the program C_i is strongly secure. ◇

To justify that Definition 7.2.12 is, indeed, a sensible definition of security for distributed programs is one of the main objectives of our case study.

7.2.3 Security Type System for DMWL

A security type system for program analysis of DMWL programs is presented in Figure 7.5. According to the typing rules, any expression may be typed *high*. On the other hand, only expressions that have no occurrences of h are typed *low* (i.e. expressions that can be safely used in assignments to l).

The type of a (secure) DMWL program $\vec{C}$ (also referred to as its *low slice*) is a DMWL program that models the timing behavior of $\vec{C}$ and in which no high variables occur. A type check of a given program either fails (program is insecure), succeeds directly (program is secure), or succeeds after transforming the program (into a secure one). The transformation rules in Figure 7.5 have the form $\vec{C} \hookrightarrow \vec{C}' : \vec{Sl}$, where $\vec{C}$ is a program, $\vec{C}'$ is the result of its transformation, and $\vec{Sl}$ is the low slice of $\vec{C}'$. The objective of transforming a program by these rules is to make an insecure program strongly secure without changing its behavior in any essential way (assignment of values to variables and communication commands remain unaffected). More precisely, the transformation is capable of eliminating certain timing leaks.

For example, according to rule [Set_{low}] in Figure 7.5, a command $l := Exp$ can be type checked (for type $l := Exp$) under the condition that no high variables occur in Exp. Otherwise, the program is rejected as insecure (identifying a potential explicit flow). In rule [If_{high}], the effect of the transformation can be observed. For the security of a conditional with a high guard, it is essential that both branches make the same assignments to low variables/low channels and that their timing behavior is identical. In order to enforce identical timing behavior, the transformation sequentially composes the program in each branch with the low slice of the respective other branch. Namely, C_1' is sequentially composed with the low slice Sl_2 of C_2' in the if-branch and the low slice Sl_1 of C_1' is sequentially composed with C_2' in the else-branch (cf. rule [If_{high}]). Cross copying the low slices results in identical timing behavior of both branches. The condition $al(Sl_1) = al(Sl_2) = ff$ on the low slices that are cross copied ensures that no assignments to low variables and no communications on low channels are performed (preventing the introduction of implicit leaks in a conservative way). In order to avoid that the transformation introduces differences in the behavior on high variables, we employ the notation $v\hat{a}r$ (where $\hat{l} = l$ and $\hat{h} = \_$) to indicate the case that h is not updated after a reception (in rules [R_{low}], [IR_{low}]). Type checking of composed programs is purely compositional (cf. rules [Seq] and [Par]).

A successful type check of a given program in the presented security type system implies that the (possibly transformed) program is strongly secure, i.e. the type system is sound in this respect. The following soundness theorem is a specialization of a result in [SM02].

Theorem 7.2.13 ([SM02]). If $\vec{C} \hookrightarrow \vec{C}' : \vec{Sl}$ is derivable then $\vec{C}'$ is strongly secure. ◇

In this section, we have briefly introduced the preliminaries for our case study. Now we are ready to move to the core of our case study in the next section.

expressions

[Exp] $Exp : high \qquad \dfrac{h \notin Vars(Exp)}{Exp : low}$

commands with local effects

[Skip] $\mathsf{skip} \hookrightarrow \mathsf{skip} : \mathsf{skip}$

[Set$_{low}$] $\dfrac{Exp : low}{l := Exp \hookrightarrow l := Exp : l := Exp}$

[Set$_{high}$] $h := Exp \hookrightarrow h := Exp : \mathsf{skip}$

[If$_{low}$] $\dfrac{B : low \qquad C_1 \hookrightarrow C_1' : Sl_1 \qquad C_2 \hookrightarrow C_2' : Sl_2}{\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \hookrightarrow \mathsf{if}\ B\ \mathsf{then}\ C_1'\ \mathsf{else}\ C_2' : \mathsf{if}\ B\ \mathsf{then}\ Sl_1\ \mathsf{else}\ Sl_2}$

[If$_{high}$] $\dfrac{B : high \qquad C_1 \hookrightarrow C_1' : Sl_1 \qquad C_2 \hookrightarrow C_2' : Sl_2 \qquad al(Sl_1) = al(Sl_2) = \mathit{ff}}{\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \hookrightarrow \mathsf{if}\ B\ \mathsf{then}\ C_1'; Sl_2\ \mathsf{else}\ Sl_1; C_2' : \mathsf{skip}; Sl_1; Sl_2}$

[While] $\dfrac{B : low \qquad C \hookrightarrow C' : Sl}{\mathsf{while}\ B\ \mathsf{do}\ C \hookrightarrow \mathsf{while}\ B\ \mathsf{do}\ C' : \mathsf{while}\ B\ \mathsf{do}\ Sl}$

[Fork] $\dfrac{C_1 \hookrightarrow C_1' : Sl_1 \quad \vec{C_2} \hookrightarrow \vec{C_2'} : \vec{Sl_2}}{\mathsf{fork}(C_1\vec{C_2}) \hookrightarrow \mathsf{fork}(C_1'\vec{C_2'}) : \mathsf{fork}(Sl_1\vec{Sl_2})}$

communication commands where *cid* is *high*

[S$_{high}$] $\mathsf{send}(cid, Exp) \hookrightarrow \mathsf{send}(cid, Exp) : \mathsf{skip}$

[IR$_{high}$] $\dfrac{C_1 \hookrightarrow C_1' : Sl_1 \qquad C_2 \hookrightarrow C_2' : Sl_2 \quad al(Sl_1) = al(Sl_2) = \mathit{ff}}{\mathsf{if\text{-}receive}(cid, h, C_1, C_2) \hookrightarrow \mathsf{if\text{-}receive}(cid, h, C_1'; Sl_2, Sl_1; C_2') : \mathsf{skip}; Sl_1; Sl_2}$

communication commands where *cid* is *low*

[S$_{low}$] $\dfrac{Exp : low}{\mathsf{send}(cid, Exp) \hookrightarrow \mathsf{send}(cid, Exp) : \mathsf{send}(cid, Exp)}$

[R$_{low}$] $\mathsf{receive}(cid, var) \hookrightarrow \mathsf{receive}(cid, var) : \mathsf{receive}(cid, \hat{var})$

[IR$_{low}$] $\dfrac{C_1 \hookrightarrow C_1' : Sl_1 \qquad C_2 \hookrightarrow C_2' : Sl_2}{\mathsf{if\text{-}receive}(cid, var, C_1, C_2) \hookrightarrow \mathsf{if\text{-}receive}(cid, var, C_1', C_2') : \mathsf{if\text{-}receive}(cid, \hat{var}, Sl_1, Sl_2)}$

composed programs

[Seq] $\dfrac{C_1 \hookrightarrow C_1' : Sl_1 \qquad C_2 \hookrightarrow C_2' : Sl_2}{C_1; C_2 \hookrightarrow C_1'; C_2' : Sl_1; Sl_2}$

[Par] $\dfrac{C_1 \hookrightarrow C_1' : Sl_1 \ \ldots \ C_n \hookrightarrow C_n' : Sl_n}{\langle C_1 \ldots C_n \rangle \hookrightarrow \langle C_1' \ldots C_n' \rangle : \langle Sl_1 \ldots Sl_n \rangle}$

$al(Sl_1)$ is true whenever there is a syntactic occurrence of either an assignment to l or a communication primitive on low channels in the low slice Sl_1.

Figure 7.5: Security typing of expressions and commands

7.3 Specifying System Behavior

We first briefly outline how the behavior of a system can be modeled by an event-based specification. Then we illustrate in detail how this can be done for our example. More precisely, we specify the behavior of DMWL processes by state-event systems.

7.3.1 How to Proceed

1. Select a formalism for specifying the system. If you model the system with event systems then proceed with 2. Otherwise, proceed with 3.

2. Specifying a system by an event system:

   (a) Model the interface of the system by two disjoint sets I and O of input events and output events, respectively.

   (b) Determine the set of all other events the system can engage in, i.e. the internal events. Define E to be the union of I, O, and this set of internal events.

   (c) Model the possible behaviors of the system under consideration by a set $Tr \subseteq E^*$ of traces. The set Tr must be non-empty and must be closed under prefixes.

3. Specifying system behavior with some other specification formalism:

   (a) Define a suitable mapping from the chosen specification formalism to event systems.

   (b) Specify the system in the specification formalism. The event system corresponding to this specification can be obtained by applying the previously defined mapping.

7.3.2 The Running Example

We follow the second approach in our case study, i.e. we perform steps 1 and 3(a,b).

Specification Formalism. We model the operational semantics of DMWL by a state-event system (cf. Definition 2.1.13).

Mapping into Event Systems. We use the mapping from state-event systems to event systems as specified in Definition 2.1.18.

System Specification. We proceed as follows: firstly, we model the *interface*, secondly, we model the *internal actions*, thirdly, we define the *state space*, fourthly, we specify the *initial state*, and finally, we model the *possible behaviors* by a transition relation. Each of these steps is performed for single processes and then it is explained how distributed systems with multiple processes are specified by composing specifications of individual processes. The resulting specification is an adequate model of DMWL programs as is shown in Appendix E.3.

Types used in the specification are summarized in Figure 7.6.

abbreviation	description	comment
PID	set of process identifiers	not further specified
TID	set of thread identifiers	$TID = \mathbb{N}^*$
CID	set of channel identifiers	not further specified
VAR	set of variables	for simplicity: $VAR = \{h, l\}$ is assumed
VAL	set of values	not further specified
THREAD	state of a thread	$THREAD = CMD$
INFO	scheduling information	$INFO = (\mathbb{N} \cup \{-1\}) \times VAL \times \mathcal{P}(TID)$

Figure 7.6: Types used in the specification

Interface Specification. The interface of a single DMWL process is shown in Figure 7.7. The events *schedule* and *yield* model the actions of the scheduler. Since the scheduler is outside the language definition of DMWL (as usual for programming languages), the specification of the precise scheduler is not part of the process specification. An event $schedule_p(tid)$ models that the (external) scheduler selects the thread with identifier $tid \in TID$ for execution in process p. An event $yield_p(info)$ models that an active thread that has used up its time slice gives up control of the process. The parameter $info \in INFO$ models information that is passed to the scheduler for determining the next thread. It includes information about the status of the previously active thread (terminated/not terminated/number of threads spawned), the value of the low variable, and information about which threads are blocked because they are waiting for a message on an empty incoming channel. This information is expected by the class of schedulers that is covered by the scheduler-independent program analysis for DMWL.

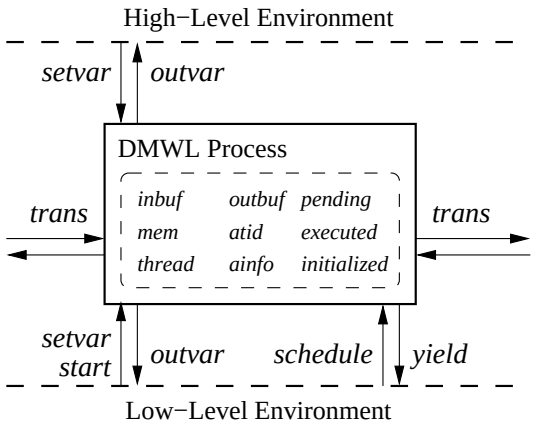


Figure 7.7: DMWL process with interface events and state objects

In DMWL, the value of the low variable l can be observed by low-level adversaries while h is only observable by the high-level environment. This means, l is part of the interface to the low-level environment and h is part of the interface to the high-level environment. In an event-based setting, the possibility to observe values of variables is explicitly modeled by corresponding interface events because the interface consists of events (and not of variables). Therefore, we have to introduce interface events that model the observation or modification of the variables l and h. An event $outvar_p(var, val)$ models that it is observed that variable *var* of

process p has value *val*. In a similar fashion, setting the (initial) value of variables is explicitly modeled by events $setvar_p(var, val)$. Values of variables can only be altered by *setvar*-events before program execution begins (*setvar*-events model the initialization of variables). The event *start* models that the initialization phase has been completed and program execution begins. After *start* has occurred, *setvar*-events are disabled.

Inter-process communication is performed in DMWL via communication channels. An event $trans(cid, val)$ models the transmission of a value *val* on a channel $cid \in CID$. Since *trans*-events belong to more than one process, they are not tagged with a process identifier. Throughout this chapter, we assume that a process cannot send messages to itself directly (intra-process communication occurs via shared variables), i.e. $\forall cid \in CID.\, sender(cid) \neq receiver(cid)$ holds where the functions $sender, receiver : CID \to PID$ model which processes may communicate on a communication channel. For a given channel *cid*, $sender(cid)$ is the identifier of the process that may send on *cid* and $receiver(cid)$ is the identifier of the process that may receive on *cid*. This means, $\{cid \mid sender(cid) = p\}$ and $\{cid \mid receiver(cid) = p\}$ are the sets of channels that the process p may use for sending and receiving, respectively.

The interface specification is summarized in Figure 7.8 where, for improving readability, the index p is omitted for all events.

$$I^p = \left\{ \begin{array}{l|l} schedule(tid), & tid : TID, \\ setvar(var, val), & var : VAR, val : VAL, \\ start, & cid : CID, \\ trans(cid, val) & receiver(cid) = p \end{array} \right\}$$

$$O^p = \left\{ \begin{array}{l|l} yield(info), & info : INFO, \\ outvar(var, val), & var : VAR, val : VAL, \\ trans(cid, val) & cid : CID, \\ & sender(cid) = p \end{array} \right\}$$

Figure 7.8: Interface specification of a DMWL process

Specification of Internal Actions. For every rule of the operational semantics of DMWL (cf. Figure 7.2) that reduces a command, we specify a corresponding event. For example, the event $skip_p$ models that a command skip is being executed by the active thread in process p (corresponding to rule [Skip]). The event $assign_p(var, val)$ models that a command $var := Exp$ is being executed by the active thread in process p where *Exp* evaluates to *val* under the current memory of the process (corresponding to rule [Assign]). The event $ite_p^{tt}(B, C_1, C_2)$ models that a command if B then C_1 else C_2 is being executed by the active thread in process p where B evaluates to tt under the current memory (corresponding to rule $[\mathrm{If}_{tt}]$). The set E^p_{local} of all events that model the execution of DMWL commands in process p is given in Figure 7.9 where, for improving readability, the index p is omitted for all events.

Definition 7.3.1. The set of events for a DMWL-thread pool with identifier p is specified by $E^p = I^p \cup O^p \cup E^p_{local}$ (cf. Figures 7.8, 7.9 for definitions of I^p, O^p, and E^p_{local}). ◇

Specification of State Space. The memory in the local configuration of a process with identifier p is modeled by a (functional) state object $mem^p : VAR \to VAL$. For a given

$$E^p_{local} = \left\{ \begin{array}{l|l} skip, \quad assign(var, val), & var : VAR, \quad val : VAL, \\ ite^{tt}(B, C_1, C_2), \quad ite^{ff}(B, C_1, C_2), & B : BOOL, \\ while^{tt}(B, C), \quad while^{ff}(B, C), & C, C_1, C_2 : CMD, \\ fork(C, \vec{D}), & \vec{D} : C\vec{M}D, \\ send(cid, val), & cid, cid' \in CID, \\ receive(cid', var, val), & C_1, C_2 \in CMD, \\ ite\text{-}rcv^{tt}(cid', var, val, C_1, C_2), & sender(cid) = p, \\ ite\text{-}rcv^{ff}(cid', var, val, C_1, C_2) & receiver(cid') = p \end{array} \right\}$$

Figure 7.9: Specification of internal events for a DMWL process

program variable *var*, $mem^p(var)$ denotes the current value of *var*. The command vector in the local configuration of a process is modeled by a functional state object $thread^p : TID \to (THREAD \cup \{\bot, \top, \langle\rangle\})$. For a given thread *tid*, $thread^p(tid)$ denotes the command to be executed by this thread (if $thread^p(tid) \in THREAD$). The values $\bot$, $\top$, $\langle\rangle$ have a special meaning. Here (and below) $\bot$ stands for "undefined". The value $\top$ models that a thread with identifier *tid* has performed a fork command at some point in the past. This is a technical peculiarity of our assignments of thread identifiers: after a thread with identifier *tid* has performed a fork this thread continues execution with the new identifier $tid.\langle 0\rangle$ and the spawned threads are assigned identifier $tid.\langle 1\rangle$, $tid.\langle 2\rangle$, ..., respectively. The value $\langle\rangle$ models that the thread has terminated at some point in the past.

The state objects $inbuf^p$, $pending^p$, and $outbuf^p$ model (parts of) the contents of communication channels that are adjacent to the process. More specifically, $inbuf^p, pending^p : CID \to VAL^*$ are functional state objects where $inbuf^p(cid)$ models the sequence of messages that have been received over channel *cid* but which are not yet ready for processing and $pending^p(cid)$ models the sequence of messages that have been received over channel *cid* and that are ready for processing. The state object $outbuf^p : (CID \times VAL) \cup \{\langle\rangle\}$ stores outgoing messages until they have been sent over a communication channel. The value $\langle\rangle$ models that the output buffer is empty and a value (cid, val) models that message *val* shall be sent over channel *cid*. There is at most one message in the output buffer of a given process.

The state objects $atid^p : TID \cup \{\bot\}$, $ainfo^p : \mathbb{N} \cup \{-1\} \cup \{\bot\}$, $executed^p : BOOL$, and $initialized^p : BOOL$ store status information. The identifier of the currently active thread is stored in $atid^p$ (if there is any — otherwise: undefined). Information that shall be passed to the scheduler as parameter of the next *yield*-event is stored in $ainfo^p$ (if there is any). That the active thread has terminated is indicated by $ainfo^p = -1$. Otherwise, $ainfo^p$ equals the number of threads spawned after execution of a DMWL-command. The flag $executed^p$ indicates whether the currently active thread has already performed a step (value *tt*) or not (value *ff*). The flag $initialized^p$ indicates whether the initialization of variables has been finished (value *tt*) or not (value *ff*).

The specification of the state space is summarized in Figure 7.10 where, for improving readability, the index p is omitted for all state objects.

Specification of the Initial State. For the initial state $s^p_0(initthread)$, we assume that all program variables are initialized with some (not further specified) value *initval*. The thread with identifier $\langle 0\rangle$ contains the program *initthread* that is to be executed, i.e. $thread^p_{s_0}(\langle 0\rangle) = initthread$. For all other thread identifiers, $thread^p_{s_0}$ returns $\bot$. Initially, all communication

$$S^p = \{(mem, thread, inbuf, pending, outbuf, atid, ainfo, executed, initialized)\}$$

mem	:	$VAR \to VAL$	, current local shared memory
$thread$	:	$TID \to THREAD \cup \{\bot, \top, \langle\rangle\}$	, current local state of threads
$inbuf$	:	$CID \to VAL^*$	, buffer for incoming messages
$pending$	:	$CID \to VAL^*$	, buffer for incoming messages
$outbuf$	:	$(CID \times VAL) \cup \{\langle\rangle\}$	, buffer for outgoing messages
$atid$	:	$TID \cup \{\bot\}$	, identifier of active thread
$ainfo$	:	$\mathbb{N} \cup \{-1\} \cup \{\bot\}$	, actual scheduler information
$executed$	:	$BOOL$	, has just a step been executed?
$initialized$	:	$BOOL$	, has the initialization been finished?

Figure 7.10: Specification of state space for a DMWL process

buffers ($inbuf^p_{s_0}$, $pending^p_{s_0}$, $outbuf^p_{s_0}$) are empty, no thread is active, there is no scheduler information, and the flags $executed^p_{s_0}$ and $initialized^p_{s_0}$ are set to *ff*.

The specification of the initial state $s^p_0(initthread)$ for a process with identifier p is given in Figure 7.11 where, for better readability, the index p is omitted for state objects.

$$s^p_0(initthread) = (mem_{s_0}, thread_{s_0}, inbuf_{s_0}, pending_{s_0}, outbuf_{s_0}, atid_{s_0}, ainfo_{s_0}, executed_{s_0}, initialized_{s_0})$$

$mem_{s_0}(var) = initval$, for all $var \in VAR$
$thread_{s_0}(\langle 0\rangle) = initthread$
$thread_{s_0}(tid) = \bot$, for all $tid \in TID$ with $tid \neq \langle 0\rangle$
$inbuf_{s_0}(cid) = \langle\rangle$, for all $cid \in CID$
$pending_{s_0}(cid) = \langle\rangle$, for all $cid \in CID$
$outbuf_{s_0} = \langle\rangle$
$atid_{s_0} = \bot$,
$ainfo_{s_0} = \bot$,
$executed_{s_0} = ff$,
$initialized_{s_0} = ff$

Figure 7.11: Specification of initial state of a DMWL process

Specification of System Behavior. In the following, we omit the index p when it can be concluded from the context. During the initialization phase (indicated by $initialized = ff$), the values of variables may be changed arbitrarily by *setvar*-events. Threads are not executed in this phase (*schedule*-events are disabled). The initialization phase is completed by the occurrence of the event *start*. After the initialization, *setvar*-events are disabled and thread execution proceeds as follows:

- If no thread is active (indicated by $atid = \bot$) then *schedule*-events are enabled. After an occurrence of $schedule(tid)$, $atid$ is set to tid, the thread with local state $thread(tid)$ becomes active, and all incoming messages are prepared for processing (moving them from $inbuf$ to $pending$). An event $schedule(tid)$ is only enabled if the thread tid is alive

($thread(tid) \notin \{\bot, \top, \langle\rangle\}$) and not blocked (i.e. trying to receive on an empty channel).

- If there is an active thread (indicated by $atid \neq \bot \wedge executed = ff$) then this thread can run. Thread execution is formally modeled by the occurrence of events that are internal to the thread pool. During execution, a thread affects the state objects *mem*, *thread*, *pending*, and *outbuf* depending on the particular command that is executed. Additionally, status information for the scheduler is stored in *ainfo*. Eventually, the active thread stops executing (indicated by $executed = tt$).

- After the active thread has stopped ($executed = tt$), any outgoing message in the output buffer is sent on the respective channel.

- After the active thread has stopped and the output buffer is empty, the scheduler can be informed about this by a *yield*-event. The event $yield(info)$ is only enabled if *info* corresponds to the actual scheduler information combined with the value of the low variable and information about which threads are blocked on empty channels (i.e. $info = (ainfo, mem(l), blocked\text{-}set)$). A *yield*-event resets the *executed*-flag, *atid*, and *ainfo*.

For the formal specification of the transition relation (in Figures 7.12, 7.13, and 7.14), we employ a notation based on preconditions and postconditions. In brief, a transition $(s, e, s') \in S \times E \times S$ complies with a pre/postcondition statement (abbreviated by PP-statement in the following) if all variables not mentioned in the affects-slot have the same value in s and in s' (frame axioms), the precondition holds for the values of variables given by s, and the postcondition holds for the values of unprimed and primed variables given by s and s', respectively.

Let us explain this notation in more detail with the example of the PP-statement for *schedule*-events in Figure 7.12. According to this PP-statement, an event $schedule_p(tid)$ leaves all state variables except for $atid^p$, $inbuf^p$, $pending^p$ unchanged. The precondition for this event is that no thread is currently active ($atid = \bot$), that a thread with identifier *tid* is defined ($thread(tid) \notin \{\bot, \top, \langle\rangle\}$), that if the first command of this thread is a blocking receive then the corresponding channel is nonempty, and that the initialization phase has been finished. The postcondition ensures that *tid* becomes the new active thread ($atid^{p\prime} = tid$) and that, for all channels, the messages received are prepared for processing (moving them from $inbuf^p$ to $pending^p$).

A more detailed explanation of PP-statements, including a definition of the formal semantics via a translation into second-order formulas, is contained in Appendix E.1.

The behavior of interface events is specified in Figure 7.12, the behavior of local computation events in Figure 7.13, and the behavior of local communication events in Figure 7.14.

Definition 7.3.2. Let $T^p \subseteq S^p \times E^p \times S^p$ be a transition relation such that $(s, e, s') \in T$ if and only if (s, e, s') complies with all PP-statements for e in Figures 7.12, 7.13, and 7.14. ◇

Process Specification. The overall specification of a DMWL process by a state-event system is summarized in the following definition.

Definition 7.3.3 (DMWL process). Let $p \in P$ and $initthread \in CMD$. We define

$$DMWLProcess(p, initthread) = (S^p, s_0^p(initthread), E^p, I^p, O^p, T^p) \qquad \diamond$$

scheduling events

***schedule*(*tid*)** affects *atid*, *inbuf*, *pending*
Pre : $atid = \bot \wedge thread(tid) \notin \{\bot, \top, \langle\rangle\} \wedge tid \notin blocked\text{-}set$
$\wedge\ initialized = tt$
Post: $atid' = tid \wedge \forall cid : CID.\ inbuf'(cid) = \langle\rangle$
$\wedge\ \forall cid : CID.\ pending'(cid) = inbuf(cid).pending(cid)$

***yield*(*info*)** affects *executed*, *atid*, *ainfo*
Pre : $executed = tt \wedge info = (ainfo, mem(l), blocked\text{-}set) \wedge outbuf = \langle\rangle$
Post: $executed' = ff \wedge atid' = \bot \wedge ainfo' = \bot$

variable accesses/initialization

***outvar*(*var*, *val*)** affects —
Pre : $mem(var) = val$
Post: tt

***setvar*(*var*, *val*)** affects $mem(var)$
Pre : $initialized = ff$
Post: $mem'(var) = val$

start affects *initialized*
Pre : $initialized = ff$
Post: $initialized' = tt$

message transmissions

***trans*(*cid*, *val*)** (case $receiver(cid) = p$) affects $inbuf(cid)$
Pre : tt
Post: $inbuf'(cid) = \langle val\rangle.inbuf(cid)$

***trans*(*cid*, *val*)** (case $sender(cid) = p$) affects *outbuf*
Pre : $outbuf = (cid, val)$
Post: $outbuf' = \langle\rangle$

where $blocked\text{-}set = \{tid \mid first(thread(tid)) = \mathsf{receive}(cid, var) \wedge pending(cid) = \langle\rangle\}$

Figure 7.12: Preconditions and postconditions of interface events

Definition 7.3.4 (*sender*,*receiver*). A process $DMWLProcess(p, initthread)$ *complies with the functions* $sender, receiver : CID \to PID$ if for every command $\mathsf{send}(cid, Exp)$, which is a subcommand of *initthread*, holds $sender(cid) = p$ and for every command $\mathsf{receive}(cid, var)$ or $\mathsf{if\text{-}receive}(cid, var, C_1, C_2)$, which is a subcommand of *initthread*, holds $receiver(cid) = p$. ◇

In the following, we implicitly assume all DMWL processes to comply with *sender* and *receiver*. Moreover, we assume $sender(cid) \neq receiver(cid)$ for all $cid \in CID$.

Specification of Distributed Programs. Distributed programs can be specified by composing process specifications (cf. Figure 7.15). The composition of state-event systems is defined along the same lines as the composition of event systems.

Definition 7.3.5 (Composable). Two state-event systems $SES_1 = (S_1, s_{0_1}, E_1, I_1, O_1, T_1)$ and $SES_2 = (S_2, s_{0_2}, E_2, I_2, O_2, T_2)$ are *composable* iff $E_1 \cap E_2 \subseteq (O_1 \cap I_2) \cup (O_2 \cap I_1)$ holds. ◇

computation events

skip affects $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge first(thread(atid)) = \mathsf{skip}$
Post: $thread'(atid) = rest(thread(atid)) \wedge done$
$\wedge ainfo' = terminates(thread(atid))$

assign(var, val) affects $mem(var)$, $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge Exp \downarrow^{mem} val \wedge first(thread(atid)) = var := Exp$
Post: $mem'(var) = val \wedge thread'(atid) = rest(thread(atid)) \wedge done$
$\wedge ainfo' = terminates(thread(atid))$

ite$^{tt}(B, C_1, C_2)$ affects $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge B \downarrow^{mem} tt \wedge first(thread(atid)) = \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$
Post: $thread'(atid) = C_1; rest(thread(atid)) \wedge done \wedge ainfo' = 0$

ite$^{ff}(B, C_1, C_2)$ affects $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge B \downarrow^{mem} ff \wedge first(thread(atid)) = \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$
Post: $thread'(atid) = C_2; rest(thread(atid)) \wedge done \wedge ainfo' = 0$

while$^{tt}(B, C_1)$ affects $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge B \downarrow^{mem} tt \wedge first(thread(atid)) = \mathsf{while}\ B\ \mathsf{do}\ C_1$
Post: $thread'(atid) = C_1; \mathsf{while}\ B\ \mathsf{do}\ C_1; rest(thread(atid)) \wedge done \wedge ainfo' = 0$

while$^{ff}(B, C_1)$ affects $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge B \downarrow^{mem} ff \wedge first(thread(atid)) = \mathsf{while}\ B\ \mathsf{do}\ C_1$
Post: $thread'(atid) = rest(thread(atid)) \wedge done$
$\wedge ainfo' = terminates(thread(atid))$

fork$(C, D_1 \ldots D_n)$ affects $thread(atid)$, $thread(atid.\langle 0\rangle) \ldots thread(atid.\langle n\rangle)$,
$executed$, $ainfo$
Pre : $ready \wedge first(thread(atid)) = \mathsf{fork}(C D_1 \ldots D_n)$
Post: $thread'(atid) = \top \wedge thread'(atid.\langle 0\rangle) = C; rest(thread(atid))$
$\wedge \forall i : \{1, \ldots, n\}.\ thread'(atid.\langle i\rangle) = D_i \wedge done \wedge ainfo' = n$

where
$first(thread(atid))$ denotes the first command in $thread(atid)$,
$rest(thread(atid))$ results from $thread(atid)$ by removing the first command,
and the following abbreviations are used:
$ready \Leftrightarrow (executed = ff \wedge atid \neq \bot)$
$done \Leftrightarrow (executed' = tt)$
$terminates(thread(atid))$ equals -1 if $rest(thread(atid)) = \langle\rangle$ and 0 otherwise.

Figure 7.13: Preconditions and postconditions of local events (1)

communication events

***send*(*cid*, *val*)** affects $thread(atid)$, $executed$, $ainfo$, $outbuf$
Pre : $ready \wedge Exp \downarrow^{mem} val \wedge first(thread(atid)) = \mathsf{send}(cid, Exp)$
Post: $outbuf' = (cid, val) \wedge thread'(atid) = rest(thread(atid)) \wedge done$
$\wedge ainfo' = terminates(thread(atid))$

***receive*(*cid*, *var*, *val*)** affects $thread(atid)$, $executed$, $ainfo$, $mem(var)$, $pending(cid)$
Pre : $ready \wedge pending(cid) \neq \langle\rangle \wedge last(pending(cid)) = val$
$\wedge first(thread(atid)) = \mathsf{receive}(cid, var)$
Post: $mem'(var) = val \wedge pending'(cid) = butlast(pending(cid))$
$\wedge thread'(atid) = rest(thread(atid)) \wedge done \wedge ainfo' = terminates(thread(atid))$

***ite-rcv*^{tt}(*cid*, *var*, *val*, C_1, C_2)** affects $thread(atid)$, $executed$, $ainfo$, $mem(var)$, $pending(cid)$
Pre : $ready \wedge pending(cid) \neq \langle\rangle \wedge last(pending(cid)) = val$
$\wedge first(thread(atid)) = \mathsf{if\text{-}receive}(cid, var, C_1, C_2)$
Post: $mem'(var) = val \wedge pending'(cid) = butlast(pending(cid))$
$\wedge thread'(atid) = C_1; rest(thread(atid)) \wedge done \wedge ainfo' = 0$

***ite-rcv*^{ff}(*cid*, *var*, *val*, C_1, C_2)** affects $thread(atid)$, $executed$, $ainfo$
Pre : $ready \wedge pending(cid) = \langle\rangle \wedge first(thread(atid)) = \mathsf{if\text{-}receive}(cid, var, C_1, C_2)$
Post: $thread'(atid) = C_2; rest(thread(atid)) \wedge done \wedge ainfo' = 0$

where
$first(thread(atid))$ denotes the first command in $thread(atid)$,
$rest(thread(atid))$ results from $thread(atid)$ by removing the first command,
$last(pending(cid))$ denotes the last message in $pending(cid)$,
$butlast(pending(cid))$ results from $pending(cid)$ by removing the last message,
and the following abbreviations are used:
$ready \Leftrightarrow (executed = ff \wedge atid \neq \bot)$
$done \Leftrightarrow (executed' = tt)$
$terminates(thread(atid))$ equals -1 if $rest(thread(atid)) = \langle\rangle$ and 0 otherwise.

Figure 7.14: Preconditions and postconditions of local events (2)

Definition 7.3.6 (Composition state-event systems). Let $SES_1 = (S_1, s_{0_1}, E_1, I_1, O_1, T_1)$ and $SES_2 = (S_2, s_{0_2}, E_2, I_2, O_2, T_2)$ be composable.

The *composition* of SES_1 and SES_2 (denoted by $SES_1 \parallel SES_2$) is the state-event system $SES = (S, s_0, E, I, O, T)$ where S, s_0, E, I, O, and T are defined as follows:

$$
\begin{array}{lcl}
S & = & S_1 \times S_2 \\
s_0 & = & (s_{0_1}, s_{0_2}) \\
E & = & E_1 \cup E_2 \\
I & = & (I_1 \setminus O_2) \cup (I_2 \setminus O_1) \\
O & = & (O_1 \setminus I_2) \cup (O_2 \setminus I_1) \\
T & = & \{((s_1, s_2), e, (s_1', s_2')) \in S \times E \times S \\
 & & \quad |((e \notin E_1 \wedge s_1' = s_1) \vee (e \in E_1 \wedge (s_1, e, s_1') \in T_1)) \\
 & & \quad\ \wedge ((e \notin E_2 \wedge s_2' = s_2) \vee (e \in E_2 \wedge (s_2, e, s_2') \in T_2))\}
\end{array}
$$

$\diamond$

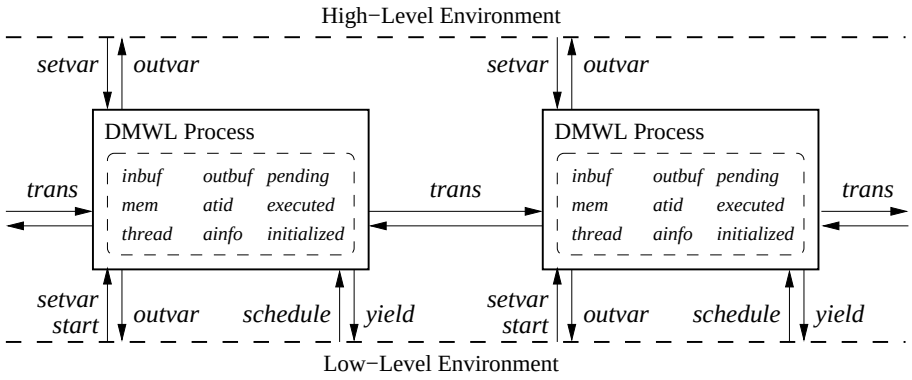


Figure 7.15: Composition of processes with interface events and state objects

The following theorem shows that our definition of a composition operator for state-event system is compatible with the notion of an induced event system.

Theorem 7.3.7. Let $SES_1 = (S_1, s_{0_1}, E_1, I_1, O_1, T_1)$ and $SES_2 = (S_2, s_{0_2}, E_2, I_2, O_2, T_2)$ be composable. Then $ES_{SES_1 \| SES_2} = ES_{SES_1} \parallel ES_{SES_2}$ holds. ◇

Proof. Follows from Definitions 2.1.18, 6.2.2, and 7.3.6. □

Note in Definition 7.3.6, that the state space of the composed system is the *product* of the component's state spaces. This means that state-event systems do *not* communicate via their state space (e.g. by shared variables). They communicate, like event systems, by synchronizing on occurrences of shared events. Hence, restricted forms of composition (product, proper cascade, and general cascade) could be defined just like for event systems.

We are now ready to present the event-based specification of distributed DMWL programs.

Definition 7.3.8 (DMWL process pool). Let $p_1, \ldots, p_n \in PID$ be a collection of process identifiers and $initthread_1, \ldots, initthread_n \in CMD$ be a collection of DMWL commands. We define

$$DMWLProcPool((p_1, initthread_1), \ldots, (p_n, initthread_n))$$
$$= DMWLProcess(p_1, initthread_1) \parallel \ldots \parallel DMWLProcess(p_n, initthread_n)$$ ◇

The state-event system specified in Definition 7.3.8 is an adequate model of DMWL process pools. This is demonstrated formally in Appendix E.3.

In Definitions 7.3.3 and 7.3.8, we have presented event-based system specifications of DMWL programs (for the non-distributed case and the distributed case, respectively). We are now ready to analyze the security of DMWL programs based on these specifications.

7.4 Specifying Security Requirements

We shall now outline how one can specify the security requirements of a given system with information flow properties in *MAKS*, and then we illustrate in detail how this can be done for our example. More precisely, we formalize the security requirements of individual DMWL processes by defining a set of views and by assembling a security predicate from appropriate BSPs.

7.4.1 How to Proceed

1. Determine all confidentiality requirements for the given system.
2. Determine all integrity requirements for the given system.
3. Select a formalism for specifying the restrictions on the information flow. If you specify the restrictions directly by a set of views then proceed with 3(a). If you use flow policies for this purpose (cf. Section 3.3.1), proceed with 3(b).
   (a) Define a set VS of views to express the security requirements for the given system and then proceed with step 4.
   (b) Define a flow policy that expresses the security requirements for the given system abstractly and specialize these abstract requirements by defining a domain assignment. The set VS of views can be calculated according to Definition 3.3.11.
4. For each view $\mathcal{V} \in VS$, define a suitable security predicate SP:
   (a) Decide if deductions about *occurrences* of confidential events must be prevented. If yes, choose one or more BSPs from the first dimension (i.e. from R, D, BSD, SR, SD, FCD^{Γ}). Justify that this choice is appropriate. In particular, ensure that the typical pitfalls (illustrated for the various BSPs in the respective subsections of Section 3.4) are avoided.
   (b) Decide if deductions about *nonoccurrences* of confidential events must be prevented. If yes, choose one or more BSPs from the second dimension (i.e. from I, IA^{ρ}, BSI, $BSIA^{\rho}$, SI, SIA^{ρ}, FCI^{Γ}, $FCIA^{\rho,\Gamma}$). Justify that this choice is appropriate. In particular, ensure that the typical pitfalls (illustrated for the various BSPs in the respective subsections of Section 3.4) are avoided. If one of IA^{ρ}, $BSIA^{\rho}$, SIA^{ρ}, $FCIA^{\rho,\Gamma}$ is chosen, then choose an admissibility function ρ and justify that ρ is appropriate (in particular, avoid the pitfalls pointed out in Remark 3.4.21).
   (c) Define SP to be the conjunction of all BSPs selected under 4(a) and 4(b).
5. Collect the set of resulting information flow properties. For each security predicate SP defined in step 4, one obtains a pair (VS_{SP}, SP) where VS_{SP} is the set of all views for which the security predicate SP has been chosen.

Remark 7.4.1. Often, the same security predicate can be used for all views in VS. This greatly simplifies the steps 4 and 5 above. ◇

7.4.2 The Running Example

In our case study, we perform steps 1, 2, 3(a), 4(a-c), and 5.

Confidentiality Requirements. The central confidentiality requirement for a DMWL process is that the initial values of high variables are not leaked to the low level. In other words, occurrences of events observable by the low level must not depend on the occurrence of events $setvar(h, val)$ (modeling the initialization of high variables).

Values of high variables may be communicated on high channels from one process to another. This means, messages transmitted on high channels may contain confidential information. Therefore, the receiving process must handle messages on high channels with the

same care as the initial value of its local high variable. In particular, the contents of messages received on high channels must not be leaked to the low level. Moreover, it must not be leaked whether messages are present on a high channel or not. Otherwise, it might be possible that confidential information is leaked to the low level. For example, if a process sends a message on a particular high channel only under the condition that its high (boolean) variable is true ($h = tt$) then the information whether a message is present on this channel or not is all the attacker needs to deduce the value of h. Therefore, the number of messages on a high channel as well as the contents of these messages must be kept confidential. In other words, the occurrence of low-level events must not depend on the occurrence of events $trans(cid_h, val)$ (where cid_h is an incoming high channel) that model the receival of high messages.

Integrity Requirements. There are no integrity requirements in this case study.

Specification of Security Requirements by Flow Restrictions. We specify the security requirements directly by a set of views.

We have three different classes of events for a given process p. Firstly, there are events whose occurrences are observable for the low-level environment. These include scheduling events ($schedule_p(tid)$, $yield_p(info)$), the completion of the initialization phase (event $start_p$), read events and write events on low variables ($outvar_p(l, val)$, $setvar_p(l, val)$), and transmissions on low channels ($trans(cid_l, val)$ where $level(cid_l) = low$). Secondly, there are events whose occurrences are not observable for the low-level environment and that are confidential. As pointed out before, these include write events on high variables ($setvar_p(h, val)$) and transmissions on incoming high channels ($trans(cid_h, val)$ where $level(cid_h) = high$ and $receiver(cid_h) = p$). Finally, there are events whose occurrences are not observable and that are also not confidential. These include all internal events (events in E^p_{local}), read events on high variables ($outvar_p(h, val)$), and transmissions on outgoing high channels ($trans(cid_h, val)$ where $level(cid_h) = high$ and $sender(cid_h) = p$). Output events that may involve high-level variables ($outvar_p(h, val)$) or high-level messages ($trans(cid_h, val)$) need not to be classified as confidential. Confidential aspects of such output events are completely attributable to high-level input events (*setvar*-events on high variables and ingoing *trans*-events on high channels) and the confidentiality of high-level input events is already explicitly demanded. Note that this argument is along the same lines as the explanation given by Guttman and Nadel for the idea underlying the definition of generalized noninterference (cf. Example 3.3.6). The resulting sets of events constitute a view.

Definition 7.4.2 ($\mathcal{HI}^p$). We define the view $\mathcal{HI}^p = (V^p, N^p, C^p)$ as follows:

$$V^p = \left\{ \begin{array}{c} schedule_p(tid), \quad yield_p(info), \\ outvar_p(l, val), \quad setvar_p(l, val), \\ start_p, trans(cid_l, val) \end{array} \middle| \; level(cid_l) = low \right\}$$

$$N^p = E^p_{local} \cup \left\{ \begin{array}{c} outvar_p(h, val), \\ trans(cid_h, val) \end{array} \middle| \begin{array}{c} level(cid_h) = high, \\ sender(cid_h) = p \end{array} \right\}$$

$$C^p = \left\{ \begin{array}{c} setvar_p(h, val), \\ trans(cid_h, val) \end{array} \middle| \begin{array}{c} level(cid_h) = high, \\ receiver(cid_h) = p \end{array} \right\}$$

◇

Security Predicate. A low-level adversary should neither be able to deduce that a high variable has been set to a particular value nor be able to deduce that a high variable has not been set to some particular value. Similarly, the adversary should not be able to learn whether messages have or have not been received on some high channel. Note that it is not enough to prevent him from learning which particular messages are on a channel (cf. paragraph on confidentiality requirements earlier in this section). Consequently, we need to prevent deductions about occurrences as well as about nonoccurrences of confidential events. Therefore, we choose BSPs from both dimensions when constructing the security predicate.

From the BSPs in the first dimension, we select *BSD*. Choosing a backwards strict BSP has the advantage (in comparison to choosing a non-strict BSP like R or D) that information leakage like in Example 3.4.22 is ruled out. Strict BSPs or forward correctable BSPs are no suitable alternatives. Strict BSPs (like *SR* or *SD*) are overrestrictive in the sense that they permit no corrections and that this does not offer any advantages wrt. the degree of security provided (in comparison to the respective backwards strict BSPs). Forward correctable BSPs are usually used in *addition* to other BSPs from the same dimension (rather than *instead* of these) with the objective to arrive at a composable security predicate. Here, we do not explicitly require that a forward correctable BSP must hold, however, we will return to issues of compositionality and forward correctable BSPs in Section 7.6.2.

From the BSPs in the second dimension, we select $BSIA^{\rho^p}$. The argument for choosing a backwards strict BSP (rather than strict or non-strict BSPs) is like for the first dimension.

The reason why we prefer $BSIA^{\rho^p}$ over *BSI* is that we want to permit that visible events influence the possibility of confidential events (but not vice versa). More specifically, we want to express that the enabledness of *setvar*-events may depend on whether a *start*-event has previously occurred or not. Namely, before *start* has occurred, *setvar*-events are enabled and after *start* has occurred they are disabled. Hence, *BSI* would be a too restrictive BSP in our setting because it requires that confidential events (including *setvar*-events on high variables) can be inserted at *every* point of a trace. In contrast to this, $BSIA^{\rho^p}$ requires only that confidential events can be inserted at points of a trace where they are ρ^p-admissible.

We define ρ^p by $\rho^p(\mathcal{HI}^p) = \{start_p\}$. With this choice, the possibility of *setvar*-events may depend on *start*-events (but not on any other events). More precisely, $BSIA^{\rho^p}_{\mathcal{HI}^p}(Tr_{SES^p})$ requires that the event $setvar(h, val)$ can be inserted into a possible trace $\beta.\alpha$ after β if *start* does not occur in β (i.e. if $\beta|_{\{start\}} = \langle\rangle$ holds). In other words, *setvar*-events on high variables need *not* to be insertable *after start* has occurred. Incoming *trans*-events are not affected by the ρ-admissibility assumption of *BSIA* because they are always enabled.

It remains to be shown that $BSIA^{\rho^p}$ is not too liberal in the sense that it permits leakage of confidential information. Recall from Remark 3.4.21 that this, in principle, might be the case for our choice ρ^p because $\rho^p(\mathcal{HI}^p) \subseteq C^p$ does not hold. However, in our setting the only difference between $BSIA^{\rho^p}$ and *BSI* (which would rule out deductions about nonoccurrences of confidential events completely) is that a low-level adversary can deduce that no *setvar*-event can occur after he has observed an occurrence of the event *start*. Since only the setting of high variables during the initialization phase is confidential, this possibility for the adversary is not problematic. Note that incoming *trans*-events do not depend on *start* (they are always enabled). Consequently, the adversary cannot deduce any information about the occurrence or nonoccurrence of transmissions from observing a *start*-event (if *BSD* and $BSIA^{\rho^p}$ hold).

Definition 7.4.3. We define the security predicate SP^p by $SP^p = BSD \wedge BSIA^{\rho^p}$ where $\rho^p(\mathcal{HI}^p) = \{start_p\}$. ◇

Information Flow Property.

Definition 7.4.4. We define the information flow property $IFP^p = (\{\mathcal{HI}^p\}, SP^p)$. ◇

Remark 7.4.5. Note that IFP^p is not equivalent to any of the known information flow properties that we investigated in Chapter 4. While the view $\mathcal{HI}^p$ is like the one in the *MAKS*-representations of *GNF*, *GNI*, or *FC*, i.e. high-level inputs are confidential, the security predicate differs substantially (cf. Theorems 4.2.22, 4.2.3, and 4.2.11). The shape of the security predicate $BSD \wedge BSIA^\rho$ is rather like in the *MAKS*-representation for *NDO*, *SEP*, or *PSP* (cf. Theorem 4.2.16, 4.2.24, and 4.2.26). However, these information flow properties assume a different view (i.e. that *all* high-level events are confidential). Hence, IFP^p is a novel information flow property.

Based on the modular representation of information flow properties in *MAKS*, it is an easy task to relate this novel property to the known information flow properties. In particular, we obtain that IFP^p is more restrictive than *GNF* (i.e. IFP^p implies *GNF*) and that *GNI* is more restrictive than IFP^p (i.e. *GNI* implies IFP^p). Further implications (like, e.g., that *FC* is more restrictive than IFP^p) can be easily obtained with the help of our taxonomy of known information flow properties (cf. Figure 4.4).

Note that "inventing" a novel information flow property "on demand" with the uniform concepts provided by *MAKS* (i.e. views and BSPs) did not pose a major intellectual challenge. However, without *MAKS*, we believe it would have been much more difficult to find an appropriate information flow property. For example, it is not possible to represent IFP^p by selective interleaving functions (cf. Theorem B.2.1). ◇

This concludes the specification of the security requirements for DMWL processes. We are now ready to prove that every process that executes a strongly secure DMWL program satisfies its security requirements.

7.5 Verifying Security Requirements

To illustrate the verification of information flow properties with the unwinding techniques proposed in Chapter 5 is the purpose of this section. We outline how to proceed in the verification and then illustrate this for the running example by showing that every DMWL process with a strongly secure program satisfies IFP^p and, for simplifying this proof, we apply our unwinding techniques.

7.5.1 How to Proceed

Let $ES = (E, I, O, Tr)$ be a specification of the system under consideration. Moreover, let VS be a set of views in E and SP be a security predicate. In order to verify that ES satisfies the information flow property (VS, SP), one has to verify that $\mathrm{BSP}_\mathcal{V}(Tr)$ holds for every view $\mathcal{V}$ in VS and for every basic security predicate BSP from which SP is assembled (cf. Definitions 3.2.4 and 3.2.7). For verifying $\mathrm{BSP}_\mathcal{V}(Tr)$, one can proceed as follows:

1. Select a verification technique for proving $\mathrm{BSP}_\mathcal{V}(Tr)$. For unwinding (cf. Chapter 5), proceed with 2. For simulation (cf. Appendix C), proceed with 3.[8]

[8] Obviously, it is also possible to prove $\mathrm{BSP}_\mathcal{V}(Tr)$ without the help of specialized verification techniques. An example for how to prove noninterference directly can be found, e.g., in [McL92].

2. Proving $\mathrm{BSP}_{\mathcal{V}}(Tr)$ by unwinding:
   (a) If an unwinding result (cf. Theorems 5.3.1, 5.3.2, and 5.3.3) is applicable then determine the unwinding conditions that are appropriate for $\mathrm{BSP}_{\mathcal{V}}(Tr)$.
   (b) If no unwinding result is applicable for this BSP then retrieve a more restrictive BSP for which an unwinding result is applicable by traversing the ordering of BSPs in reverse direction. Figure 7.16 summarizes the results of Theorems 3.5.3, 3.5.4, 3.5.12, and 3.5.13 that are relevant for this purpose. In the figure, BSPs for which we have presented unwinding results in Section 5.3 are marked by surrounding boxes. Afterwards, determine the unwinding conditions appropriate for verifying the more restrictive BSP using Theorems 5.3.1, 5.3.2, and 5.3.3.
   (c) Construct an unwinding relation $\ltimes \subseteq S \times S$.
   (d) Verify all previously determined unwinding conditions for this unwinding relation.

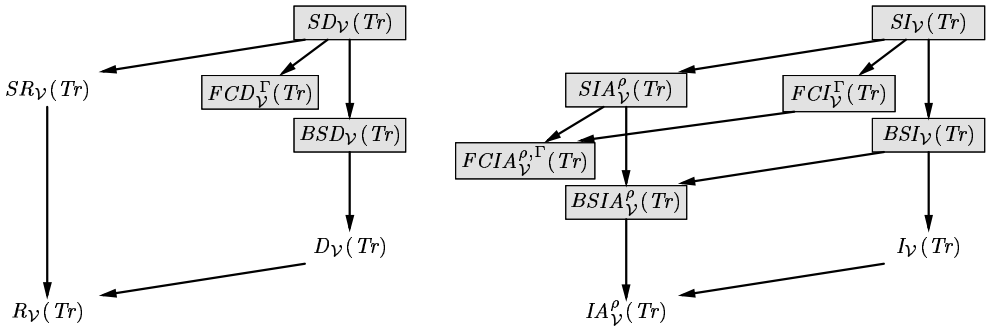


Figure 7.16: Ordering BSPs with and without local verification conditions

3. Proving $\mathrm{BSP}_{\mathcal{V}}(Tr)$ by simulation:
   (a) If Theorem C.2.1 or Theorem C.2.2 is applicable for this BSP then determine the appropriate refinement statements from these theorems.
   (b) If Theorems C.2.1 and C.2.2 are not applicable for this BSP then retrieve a stronger BSP for which one of these theorems is applicable by traversing the ordering of BSPs in reverse direction.[9] Afterwards, determine the refinement statements appropriate for the stronger BSP using Theorems C.2.1 and C.2.2.
   (c) For every refinement statement that has been determined, proceed as follows:
      i. Select one of the four simulation techniques, i.e. forward, backward, forward-backward, or backward-forward simulation.
      ii. Determine local verification conditions appropriate for verifying the refinement statement from Definitions C.1.3 (for forward simulation), C.1.5 (for backward simulation), C.1.8 (for forward-backward simulation), and C.1.9 (for backward-forward simulation).
      iii. Construct a simulation relation.
      iv. Verify all previously determined local verification conditions for this relation.

[9]The BSPs reformulated by refinement statements in Section C.2 are the same as the ones for which we have derived unwinding results (marked by surrounding boxes in Figure 7.16).

7.5.2 The Running Example

In our case study, we follow the unwinding approach (steps 1 and 2(a–d)). More specifically, unwinding is used in the proof of the following theorem, which shows that, for single processes, strong security implies IFP^p.

Theorem 7.5.1 (Satisfaction of security requirements for single processes). Let $C \in CMD$ be a DMWL command, $p \in PID$ be a process identifier, and ES be the event system that is induced by the state-event system $DMWLProcess(p, C)$.

If C is strongly secure then ES satisfies IFP^p. ◇

Proof. In the proof, we abbreviate the state-event system $DMWLProcess(p, C)$ by $SES^p = (S^p, s_0^p(C), E^p, I^p, O^p, T^p)$. For better readability, we omit the index p from state objects and events. We proceed like outlined in Section 7.5.1.

Selection of a Verification Technique. We prove the theorem by unwinding.

Unwinding Conditions. According to Theorem 5.3.1, the unwinding conditions for BSD are lrf and osc while the unwinding conditions for $BSIA^{\rho^p}$ are $lrbe^{\rho^p}$ and osc. That is, we have to prove that $lrf_{\mathcal{HI}^p}(T^p, \ltimes_1)$, $osc_{\mathcal{HI}^p}(T^p, \ltimes_1)$, $lrbe^{\rho^p}_{\mathcal{HI}^p}(T^p, \ltimes_2)$, and $osc_{\mathcal{HI}^p}(T^p, \ltimes_2)$ hold.

Unwinding Relation. We define $\ltimes_1 = \bowtie$ and $\ltimes_2 = \bowtie$ where the (symmetric) unwinding relation $\bowtie \subseteq S^p \times S^p$ is defined as follows (where $s, s' \in S^p$ are reachable states):

$s \bowtie s'$ holds if and only if the following conditions are all true:

1. $initialized_s = initialized_{s'}$
2. $executed_s = executed_{s'}$
3. $ainfo_s = ainfo_{s'}$
4. $atid_s = atid_{s'}$
5. $inbuf_s =_L inbuf_{s'} \wedge pending_s =_L pending_{s'}$
6. $\forall cid_l \in CID.\, \forall val \in VAL.\, (level(cid_l) = low \Rightarrow (outbuf_s = (cid_l, val) \Leftrightarrow outbuf_{s'} = (cid_l, val)))$
7. $mem_s =_L mem_{s'}$
8. $\forall tid \in TID.(thread_s(tid) = \bot = thread_{s'}(tid) \vee thread_s(tid) = \top = thread_{s'}(tid) \vee thread_s(tid) = \langle\rangle = thread_{s'}(tid) \vee thread_s(tid) \approx_L thread_{s'}(tid))$
9. $blocked\text{-}set_s = blocked\text{-}set_{s'}$

Verification of Unwinding Conditions. According to Theorem 5.3.1, we have to prove that $lrf_{\mathcal{HI}^p}(T^p, \bowtie)$, $lrbe^{\rho^p}_{\mathcal{HI}^p}(T^p, \bowtie)$, and $osc_{\mathcal{HI}^p}(T^p, \bowtie)$ hold. As an example, we prove the first of these unwinding conditions here. The other two unwinding conditions are proven in Appendix E.4.

Lemma 7.5.2. Let $C \in CMD$, $p \in PID$, and $SES^p = (S^p, s_0^p, E^p, I^p, O^p, T^p)$, be defined by $SES^p = DMWLProcess(p, C)$. Moreover, let $\bowtie \subseteq S \times S$ be defined like in the proof of Theorem 7.5.1. If C is strongly secure then $lrf_{\mathcal{HI}^p}(T^p, \bowtie)$ holds. $\diamond$

Proof. Let $s, s' \in S^p$ and $c \in C^p$ be arbitrary. Assume $reachable(SES^p, s)$ and $s \xrightarrow{c}_{T^p} s'$ hold. According to the definition of C^p, either $c = setvar(h, val)$ or $c = trans(cid_h, val)$ holds (where $val \in VAL$, $cid_h \in CID$ with $receiver(cid_h) = p$ and $level(cid_h) = high$). We make a case distinction on these two possibilities.

Firstly, assume $c = setvar(h, val)$. According to Figure 7.12, an occurrence of this event only affects the value of $mem(h)$. Hence, $mem_{s'} =_L mem_s$. Conditions 1–9 are true for s' and s (Condition 8 is ensured by Lemma E.4.2). Consequently, $s' \bowtie s$ holds.

Secondly, assume $c = trans(cid_h, val)$. According to Figure 7.12, an occurrence of this event only affects the value of $inbuf(cid_h)$. Since $level(cid_h) = high$, we have $inbuf_s =_L inbuf_{s'}$. Conditions 1–9 are true for s' and s (Condition 8 is ensured by Lemma E.4.2). Consequently, $s' \bowtie s$ holds. $\square$

From Lemma E.4.1 in the appendix, we obtain that $lrbe^{\rho^p}_{\mathcal{HI}^p}(T^p, \bowtie)$ and $osc_{\mathcal{HI}^p}(T^p, \bowtie)$ hold. From Theorem 5.3.1, we obtain that ES satisfies IFP^p.

This concludes the proof of Theorem 7.5.1. $\square$

The following theorem shows that if a DMWL process executes a program that is the result of a successful type check with the type system from Figure 7.5 then the process satisfies its security requirements. This theorem provides a basis for checking IFP^p mechanically.

Theorem 7.5.3. Let $C, C' \in CMD$ be DMWL commands, $p \in PID$ be a process identifier, and ES be the event system induced by the state-event system $DMWLProcess(p, C')$.

If $\vec{C} \hookrightarrow \vec{C}' : \vec{S}l$ is derivable then ES satisfies IFP^p. $\diamond$

Proof. Follows immediately from Theorems 7.2.13 and 7.5.1. $\square$

This concludes the security analysis of individual processes. We have shown that every process that executes a strongly secure program satisfies its security requirements (in Theorem 7.5.1) and that these security requirements can be checked with the type system in Figure 7.5.

7.6 Composing Systems

Let us now outline how information flow properties can be verified for complex systems that are composed of multiple components by exploiting a compositionality theorem. In the running example, we investigate distributed systems that are composed of one or more DMWL processes, each of which is strongly secure. We identify the security requirements for such a system along the same lines as for single processes. Then we verify that the composed system satisfies its security requirements. In order to simplify this proof, we employ a compositionality theorem.

7.6.1 How to Proceed during System Composition

In the following, we assume that each system component has already been specified by an event system, i.e. there is a finite set $\mathcal{ES}$ of event systems where each $ES' \in \mathcal{ES}$ specifies one system component. Moreover, we assume that the security requirements of each component

have been specified by a set of information flow properties that have already been verified. More precisely, we assume that, for each component $ES' \in \mathcal{ES}$, a finite set of information flow properties $\{IFP_1, \ldots, IFP_{n_{ES'}}\}$ has been verified.

A formal security model for the composed system can be constructed as follows:

1. Specify the overall system by the event system $ES = (E, I, O, Tr)$ that is defined by $ES = \|_{ES' \in \mathcal{ES}} ES'$. This means, ES is the event system that results when composing the event systems for the system components.

2. Specify the security requirements for the composed system by a set of information flow properties $\{IFP_1, \ldots, IFP_{n_{ES}}\}$. This can be done like explained in Section 7.4.

3. For each information flow property IFP_j specified in step 2, proceed as follows:
   (a) Check whether a known compositionality theorem is applicable in the sense that it entails that ES satisfies IFP_j and all of its preconditions are fulfilled. For example, the compositionality theorems in Sections 6.5–6.7 are possible candidates. If a known compositionality theorem is applicable then conclude that ES satisfies IFP_j.
   (b) If none of the known compositionality theorems is applicable then try to derive a compositionality theorem (e.g. along the lines described in Section 6.5). If you have been successful and the derived result is applicable then conclude that ES satisfies IFP_j (and add the theorem to your collection for use in future verifications).
   (c) If the attempts under 3(a) and 3(b) are not successful then do one of the following:
      i. Choose an information flow property $IFP_j{}'$ that is more restrictive and return to step 3(a) in order to verify that ES satisfies $IFP_j{}'$.
      ii. Prove IFP_j without the help of a compositionality theorem (like described in Section 7.5).

7.6.2 The Running Example

In our running example, we proceed as follows. After performing steps 1, 2, 3(a–b), we will detect that no suitable compositionality theorem can be derived for IFP^P. Therefore, we will move to a more restrictive information flow property (step 3(c)i). After performing step 3(a,b) for this strengthened property, we will succeed with our verification effort.

In the following, let PID be a set of process identifiers and, for each $p \in PID$, let $initthread_p \in CMD$ be a DMWL-command.

System Specification. In this section, we employ the system model of event systems (rather than that of state-event systems). In the following two definitions, we specify DMWL processes and DMWL process pools, respectively, by event systems.

Definition 7.6.1 (ES^p). For $p \in PID$, we define $ES^p = (E^p, I^p, O^p, Tr^p)$ as the event system that is induced by the state-event system $DMWLProcess(p, initthread_p)$. ◇

Definition 7.6.2 (ES^P). Let $P = \{p_1, \ldots, p_n\} \subseteq PID$ be finite and nonempty. We define $ES^P = (E^P, I^P, O^P, Tr^P)$ by $ES^P = ES^{p_1} \| \ldots \| ES^{p_n}$. ◇

The following theorem shows that ES^P is the event system that is induced by the state-event system $DMWLProcPool((p_1, initthread_1), \ldots, (p_n, initthread_n))$, i.e. the state-event system that models the DMWL process pool.

Theorem 7.6.3. Let $P = \{p_1, \ldots, p_n\} \subseteq PID$ be finite and nonempty. The event system induced by $DMWLProcPool((p_1, initthread_1), \ldots, (p_n, initthread_n))$ and ES^P are equal. ◇

Proof. Follows from Theorem 7.3.7 and Definitions 7.3.8, 7.6.1, and 7.6.2. □

Security Specification. The security requirements for DMWL process pools can be determined along the same lines as for individual DMWL processes (cf. the paragraphs on confidentiality requirements and integrity requirements in Section 7.5.2).

The resulting security requirement is that initial values of high variables must not be leaked to the low level. In other words, occurrences of events observable for the low-level environment must not depend on the occurrence of events $setvar_p(h, val)$ (modeling the initialization of the high variable in some process $p \in P$). Values of high variables may be communicated on high channels between processes and process pools. Therefore, a process pool must handle messages on incoming high channels with the same care as the initial value of high variables in any of its processes. Moreover, it must not be leaked how many messages are present on incoming high channels. The resulting security requirements are captured in the following definition of the view $\mathcal{HI}^P$, which is very similar to the specification of the security requirements of a single DMWL process by the view $\mathcal{HI}^p$ (cf. Definition 7.4.2).

Definition 7.6.4. For a finite and nonempty set $P \subseteq PID$, the view $\mathcal{HI}^P = (V^P, N^P, C^P)$ is defined as follows:

$$V^P = \left\{ \begin{array}{c} schedule_p(tid), \quad yield_p(info), \\ outvar_p(l, val), \quad setvar_p(l, val), \\ start_p, \quad trans(cid_l, val) \end{array} \middle| \begin{array}{c} p \in P, \\ level(cid_l) = low \end{array} \right\}$$

$$N^P = \bigcup_{p \in P} E^p_{local} \cup \left\{ \begin{array}{c} outvar_p(h, val), \\ trans(cid_h, val) \end{array} \middle| \begin{array}{c} p \in P, \\ level(cid_h) = high, \\ sender(cid_h) \in P \end{array} \right\}$$

$$C^P = \left\{ \begin{array}{c} setvar_p(h, val), \\ trans(cid_h, val) \end{array} \middle| \begin{array}{c} p \in P, level(cid_h) = high, \\ receiver(cid_h) \in P, \\ sender(cid_h) \notin P \end{array} \right\}$$ ◇

According to the above definition, $V^P = \bigcup_{p \in P} V^p$ and $N^P = \bigcup_{p \in P} N^p$ hold.

The security predicate for DMWL process pools can be determined along the same lines as for individual processes (cf. the corresponding paragraph in Section 7.5.2). Like for individual processes, this leads to the selection of BSD and an instance of $BSIA^\rho$. The admissibility function ρ^P that is used to instantiate $BSIA^\rho$ returns the set of all *start*-events of processes in the process pool, i.e. $\rho^P(\mathcal{HI}^P) = \{start_p \mid p \in P\}$. It is important that *all* *start*-events are contained in this set because the set C^P of confidential events contains events $setvar_p(h, val)$ for *all* processes in the process pool.

Definition 7.6.5. Let $P \subseteq PID$ be finite and nonempty. Define $SP^P = BSD \wedge BSIA^{\rho^P}$ where $\rho^P(\mathcal{HI}^P) = \{start_p \mid p \in P\}$. ◇

This results in the following information flow property for process pools.

Definition 7.6.6. Let $P \subseteq PID$ be finite and nonempty. Define the information flow property $IFP^P = (\{\mathcal{HI}^P\}, SP^P)$. ◇

Obviously, the event system that specifies a process pool that consists of a single process only equals the event system that specifies this process (i.e. $ES^{\{p\}} = ES^p$ holds). The specification of the security requirements for a process pool that consists of a single process should also equal the specification of the security requirements of that process. The following three theorems demonstrate that this is, indeed, the case.

Theorem 7.6.7. For each $p \in PID$, we have $\mathcal{HI}^{\{p\}} = \mathcal{HI}^p$. ◇

Proof. The equations $V^{\{p\}} = V^p$, $N^{\{p\}} = N^p$, and $C^{\{p\}} = C^p$ follow immediately from Definitions 7.4.2 and 7.6.4. □

Theorem 7.6.8. For each $p \in PID$, we have $SP^{\{p\}} = SP^p$. ◇

Proof. Follows immediately from Definitions 7.4.3 and 7.6.5. □

Theorem 7.6.9. For each $p \in PID$, we have $IFP^{\{p\}} = IFP^p$. ◇

Proof. Follows from Definitions 7.4.4 and 7.6.6 and Theorems 7.6.7 and 7.6.8. □

Verification of Security Requirements. The following theorem states that, for arbitrary collections of DMWL processes, strong security implies that IFP^P holds. In other words, if every program in a distributed system is strongly secure then the overall system satisfies its security requirements.

Theorem 7.6.10 (Satisfaction of security requirements for DMWL process pools). Let $P \subseteq PID$ be finite and nonempty. If, for each $p \in P$, $initthread_p$ is strongly secure then ES^P satisfies IFP^P. ◇

Rather than proving from scratch that IFP^P holds for ES^P, which would be quite tedious, we want to exploit the fact that IFP^p holds for each individual process ES^p (follows from Theorem 7.5.1) by applying a compositionality theorem.

The following theorem permits us to view ES^P as the composition of two event systems.

Theorem 7.6.11. Let $P, P', P'' \subseteq PID$ be finite and nonempty. If $P' \cap P'' = \emptyset$ and $P' \cup P'' = P$ hold then $ES^P = ES^{P'} \parallel ES^{P''}$ holds. ◇

Proof. Follows immediately from Definition 7.6.2 and the associativity/commutativity of the composition operator $\parallel$ (cf. Theorems 6.2.3 and 6.2.4). □

For proving Theorem 7.6.10, we proceed like outlined in Section 7.6.1 (steps 3(a)–3(c)).

Check for Known Compositionality Theorems. Since IFP^P is a novel information flow property, no compositionality theorem is known for it.

Derive a Compositionality Theorem. Unfortunately, we cannot derive a general compositionality theorem for IFP^P. Let us illustrate the reasons for this at the example of a system ES^P that is composed of two components $ES^{P'}$ and $ES^{P''}$ that satisfy $IFP^{P'}$ and $IFP^{P''}$, respectively. Since output events of $ES^{P'}$ may be input events of $ES^{P''}$ and output events of $ES^{P''}$ may be input events of $ES^{P'}$, $N^{P'} \cap E^{P''} \neq \emptyset$ as well as $N^{P''} \cap E^{P'} \neq \emptyset$ may hold. In other words, corrections made during the local information flow analysis of $ES^{P'}$

may cause perturbations for $ES^{P''}$ and corrections made during the local information flow analysis of $ES^{P''}$ may cause perturbations for $ES^{P'}$. Since the BSPs BSD and $BSIA^{\rho^P}$ are ingredients of IFP^P, it is possible to locally correct for each component the perturbations caused by the respective other component. However, there is no guarantee that the overall process of locally correcting perturbations terminates because correcting a perturbation for the first component may lead to perturbations for the second component, whose corrections for the second component may lead to perturbations for the first component, and so on. In other words, we cannot conclude that the composition of $ES^{P'}$ and $ES^{P''}$ is well behaved wrt. the views $\mathcal{HI}^{P'}$ and $\mathcal{HI}^{P''}$ (cf. Definition 6.3.6). This is the reason why we cannot derive a compositionality theorem for IFP^P,

Since we cannot derive a compositionality theorem for IFP^P, we move to a more restrictive information flow property.

Strengthening the Information Flow Property. Our above observations together with the definition of a well-behaved composition (cf. Definition 6.3.6) suggest two possibilities for strengthening IFP^P. Firstly, we could modify the definition of $\mathcal{HI}^P$ (aiming at making condition 1, 2, or 3 in Definition 6.3.6 true) and, secondly, we could change the definition of SP^P (aiming at making condition 4 in Definition 6.3.6 true). We choose the second possibility.[10] More specifically, we strengthen the security predicate by adding an instance of FCI^Γ. Adding an instance of FCI^Γ to BSD and $BSIA^{\rho^P}$ has the effect that more perturbations are demanded and that fewer corrections are permitted. Namely, for $\Gamma = (\nabla, \Delta, \Upsilon)$, events in $C \cap \Upsilon$ must be insertable at *every* point of a trace where some event in $V \cap \nabla$ is enabled (rather than only at points where they are ρ-admissible) and the permitted corrections are constrained when an event in $C \cap \Upsilon$ is inserted at a point where it is immediately followed by an occurrence of some event in $V \cap \nabla$ (cf. Section 3.4.5 for how forward-correctable BSPs restrict the permitted corrections). Let us define appropriate parameters for FCI^Γ.[11]

Definition 7.6.12. Let $P \subseteq PID$ be finite and nonempty. We define $\Gamma^P = (\nabla^P, \Delta^P, \Upsilon^P)$ by:

$$\begin{aligned} \nabla^P &= \{trans(cid, val) \mid receiver(cid) \in P, sender(cid) \notin P\} \\ \Delta^P &= \emptyset \\ \Upsilon^P &= \nabla^P \end{aligned}$$ ◇

This results in the following modified security predicate and information flow property.

Definition 7.6.13 (CSP^P). Let $P \subseteq PID$ be finite and nonempty. We define the security predicate $CSP^P = BSD \wedge BSIA^{\rho^P} \wedge FCI^{\Gamma^P}$. ◇

Definition 7.6.14 ($CIFP^P$). Let $P \subseteq PID$ be finite and nonempty. We define the information flow property $CIFP^P = (\{\mathcal{HI}^P\}, CSP^P)$.[12] ◇

Since we have constructed CSP^P from SP^P by adding FCI^{Γ^P}, it is quite obvious that $CIFP^P$ implies IFP^P. For future references, this is stated in the following theorem.

[10] Choosing the first possibility would result in a quite essential change of the information flow property and there is little hope that a composable property derived along these lines would be implied by strong security.

[11] The parameter $\Gamma^P = (\nabla^P, \Delta^P, \Upsilon^P)$ is chosen such that the conditions demanded by Definition 6.3.6 and Theorem 6.4.1 are satisfied. That this is indeed the case will be shown in Lemmas 7.6.17, 7.6.18, and 7.6.19.

[12] The "C" in *CSP* and *CIFP* stands for "composable".

Theorem 7.6.15. Let $P \subseteq PID$ be finite and nonempty. If ES^P satisfies $CIFP^P$ then ES^P also satisfies IFP^P. ◇

Proof. Follows immediately from Definitions 7.6.6, 7.6.5, 7.6.14, and 7.6.13. □

In general, $BSIA^{\rho\,P}$ does not imply $FCI^{\Gamma\,P}$ (i.e. for arbitrary systems). Hence, $CIFP^P$ is more restrictive than IFP^P. Fortunately, for the special case of DMWL processes, the satisfaction of IFP^p implies the satisfaction of $CIFP^{\{p\}}$.

Theorem 7.6.16. Let $p \in PID$. If ES^p satisfies IFP^p then ES^p satisfies $CIFP^{\{p\}}$. ◇

Proof. We show that if $BSIA^{\rho\,p}_{\mathcal{HI}^p}(Tr^p)$ holds then $FCI^{\Gamma\,\{p\}}_{\mathcal{HI}^{\{p\}}}(Tr^p)$ holds.

Assume $\alpha, \beta \in E^*$, $c \in C^{\{p\}} \cap \Upsilon^{\{p\}}$, and $v \in V^{\{p\}} \cap \nabla^{\{p\}}$ with $\beta.\langle v\rangle.\alpha \in Tr^p$ and $\alpha|_{C^{\{p\}}} = \langle\rangle$. Hence, $c = trans(cid, val)$ where $level(cid) = high$ and $receiver(cid) = p$. Moreover, we have $v = trans(cid', val')$ with $level(cid') = low$ and $receiver(cid') = p$.

$BSIA^{\rho\,p}_{\mathcal{HI}^p}(Tr^p)$, $total(ES^p, \{trans(cid, val) \mid receiver(cid) = p\})$, and Theorem 7.6.7 ensure that $\alpha' \in E^{p*}$ exists with $\beta.\langle v.c\rangle.\alpha' \in Tr^p$, $\alpha'|_{V^{\{p\}}} = \alpha|_{V^{\{p\}}}$, $\alpha'|_{C^{\{p\}}} = \langle\rangle$. The event c has no preconditions and affects only $inbuf_p(cid)$ (where $p = receiver(cid)$). Since $cid' \neq cid$, the occurrence of $trans(cid', val')$ neither depends on $inbuf_p(cid)$ nor does it affect $inbuf_p(cid)$. Hence, c and v can be exchanged in the trace. This means $\beta.\langle c.v\rangle.\alpha' \in Tr^p$ holds. □

We are now ready to return to step 3(a) with the strengthened information flow property.

Check for Known Compositionality Theorem. Since $CIFP^P$ is a novel information flow property, no compositionality theorem is known for it.

Try to Derive a Compositionality Theorem. We proceed like described in Section 6.5: Firstly, we show that the views for the system components constitute a proper separation of the view for the composed system. Secondly, we show that the composition of the components is well behaved wrt. the views for the components. Thirdly, we verify the preconditions of the compositionality results for all BSPs from which CSP^P is assembled. Finally, we apply these compositionality results and conclude that $CIFP^P$ holds for the composed system.

The following lemma shows that $\mathcal{HI}^{P'}$ and $\mathcal{HI}^{P''}$ constitute a proper separation of $\mathcal{HI}^P$.

Lemma 7.6.17 (Proper separation). Let $P, P', P'' \subseteq PID$ be finite, nonempty sets such that $P' \cap P'' = \emptyset$ and $P = P' \cup P''$ hold. The following propositions are valid:

1. $V \cap E^{P'} = V^{P'}$ and $V \cap E^{P''} = V^{P''}$ hold.
2. $C \cap E^{P'} \subseteq C^{P'}$ and $C \cap E^{P''} \subseteq C^{P''}$ hold.
3. $N^{P'} \cap N^{P''} = \emptyset$ holds. ◇

Proof. Follows immediately from Definition 7.6.4. □

We show that the composition of $ES^{P'}$ and $ES^{P''}$ is well behaved wrt. $\mathcal{HI}^{P'}$ and $\mathcal{HI}^{P''}$.

Lemma 7.6.18 (Well-behaved composition). Let $P, P', P'' \subseteq PID$ be finite and nonempty sets with $P' \cap P'' = \emptyset$ and $P = P' \cup P''$. If $ES^{P'}$ and $ES^{P''}$ satisfy $CIFP^{P'}$ and $CIFP^{P''}$, respectively, then the composition of $ES^{P'}$ and $ES^{P''}$ is well behaved wrt. $\mathcal{HI}^{P'}$ and $\mathcal{HI}^{P''}$. ◇

Proof. We show that condition 4 in Definition 6.3.6 is fulfilled.

$BSIA^{\rho^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$ and $BSIA^{\rho^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$ hold by assumption. Since *trans*-events that are input events of $ES^{P'}$ and $ES^{P''}$ are always enabled, we have $total(ES^{P'}, C^{P'} \cap \Upsilon^{P'})$ and $total(ES^{P''}, C^{P''} \cap \Upsilon^{P''})$. Hence, condition 4(a) holds.

$FCIA^{\rho^{P'},\Gamma^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$ follows from $FCI^{\Gamma^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$, and $FCIA^{\rho^{P''},\Gamma^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$ follows from $FCI^{\Gamma^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$. Hence, condition 4(b) holds.

DMWL process pools that are composed may only have *trans*-events in common. Moreover, the shared events must be input events of one process pool and output events of the other. Since $\nabla^{P'}$ and $\nabla^{P''}$ contain all *trans*-events that are input events for $ES^{P'}$ and $ES^{P''}$, respectively, we have $V^{P'} \cap V^{P''} \subseteq \nabla^{P'} \cup \nabla^{P''}$. Hence, condition 4(c) holds.

Since the components may only have *trans*-events in common, all events in $C^{P'}$ and $C^{P''}$ are input events of $ES^{P'}$ and $ES^{P''}$, respectively, and $\Upsilon^{P'}$ and $\Upsilon^{P''}$ contains all *trans*-events that are input events for $ES^{P'}$ and $ES^{P''}$, respectively, we have $C^{P'} \cap N^{P''} \subseteq \Upsilon^{P'}$ and $C^{P''} \cap N^{P'} \subseteq \Upsilon^{P''}$. Hence, condition 4(d) holds.

From $\Delta^{P'} = \emptyset$ and $\Delta^{P''} = \emptyset$, we obtain that $N^{P'} \cap \Delta^{P'} \cap E^{P''} = \emptyset$ and $N^{P''} \cap \Delta^{P''} \cap E^{P'} = \emptyset$ hold. Hence, condition 4(e) holds. □

Let us show that the remaining preconditions of Theorem 6.4.1(1,3,5) are satisfied.

Lemma 7.6.19. Let $P, P', P'' \subseteq PID$ be finite, nonempty sets such that $P' \cap P'' = \emptyset$ and $P = P' \cup P''$ hold. If $ES^{P'}$ and $ES^{P''}$ satisfy $CIFP^{P'}$ and $CIFP^{P''}$, respectively, then the following propositions are valid:

1. $\rho^{P'}(\mathcal{HI}^{P'}) \subseteq \rho^{P}(\mathcal{HI}^{P}) \cap E^{P'}$ and $\rho^{P''}(\mathcal{HI}^{P''}) \subseteq \rho^{P}(\mathcal{HI}^{P}) \cap E^{P''}$ hold.
2. $total(ES^{P'}, C^{P'} \cap \Upsilon^{P'})$ and $total(ES^{P''}, C^{P''} \cap \Upsilon^{P''})$ hold.
3. $\nabla^{P} \cap E^{P'} \subseteq \nabla^{P'}$ and $\nabla^{P} \cap E^{P''} \subseteq \nabla^{P''}$ hold.
4. $\Upsilon^{P} \cap E^{P'} \subseteq \Upsilon^{P'}$ and $\Upsilon^{P} \cap E^{P''} \subseteq \Upsilon^{P''}$ hold.
5. $\Delta^{P} \supseteq ((N^{P'} \cap \Delta^{P'}) \cup (N^{P''} \cap \Delta^{P''}))$ holds.
6. $N^{P'} \cap \Delta^{P'} \cap E^{P''} = \emptyset$ and $N^{P''} \cap \Delta^{P''} \cap E^{P'} \subseteq \Upsilon^{P'}$ hold. ◇

Proof. We have $\rho^{P'}(\mathcal{HI}^{P'}) = \{start_p \mid p \in P'\} = \{start_p \mid p \in P\} \cap E^{P'} = \rho^{P}(\mathcal{HI}^{P}) \cap E^{P'}$. That $\rho^{P''}(\mathcal{HI}^{P''}) \subseteq \rho^{P}(\mathcal{HI}^{P}) \cap E^{P''}$ holds can be shown along the same lines.

Proposition 2 has already been shown in the proof of Lemma 7.6.18.

Since ∇^{P} and $\nabla^{P'}$ contain only *trans*-events that are input events of ES^{P} and $ES^{P'}$, respectively, $\nabla^{P} \cap E^{P'}$ equals $\nabla^{P'}$ minus the set of *trans*-events on channels that become communication events when composing the components. Hence, $\nabla^{P} \cap E^{P'} \subseteq \nabla^{P'}$ holds. That $\nabla^{P} \cap E^{P''} \subseteq \nabla^{P''}$ holds can be shown along the same lines.

Proposition 4 can be shown like proposition 2 ($\Upsilon^{P} = \nabla^{P}$ holds).

Propositions 5 and 6 follow immediately from $\Delta^{P'} = \emptyset$ and $\Delta^{P''} = \emptyset$. □

We are now ready to derive a compositionality theorem for $CIFP^{P}$.

Theorem 7.6.20. Let $P, P', P'' \subseteq PID$ be finite, nonempty sets such that $P' \cap P'' = \emptyset$ and $P = P' \cup P''$ hold. If $ES^{P'}$ and $ES^{P''}$ satisfy $CIFP^{P'}$ and $CIFP^{P''}$, respectively, then ES^{P} satisfies $CIFP^{P}$. ◇

Proof. ES^P satisfies $CIFP^P$ iff and only if $BSD_{\mathcal{HI}^P}(Tr^P) \wedge BSIA^{\rho^P}_{\mathcal{HI}^P}(Tr^P) \wedge FCI^{\Gamma^P}_{\mathcal{HI}^P}(Tr^P)$ (cf. Definition 7.6.14). By assumption, $BSD_{\mathcal{HI}^{P'}}(Tr^{P'})$, $BSD_{\mathcal{HI}^{P''}}(Tr^{P''})$, $BSIA^{\rho^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$, $BSIA^{\rho^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$, $FCI^{\Gamma^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$, and $FCI^{\Gamma^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$. We obtain from Lemmas 7.6.17, 7.6.18, and 7.6.19 that Theorem 6.4.1(1,3,5) is applicable.

$BSD_{\mathcal{HI}^P}(Tr^P)$ follows from Theorem 6.4.1(1), $BSD_{\mathcal{HI}^{P'}}(Tr^{P'})$, and $BSD_{\mathcal{HI}^{P''}}(Tr^{P''})$.

$BSIA^{\rho^P}_{\mathcal{HI}^P}(Tr^P)$ follows from Theorem 6.4.1(3), $BSIA^{\rho^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$, $BSIA^{\rho^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$.

$FCI^{\Gamma^P}_{\mathcal{HI}^P}(Tr^P)$ follows from Theorem 6.4.1(5), $FCI^{\Gamma^{P'}}_{\mathcal{HI}^{P'}}(Tr^{P'})$, and $FCI^{\Gamma^{P''}}_{\mathcal{HI}^{P''}}(Tr^{P''})$. □

We are now ready to prove that a DMWL process pool ES^P that executes programs that are strongly secure satisfies $CIFP^P$. In the proof of the following theorem, we exploit the fact that if a DMWL Process ES^p is strongly secure then it satisfies $CIFP^{\{p\}}$ and the fact that $CIFP^P$ is preserved when composing DMWL process pools.

Theorem 7.6.21. Let $P \subseteq PID$ be finite and nonempty.

If, for each $p \in P$, $initthread_p$ is strongly secure then ES^P satisfies $CIFP^P$. ◇

Proof. We prove the theorem by induction on the size of P.

In the *base case* $P = \{p\}$ holds for some $p \in PID$. Since $initthread_p$ is strongly secure, Theorem 7.5.1 implies that ES^p satisfies IFP^p. From Theorems 7.6.9 and 7.6.16, we obtain that $ES^{\{p\}}$ satisfies $CIFP^{\{p\}}$. This concludes the proof of the base case.

In the *step case*, let $P', P'' \subseteq PID$ be finite, nonempty sets such that $P' \cap P'' = \emptyset$ and $P = P' \cup P''$ hold. Since P' and P'' each contain strictly fewer process identifiers than P, the induction hypothesis is applicable. We obtain from the induction hypothesis that $ES^{P'}$ and $ES^{P''}$ satisfy $CIFP^{P'}$ and $CIFP^{P''}$, respectively. From Theorem 7.6.20, we obtain that ES^P satisfies $CIFP^P$. □

Now we are ready to prove Theorem 7.6.10, which is the main result of this section

Proof (of Theorem 7.6.10). According to Theorem 7.6.21, ES^P satisfies $CIFP^P$. From Theorem 7.6.15, we obtain that ES^P satisfies IFP^P. □

We have finally shown that a DMWL process pool ES^P that executes strongly secure programs fulfills its security requirements (modeled by the information flow property IFP^P).

The following theorem shows that if a DMWL process pool executes programs that are the result of a successful type check with the type system from Figure 7.5 then the process pool satisfies its security requirements. This theorem provides the basis for checking IFP^P mechanically.

Theorem 7.6.22. Let $P \subseteq PID$ be finite and nonempty. For each $p \in P$, let $C_p \in C\vec{M}D$.

If, for each $p \in P$, $C_p \hookrightarrow initthread_p : \vec{Sl}$ is derivable then ES^P satisfies IFP^P. ◇

Proof. Follows immediately from Theorems 7.2.13 and 7.6.10. □

Note that, by following the approach proposed in Section 6.5, the derivation of a compositionality theorem for $CIFP^P$ has been quite straightforward (cf. proof of Theorem 7.6.20). In other words, we have demonstrated by our case study that it is feasible to use novel information flow properties (for which no unwinding theorems or compositionality theorems are known) in the construction of a formal security model and to derive unwinding theorems and compositionality theorems for this property when they are needed.

Remark 7.6.23. In Theorem 7.5.1 we have shown that every strongly secure process (with identifier p) satisfies the information flow property IFP^p. This theorem has been applied in the base case of the proof of Theorem 7.6.21. A comparison of the proofs of Theorems 7.5.1 and 7.6.21 shows how much simpler it is to prove the satisfaction of an information flow property with a compositionality theorem than to prove it with the help of unwinding. For a fair comparison of the two proofs, the proof of Lemmas 7.5.2, E.4.1, and E.4.2 should be counted as being part of the proof of Theorem 7.5.1. Note also that Theorem 7.6.21 involves a more complex system than Theorem 7.5.1 (process pools rather than individual processes).

The obvious advice that follows from this is: if possible, prove security requirements with the help of a compositionality theorem. In particular, for complex systems, this is preferable to proving security requirements directly (with or without unwinding). ◇

7.7 History of this Case Study

Sabelfeld and Sands investigated language-based security in the context of a simple imperative programming language, the MWL language [SS00]. This language incorporates features for multi-threaded programming but does not provide the commands for inter-process communication that are essential for distributed programming (cf. the beginning of Section 7.2). The version of the strong security condition that Sabelfeld and Sands proposed for MWL can be obtained from Definition 7.2.9 by dropping all requirements for channel status functions (in particular, in Definition 7.2.7). Channel status functions are not needed in a non-distributed setting. For checking the strong security condition for MWL programs, Sabelfeld and Sands's proposed a security type system. Their type system can be obtained from the one in Figure 7.5 by dropping the rules for communication commands. Sabelfeld and Sands proved formally that their type system is sound in the sense that if the type check succeeds then a strongly secure program results from the transformation. Moreover, they showed that strong security is a scheduler-independent security condition (deriving it from a scheduler-dependent security condition by universally quantifying over all schedulers from a large class). That is, programs that are strongly secure can be securely executed on systems with arbitrary schedulers. In particular, this covers the case where the adversary chooses the scheduler. However, strong security has not been formally compared to more established security properties. The main motivation for a first version of this case study (presented in [MS01]) was to establish such a connection in a rigorous way and, thereby to increase the confidence in the strong security condition. This first version covered the scheduler-independent language-based techniques for MWL as described in [SS00].

The first version of our case study inspired an extension of the language-based techniques to a distributed programming language with commands for inter-process communication. In [MS03a], the extended language DMWL is described and a suitable modification of strong security is defined. A type system for checking the modified strong security condition is presented in [SM02]. This extension of Sabelfeld and Sands's approach to language-based security constituted the starting point of our case study in this chapter.

In a second version of our case study (also presented in [MS03a]), we have lifted the results of the first version to DMWL. We defined an information flow property *SecProp*, proved that every strongly secure process satisfies *SecProp*, and then verified with the help of a compositionality result that collections of strongly secure processes (i.e. distributed programs) also satisfy *SecProp*. This second version of our case study has the same scope as the one

presented in this chapter but, nevertheless, there are a few noteworthy differences.

One difference is how the information flow properties have been verified. While we have verified them without using any verification technique in [MS03a], we have applied the unwinding technique here in order to simplify the proof. For lifting this result to distributed programs, a compositionality theorem has been derived from scratch in [MS03a]. In contrast to this, we here have employed the approach proposed in Chapter 6 for deriving a suitable compositionality theorem. In particular, we have exploited our compositionality results for BSPs in this process. This considerably simplified the proof of the compositionality result (both in length and complexity).

The presentation in [MS03a] (or [MS01]) has a peculiarity that we have omitted in this chapter. This is that the system specification is constructed in two steps. Firstly, a generic specification of processes is defined that is independent from the particular programming language. Secondly, this generic specification is specialized for DMWL (or MWL). The information flow property is defined using terms only from the generic specification. The objective of this approach was to make possible an extension to other programming languages besides DMWL (MWL). Having the option to use different languages for different processes in a distributed program but, nevertheless, having a uniform notion of security for all processes is an appealing perspective. However, this possibility is an option for the future that has not been exploited so far.

For completeness, Appendix E.5 lists the remaining differences between the second version of our case study and the version in this chapter.

7.8 Summary

In this chapter, we have demonstrated how our framework *MAKS* (Chapter 3) and the results about information flow properties that we derived with the help of *MAKS* (in Chapters 4–6) can be successfully applied in the context of a concrete case study. The case study that we have investigated stems from the area of language-based security.

We have presented preliminaries on language-based security to the extent necessary for understanding the case study (cf. Section 7.2). In particular, we have defined syntax and semantics of the DMWL language, a simple imperative language offering features for distributed and multi-threaded programming. We have also presented the definition of strong security, a language-based security condition, and a sound security type system for checking strong security mechanically. The particular variant of language-based security that we investigated is an extension of the approach by Sabelfeld and Sands [SS00] to a distributed setting. This extension is a joint work by Sabelfeld and myself [MS03a, SM02], which, interestingly, was motivated by a previous version of our case study [MS01].

We have specified DMWL processes in an event-based specification formalism (cf. Section 7.3). This specification captures the operational semantics of DMWL in an adequate way (cf. Theorems E.3.5–E.3.8). The specification of DMWL processes constitutes the system component of our security model (cf. Figure 3.1), which is a prerequisite for applying information flow properties in a concrete setting.[13]

The demonstration of how our results can be applied in the context of the case study

[13]Recall that, in Chapters 3–6, we have left the system specification parametric (except for in examples) and only committed ourselves to a particular system model, namely event systems. This means, our results in these chapters can be applied for all systems specified by event systems.

started in Section 7.4. In that section, we have specified the security requirements for a single DMWL process by an information flow property. This information flow property was defined using the concepts of *MAKS* (introduced in Chapter 3), i.e. we defined a view ($\mathcal{HI}^p$) and assembled a security predicate (SP^p) from two BSPs (*BSD* and $BSIA^{p^p}$). The resulting property specification (IFP^p) constitutes the security component of our security model (cf. Figure 3.1). The information flow property IFP^p differs from previously proposed ones, i.e., while performing the case study we have discovered a new information flow property. By this example, we have illustrated how easily information flow properties can be constructed in a goal-directed and application-driven way with the help of *MAKS*. By incorporating the newly discovered information flow property into our taxonomy (from Chapter 4), we have clarified the relations between this property and the known ones (cf. Remark 7.4.5).

The main result of our case study (Theorem 7.6.10) establishes a rigorous relation between strong security and the defined information flow property. More precisely, it states that if every program in a given collection is strongly secure then the resulting system satisfies IFP^P (the straightforward extension of IFP^p to sets of processes). In the proof of this result, we have exploited the techniques proposed in Chapters 5 and 6. Rather than verifying IFP^P for distributed programs directly (i.e. by investigating the set of possible traces), we have applied a compositionality theorem in this process. Since no suitable compositionality theorem existed (IFP^P is novel), we had to derive a new one. Following the approach from Chapter 6 greatly simplified this derivation (in comparison to the proof of Theorem 10 in [MS03a] that does not make use of these generic techniques). However, for deriving a suitable compositionality theorem, we had to strengthen IFP^P resulting in $CIFP^P$ (another novel information flow property). Without this modification, the derivation of the theorem would not have been possible. Apparently, strengthening an information flow property for deriving a compositionality result bears similarities, in spirit, with inductive theorem proving where it is often necessary to strengthen an induction formula for proving the step case. A prerequisite for applying a compositionality result is that all system components are known to satisfy the information flow property under consideration. Therefore, we verified that IFP^p holds for each strongly secure process with identifier p (cf. Theorem 7.5.1). Applying the unwinding techniques (from Chapter 5) simplified the proof of this result considerably (in comparison to the proof of Theorem 4 in [MS03a] which does not make use of unwinding). Note that we could easily apply unwinding although IFP^p is a new information flow property (and, hence, obviously no specialized unwinding results were known for it prior to our case study). By this example, we have illustrated the strengths of the approach that we proposed in Chapter 5: By representing IFP^p in *MAKS*, we have obtained an unwinding result "for free". After showing that IFP^p implies $CIFP^p$ for DMWL processes (cf. Theorem 7.6.16), we obtained from the fact that every strongly secure process satisfies IFP^p and our compositionality result that $CIFP^P$ holds for DMWL process pools that execute strongly secure programs. This was the result for which we were aiming with our case study because it increases the confidence in the strong security condition by relating it to a more established class of security properties. At this point, the reader is invited to go back to Definitions 7.2.5–7.2.9 in order to compare the definition of strong security with the information flow property IFP^P (cf. Definition 7.6.6). In our opinion, the information flow property is more intuitive and also reveals aspects that cannot be easily obtained from the definition of strong security. In particular, it is more amenable to comparisons with other security properties (cf. Remark 7.4.5). This is the benefit gained by our case study.

From a different perspective, our results establish a basis for mechanically verifying IFP^P

by applying the security type system (cf. Figure 7.5) to each program. If all programs are type correct then the overall system satisfies IFP^P (cf. Theorem 7.6.22).

The version of our case study in this chapter is complemented by two slightly different versions (due to Sabelfeld and myself) [MS01, MS03a]. Each of these versions emphasizes a different aspect. For example, the first version [MS01] established a rigorous connection between two independent approaches to security that were not "made to fit", i.e. *MAKS* and the original strong security condition for MWL [SS00] (cf. Section 7.7).

Although our case study is quite recent, the insights gained from it had already an impact on the area of language-based security. Namely, an earlier version of the case study inspired the extension of Sabelfeld and Sands's approach to a distributed setting [MS03a]. Moreover, a comparison of various combinations of communication primitives with different types of communication channels has clarified the advantages and disadvantages of these combinations wrt. security of the resulting programs. This has culminated in quite concrete advice for system architects (where to protect communication, e.g., by encryption or firewalls) and programmers (which primitives to use for secure communication on a given channel) [SM02].[14]

Summarizing, the case study that we have presented in this chapter demonstrates the applicability and usefulness of our results. This includes the construction of a suitable information flow property driven by the demands of a particular application (using *MAKS* as described in Chapter 3). It also includes the elaboration of a basic understanding of the constructed information flow property by relating it to more established ones (using the taxonomy in Chapter 4). Moreover, the verification of the information flow property for simple systems (individual processes) by unwinding illustrates the application of our unwinding techniques (from Chapter 5). Finally, the verification of the information flow property for more complex, distributed systems (collections of processes) illustrates the value of a compositionality result (derived using the approach proposed in Chapter 6). In short, we have shown that *MAKS* eases the application of information flow properties in concrete applications. In particular, *MAKS* provides a suitable basis for defining new information flow properties when they are needed and for easily building up the infrastructure (unwinding theorems and compositionality theorems), which is crucial for applying these properties in formal developments.

In this chapter, we have applied our results in the domain of language-based security. Many other application areas are possible like, e.g., access control mechanisms, security protocols, database security, operating system security, and multi-agent system security. We refer to the literature for other successful applications of noninterference-like information flow properties. For example, in [Rus92], noninterference has been used for the analysis of Bell/La Padula-like access control, including the precise identification of the so called reference monitor assumption. Another example is the use of information flow properties to express the requirements of security protocols [FGG97, FGM99, FGM00, DFG00]. Moreover, in [SRS+00] noninterference is used to specify the main security requirements of an operating system in a formal security model and the unwinding technique is used to verify that these requirements are satisfied. The application of the results presented in this thesis in the context of secure multi-agent systems is underway [HMS03, Sch03].

[14] Besides the communication primitives considered in this chapter, there is also a primitive for synchronous send in [SM02]. Moreover, besides low channels or high channels, which can be viewed as channels that are, respectively, openly accessible or not accessible at all (e.g. protected by a firewall), channels for encrypted communication are investigated in that article. The extension of the case study in this respect is a task for future work.

Chapter 8

Conclusion and Outlook

8.1 Conclusions

This thesis has been motivated by the observation that specifying and verifying security requirements is a highly nontrivial task in practice. Our experience from real-world projects and various case studies is that the main difficulty lies in getting the specification right. The systems involved are often quite complex and one has to take into account that secure systems have to operate correctly even if the environment is hostile. This setting leads to a high complexity of the specification task and, due to this complexity, ad hoc approaches to specify security requirements are too error prone. Hence, the elaboration and improvement of more general approaches is crucial. This thesis is a contribution in this direction.

The approach that we have pursued is that of information flow security. Information flow properties provide a very elegant basis for the specification of security requirements concerning confidentiality and integrity. In the past, numerous information flow properties had been proposed that are based on the same intuitive idea (namely that of noninterference) but, nevertheless, differ in quite subtle ways. The desire to understand the various information flow properties, their similarities, their differences, and the consequences of these differences more deeply has been a driving force in our studies. Another driving force has been the aim to ease the application of information flow properties in practice. In this respect, it was of crucial importance to construct a framework that supports a specifier in defining appropriate information flow properties when they are needed.

By splitting the well known information flow properties into more primitive pieces and then generalizing these pieces, we have obtained a collection of building blocks for information flow properties (i.e. the BSPs). This collection constitutes the core of our framework *MAKS*. In *MAKS*, information flow properties are represented by a pair consisting of a set of views (specifying the application-specific restrictions on the flow of information) and a security predicate (giving a meaning to these restrictions) where the security predicate is assembled from BSPs by logical conjunction. This modular representation provides the basis for a divide-and-conquer approach to the investigation of information flow properties: complex information flow properties are investigated by, firstly, investigating all BSPs from which they are assembled and, secondly, lifting the results for the BSPs over the assembling operation.

This divide-and-conquer approach has been applied for various purposes: Based on a taxonomy for BSPs, we have derived a taxonomy of information flow properties that clarifies the relationship between different properties. Moreover, based on unwinding results for BSPs,

we have derived unwinding theorems for more complex information flow properties. These theorems can be applied to simplify the verification of information flow properties (i.e. global requirements on sets of traces) by reducing the overall verification task to the verification of unwinding conditions (i.e. local requirements on individual transitions). Finally, based on compositionality results for BSPs, we have derived several compositionality theorems for information flow properties. These compositionality theorems can be applied to reduce the verification of the overall system to the verification of each system component, thereby making the verification of large and complex systems feasible. In each of these cases, the divide-and-conquer approach has simplified the derivation of our results considerably.

Another important advantage of our divide-and-conquer approach is that our results for known properties can be easily extended to novel properties: After representing a novel information flow property in *MAKS*, the integration of this property into our taxonomy becomes straightforward, an unwinding theorem for this property can be obtained easily, and compositionality theorems for this property can be derived by exploiting the corresponding results for BSPs. We have illustrated these advantages by several examples (e.g. for the novel properties *WFC*, *GNI**, and *FC**). In practice, this means that one may use the information flow property that is most suitable for the given application in the construction of a formal security model (rather than being restricted to properties for which an infrastructure consisting of unwinding theorems and compositionality theorems already exists). In particular, the use of novel information flow properties for this purpose has become feasible because, based on our contributions, the necessary infrastructure can be obtained with almost no effort.

The basic concepts of *MAKS* (views and BSPs) are, on the one hand, sufficiently specific for deriving useful results about them and, on the other hand, sufficiently general for representing the well known information flow properties. This combination distinguishes our framework from those previously proposed. The concepts underlying the frameworks of Focardi/Gorrieri [FG95] and Zakinthinos/Lee [Zak96, ZL97] are not specific enough to derive interesting results about them. As a consequence, unwinding theorems and compositionality theorems have to be derived from scratch and for each information flow property independently, which can become quite tedious. In contrast to this, the concepts underlying the framework by McLean [McL94a, McL96] are specific enough to derive compositionality results, but they are not expressive enough to represent all properties of interest (cf. Appendix B). For example, forward correctability, the perfect security property, and the information flow property that we used in the case study cannot be represented in this framework. Table 8.1 highlights how the various frameworks compare in these respects and Table 8.2 compares the results about information flow properties that have been obtained in these frameworks.

In a case study, we have applied our framework *MAKS* for deriving an information flow property that is adequate for modeling the security requirements of distributed, multi-threaded, imperative programs. We have elaborated a basic understanding of the derived property by relating it to the well-established information flow properties with the help of our taxonomy. Moreover, we have employed our unwinding techniques to simplify the proof that every process (executing a program that is strongly secure in the sense of Sabelfeld and Sands) satisfies the information flow property. Finally, we have presented a compositionality theorem that allowed us to conclude the security of distributed systems consisting of multiple processes from the security of each individual process. In summary, we have successfully applied results from all areas to which we have contributed in this thesis. Based on our experiences, we are confident that this case study can serve as a guideline for future applications of our results, also in other application domains than that of language-based security (also

cf. the paragraph on applications in the next section).

8.2 Further Work and Outlook

During our studies, we have noticed several open issues in the area of information flow security. Naturally, we have not tackled all of them, for reasons of time. Moreover, we have not described all of our contributions in the main chapters of this thesis. However, many of these results have been documented elsewhere in detail [Man00c, Man01a, Man01b, SM02, MS03b, HMS03]. In the remainder of this conclusion, we briefly sketch these contributions and also point out areas for future work that we think are important.

Refinement. Refining a specification means moving from an abstract specification to a more concrete one, e.g., by making design decisions or by adding technical details. Unfortunately, information flow properties are not preserved under trace refinement, in general. This fact, also known as the *refinement paradox* [Ros95], is a consequence of the fact that information flow properties are properties of sets of traces rather than properties of traces [McL94a]. Hence, after refining a given specification that has been verified to satisfy some information flow property, one has to make sure that this property still holds after the refinement. However, re-verifying information flow properties after each refinement step would lead to a tremendous overhead thereby rendering the use of refinement during stepwise development impractical. Therefore, it is desirable to have restricted forms of refinement under which information flow properties are preserved.

Building on previous work by Jacob [Jac88, Jac89], McLean [McL92, McL94a], and Roscoe et alter [RWW94, Ros95], we have proposed operators that can be used to refine specifications while preserving the information flow properties of interest [Man00c, Man01b]. These *refinement operators* take an abstract specification, the desired trace refinement (cf. Definition C.1.1), and information about the proof of the information flow property at the abstract level as parameters and return a refined specification. Our refinement operators are sound in the sense that they ensure that the given information flow property is indeed preserved. The basic idea is to exploit the proof at the abstract level in order to construct the refinement in a way that preserves the given property. For example, if the perfect security property has been verified for the abstract system specification by unwinding then the unwinding relation can be used to ensure that the perfect security property is preserved under refinement [Man01b]. Obviously, the refined specification cannot be equal to the desired refinement in all cases (due to the refinement paradox), which means that the refinement operators have to construct an approximation of the desired trace refinement. These approximations are quite close to the desired refinements and we have been able, in some cases, to show that the constructed refinements are optimal approximations. The integration of refinement operators into *MAKS* is underway but has not yet been entirely completed. Another valuable direction for the future is to extend refinement operators to other notions of refinement than trace refinement like, e.g., action refinement or data refinement (building on work in [GCS91]).

Intransitive Information Flow Policies. Rather than interpreting a noninterference statement $F \not\rightsquigarrow P$ by "information must not flow from F to P" (as we have done in this thesis), one sometimes might want to interprete it more liberally by "information must not flow from F to P *directly*". The difference between the two interpretations becomes apparent

if there is a security domain L with $F \rightsquigarrow_V L$ and $L \rightsquigarrow_V P$. Under the more liberal interpretation, information may flow from F to P via L while this is forbidden under the rigorous interpretation. The liberal interpretation is useful to model some form of *downgrading* or *sanitizing* of information (as we have illustrated in Example 3.3.5). Information flow properties that assume this interpretation are known as *intransitive noninterference*.[1]

Historically, the development of intransitive noninterference occurred independently from the extension of noninterference to nondeterministic systems. As a result of this, the proposed notions of intransitive noninterference either were applicable only to deterministic systems [HY87, Rus92, Pin95] or permitted only very limited nondeterminism [RG99]. In [Man01b], we have proposed an approach to deal with intransitive information flow that overcomes this limitation in the sense that it can be applied to arbitrary nondeterministic systems. A corresponding extension of *MAKS* has already been developed and it has been shown that unwinding theorems for the resulting (intransitive) information flow properties can be obtained from unwinding results for BSPs like in the transitive case (cf. Chapter 5). We think that a similar extension of our approach to derive compositionality results (Chapter 6) to the intransitive case will be possible — but this remains to be elaborated.

Other Variants of Information Flow Security. So far, we have assumed a possibilistic, trace-based system model. However, information flow properties have also been investigated in other system models, possibilistic ones as well as probabilistic ones.

Probabilistic information flow properties like the flow model in [McL90] or probabilistic noninterference as in [Gra91] aim at the detection of covert channels that, otherwise, could be exploited using concepts from information theory [WJ90, Sha48]. Sabelfeld and Sands have demonstrated how a probabilistic security property can be approximated by a possibilistic property [SS00] that constitutes a sound abstraction of the probabilistic property. How to abstract from the peculiarities of cryptographic primitives in a cryptographically-sound way has been elaborated by Pfitzmann, Schunter, Waidner, Steiner, and Backes [PSW00, Sch00b, PW01, Ste02, Bac02]. Heisel, Pfitzmann, and Santen have introduced the notion of confidentiality-preserving refinement [HPS01, SHP02] and have used it to abstract away from the probabilities in a model of encrypted communication (where the probability distributions come from the choice of the cryptographic keys). Abstracting from probabilities and the peculiarities of cryptographic primitives is quite attractive because the resulting possibilistic properties and systems are conceptually much simpler than the real ones. What the possibilities and limits of such abstractions are is a question that deserves further investigation. Another interesting question is to what extent these abstractions can be combined with the possibilistic information flow properties investigated in this thesis.

The notion of equality underlying Sabelfeld and Sands's system model is strong bisimulation. Other possibilistic semantics that have been used are ready sets semantics (by Ryan in [Rya91]), failure divergence semantics (by Roscoe et al. in [RWW94, Ros95]), and weak bisimulation (by Focardi/Gorrieri in [FG95]). We think that the concepts of *MAKS* could be adapted to these other semantics as well — but this remains to be shown. Another interesting direction is to identify mappings to move from one semantics to another. A first step has been

[1] Unfortunately, the term "intransitive noninterference" is misleading. Firstly, it is the interference relation $\rightsquigarrow_V$ that is not transitive rather than the noninterference relation $\not\rightsquigarrow$. Secondly, the important issue is how the noninterference relation is interpreted (liberally or rigorously) rather than whether the interference relation is transitive or not. However, for historic reasons, we stick to these established terms.

made in [MS01] where we have shown how a security property based on strong bisimulation can be expressed by an equivalent information flow property in a trace-based system model.

Tool Support. For applying formal methods in real world developments, the availability of mature tool support is essential. For example, editors that support the preferred specification languages are needed for constructing specifications, and theorem provers with a high degree of automation are needed for making formal verification feasible.

Rather than developing specialized tool support for information flow properties from scratch, our approach is to tailor general-purpose tools for formal methods to this setting. The tool on which we build is the Verification Support Environment (abbreviated by VSE) that has been developed at DFKI [HLS+96, HMR+98, AHL+00]. The integrated tool support provided by VSE includes editors, a graphical front end to navigate in structured specifications, an automatic static type checker for specifications, a generator that automatically determines the proof obligations for a given specification, theorem provers, and a management of change that ensures that formal developments always remain in a consistent state. In particular, VSE provides tool support for large-scale formal developments and it has been demonstrated in various case studies as well as real-world projects that VSE, indeed, is very suitable for this purpose. For simplifying the use of information flow properties in VSE, a front end to VSE is being built that simplifies the specification of event systems. Moreover, VSE-specifications for many building blocks and their unwinding conditions are provided in a library-like fashion such that the specifier can simply use the fundamental notions of *MAKS* without having to type in their VSE specifications [MS03b]. A prototypical implementation of this tool support has been successfully applied in a case study already [MSK+01]. To build a similar front end for the MAYA-system [AHMS00, AH03], a tool for the application of formal methods that is also developed at DFKI and that (in combination with the INKA system [AHMS99]) has a similar scope like VSE but is of a more experimental nature, would be a valuable goal for the future. To increase the degree of automation during the verification of information flow properties is the main objective of an ongoing cooperation with members of the Ωmega group at Saarland University. In this cooperation, it is investigated to what extent SAT-solvers and theorem provers, developed in the context of the Calculemus network, are suitable for the automatic verification of unwinding conditions with promising results in first experiments [Ben03].

Further Application Domains. Information flow properties have been successfully applied for several purposes, e.g., as a foundation for Bell/La Padula-like access control [Rus92], as a foundation for the verification of security protocols [FGG97, FGM99, FGM00, DFG00], and in formal security models for operating systems [SRS+00]. Our case study on language-based security has opened up another area of application for information flow properties, namely as a foundation for language-based security techniques. This application has already inspired extensions of the language-based techniques, for example, to distributed systems [MS03a, SM02]. This direction deserves further investigation. Another case study based on *MAKS* has been concerned with the protection of location information in cellular phones [MSK+01]. The application of *MAKS* in other application domains, including operating systems security and secure multi-agent systems is being investigated [HMS03, Sch03]. There are many exciting possibilities to apply our results and their exploitation has only just begun.

	framework by McLean	framework by Focardi and Gorrieri	framework by Zakinthinos and Lee	*MAKS*
sufficiently expressive		√	√	√
uniform system model	√	√	√	√
specific uniform concepts	*enforced*		*provided*	*enforced*
interesting and useful results about uniform concepts	√			√

Table 8.1: Comparison of expressiveness and uniformity

	framework by McLean			framework by Focardi and Gorrieri			framework by Zakinthinos and Lee			*MAKS*		
	represented	unwinding theorem	compositionality result	represented	unwinding theorem	compositionality result	represented	unwinding theorem	compositionality result	represented	unwinding theorem	compositionality result
	properties protecting occurrences and nonoccurrences of all high-level events											
SEP	√		√				√			√	√	√
NDO										√	√	√
*NDO**										√	√	√
PSP							√	√	√	√	√	√
	properties protecting occurrences of all high-level events											
NF	√		√	√		√	√	√	√	√	√	√
	properties protecting occurrences and nonoccurrences of high-level inputs											
FC				√		√	√	√		√	√	√
*FC**										√	√	√
GNI							√	√		√	√	√
*GNI**										√	√	√
IBGNI	√		√				√		√	√	√	
*IBGNI**	√		√							√	√	
WFC										√	√	√
IFP^P										√	√	√
	properties protecting occurrences of high-level inputs											
GNF	√		√	√			√	√	√	√	√	√

Table 8.2: Comparison of derived results about information flow properties. This table summarizes Tables 4.7, 5.3, and 6.2. The entries are explained in Sections 4.4, 5.7, and 6.8.

References

[Aga00] J. Agat. Transforming out Timing Leaks. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 40–53, 2000.

[AH03] S. Autexier and D. Hutter. Formal Software Development in MAYA. In *Festschrift in honour of Jörg H. Siekmann*. Springer, 2003. to appear.

[AHL+00] S. Autexier, D. Hutter, B. Langenstein, H. Mantel, G. Rock, A. Schairer, W. Stephan, R. Vogt, and A. Wolpers. VSE: Formal Methods Meet Industrial Needs. *Special Issue on Mechanized Theorem Proving for Technology Transfer of the STTT-Springer International Journal on Software Tools for Technology Transfer*, 3(1):66–77, 2000.

[AHMS99] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. System Description: INKA 5.0 – A Logic Voyager. In *Proceedings of the 16th International Conference on Automated Deduction, CADE-16*, volume 1632 of *LNAI*, pages 207–211, Trento, Italy, 1999.

[AHMS00] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an Evolutionary Formal Software-Development Using CASL. In *14th International Workshop on Algebraic Development Techniques, WADT'99, Selected Papers*, volume 1827 of *LNCS*, pages 73–88, 2000.

[AM98] S. Autexier and H. Mantel. Semantical Investigation of Simultaneous Skolemization for First-Order Sequent Calculus. Technical Report SEKI Report SR-98-05, Saarland University, Germany, 1998.

[AMS98] S. Autexier, H. Mantel, and W. Stephan. Simultaneous Quantifier Elimination. In *Proceedings of KI-98: Advances in Artificial Intelligence, 22nd Annual German Conference on Artificial Intelligence*, volume 1504 of *LNAI*, pages 141–152, Bremen, Germany, 1998.

[AS85] B. Alpern and F. B. Schneider. Defining Liveness. *Information Processing Letters*, 21:181–185, 1985. North-Holland.

[AS01] J. Agat and D. Sands. On Confidentiality and Algorithms. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 64–77, Oakland, CA, USA, 2001.

[Bac02] M. Backes. *Cryptographically Sound Analysis of Security Protocols.* PhD thesis, Technische Fakultät, Universität des Saarlandes, 2002.

[BC92] P. Bieber and F. Cuppens. A Logical View of Secure Dependencies. *Journal of Computer Security*, 1(1):99–129, 1992.

[BCC94] N. Boulahia-Cuppens and F. Cuppens. Asynchronous Composition and Required Security Conditions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 68–78, Oakland, CA, USA, 1994.

[Ben03] C. Benzmüller, editor. CALCULEMUS Project Report. SEKI Report, Saarland University, 2003. to appear.

[BN02] A. Banerjee and D. A. Naumann. Secure Information Flow and Pointer Confinement in a Java-like Language. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 253–270, Cape Breton, Nova Scotia, Canada, 2002.

[CC99] Common Criteria version for Information Technology and Security Evaluation, Version 2.1, August 1999. (aligned with ISO/IEC International Standard (IS) 15408).

[DD77] D. E. Denning and P. J. Denning. Certification of Programs for Secure Information Flow. *Communications of the ACM*, 20(7):504–513, 1977.

[Den76] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, 1976.

[DFG00] A. Durante, R. Focardi, and R. Gorrieri. A Compiler for Analysing Cryptographic Protocols Using Non-Interference. *ACM Transactions on Software Engineering and Methodology*, 9(4):488–528, 2000.

[FG95] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[FGG97] R. Focardi, A. Ghelli, and R. Gorrieri. Using Non Interference for the Analysis of Security Protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS Center, Rutgers University, 1997.

[FGM99] R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in Security Protocols as Non Interference. In *Proceedings of DERA/RHUL Workshop on Secure Architectures and Information Flow, ENTCS*, volume 32, 1999.

[FGM00] R. Focardi, R. Gorrieri, and F. Martinelli. Non Interference for the Analysis of Cryptographic Protocols. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, volume 1853 of *LNCS*, pages 354–372, Geneve, Switzerland, 2000.

[FLR77] R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving Multilevel Security of a System Design. In *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*, pages 57–65, 1977.

[Fol87] S. N. Foley. A Universal Theory of Information Flow. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 116–122, Oakland, CA, USA, 1987.

[FR02] R. Focardi and S. Rossi. Information Flow Security in Dynamic Contexts. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 307–319, Cape Breton, Nova Scotia, Canada, 2002.

[GCS91] J. Graham-Cumming and J. W. Sanders. On the Refinement of Non-interference. In *Proceedings of the 4th IEEE Computer Security Foundations Workshop*, pages 35–42, Franconia, NH, USA, 1991.

[GM82] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, CA, USA, 1982.

[GM84] J. A. Goguen and J. Meseguer. Inference Control and Unwinding. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 75–86, Oakland, CA, USA, 1984.

[GN88] J. D. Guttman and M. E. Nadel. "What Needs Securing?". In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 34–57, 1988.

[Gra91] J. W. Gray. Toward a Mathematical Foundation for Information Flow Security. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 21–34, Oakland, CA, USA, 1991.

[He89] J. He. Process Simulation and Refinement. *Journal of Formal Aspects of Computing Science*, 1:229–241, 1989.

[HHS87] C. A. R. Hoare, J. He, and J. W. Sanders. Prespecification in Data Refinement. *Information Processing Letters*, 25:71–76, 1987.

[HLS+96] D. Hutter, B. Langenstein, C. Sengler, J. Siekmann, W. Stephan, and A. Wolpers. Verification Support Environment (VSE). *Journal of High Integrity Systems*, 1(6):523–530, 1996.

[HMR+98] D. Hutter, H. Mantel, G. Rock, W. Stephan, A. Wolpers, M. Balser, W. Reif, G. Schellhorn, and K. Stenzel. VSE: Controlling the Complexity in Formal Software Developments. In *Proceedings of International Workshop on Applied Formal Methods - FM-Trends*, volume 1641 of *LNCS*, pages 351–358, Boppard, Germany, 1998.

[HMS03] D. Hutter, H. Mantel, and A. Schairer. Informationsflusskontrolle als Grundlage für die Sicherheit von Multiagentensystemen. *PIK, Praxis der Informationsverarbeitung und Kommunikation*, 26(1):39–47, 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice-Hall, 1985.

[HPS01] M. Heisel, A. Pfitzmann, and T. Santen. Confidentiality-Preserving Refinement. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 295–305, Cape Breton, Nova Scotia, Canada, 2001.

[HR98] N. Heintze and J. G. Riecke. The SLam Calculus: Programming with Secrecy and Integrity. In *Proceedings of the 25th ACM Symposium on Principles of Programming Languages*, pages 365–377, 1998.

[HY87] J. T. Haigh and W. D. Young. Extending the Noninterference Version of MLS for SAT. *IEEE Transactions on Software Engineering*, SE-13(2):141–150, 1987.

[ITS91] Information Technology Security Evaluation Criteria, Version 1.2, June 1991. Office for Official Publications of the European Communities.

[Jac88] J. Jacob. Security Specifications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 14–23, Oakland, CA, USA, 1988.

[Jac89] J. Jacob. On the Derivation of Secure Components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, Oakland, CA, USA, 1989.

[Jac90] J. Jacob. Categorising Non-interference. In *Proceedings of the Computer Security Workshop*, pages 44–50, Franconia, NH, USA, 1990.

[Jon89] B. Jonsson. On Decomposing and Refining Specifications of Distributed Systems. In *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *LNCS*, pages 361–387, Mook, The Netherlands, 1989.

[Jon91] B. Jonsson. Simulations between Specifications of Distributed Systems. In *Proceedings of CONCUR'91*, volume 527 of *LNCS*, pages 346–360, Amsterdam, The Netherlands, 1991.

[Jon01] C. B. Jones. Thinking Tools for the Future of Computer Science. In *Informatics, 10 Years Back. 10 Years Ahead*, volume 2000 of *LNCS*, pages 112–130, 2001.

[Jos88] M. B. Josephs. A State-Based Approach to Communicating Processes. *Distributed Computing*, 3:9–18, 1988.

[JT88] D. M. Johnson and F. J. Thayer. Security and the Composition of Machines. In *Proceedings of the Computer Security Foundations Workshop*, pages 72–89, Franconia, NH, USA, 1988.

[Jür00] J. Jürjens. Secure Information Flow for Concurrent Processes. In *Proceedings of the International Conference on Concurrency Theory, Concur 2000*, volume 1877 of *LNCS*, pages 395–409, 2000.

[KM03] C. Kreitz and H. Mantel. A Matrix Characterization for Multiplicative Exponential Linear Logic. *Journal of Automated Reasoning*, 2003. accepted for publication.

[KMOS97] C. Kreitz, H. Mantel, J. Otten, and S. Schmitt. Connection-Based Proof Construction in Linear Logic. In *Proceedings of the 14th International Conference on Automated Deduction, CADE-14*, volume 1249 of *LNAI*, pages 207–221, Townsville, Australia, 1997.

[KS93] N. Klarlund and F. B. Schneider. Proving Nondeterministically Specified Safety Properties using Progress Measures. *Information and Computation*, 107(1):151–170, 1993.

[Lam94] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[Lau01] P. Laud. Semantics and Program Analysis of Computationally Secure Information Flow. In *Proceedings of European Symposium on Programming*, volume 2028 of *LNCS*, pages 77–91, Genova, Italy, 2001.

[LV95] N. Lynch and F. Vaandrager. Forward and Backward Simulations, Part I: Untimed Systems. *Information and Computation*, 121(2):214–233, 1995. Also, Technical Memo, MIT/LCS/TM-486.b, Laboratory for Computer Science, Massachusetts Institute of Technology, August 1994.

[Man98] H. Mantel. Developing a Matrix Characterization for MELL. Technical Report DFKI Report RR-98-03, DFKI, Kaiserslautern, Germany, 1998.

[Man00a] H. Mantel. A New Framework for Possibilistic Security - A Summary. Abstract presented at IEEE Symposium on Security and Privacy, May 14–17 2000.

[Man00b] H. Mantel. An Approach to Information Flow Control. Abstract presented at Dagstuhl Seminar 00501, Security through Analysis and Verification, Dagstuhl Report No. 294, December 10-15 2000.

[Man00c] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, 2000.

[Man00d] H. Mantel. Unwinding Possibilistic Security Properties. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *LNCS*, pages 238–254, Toulouse, France, 2000.

[Man01a] H. Mantel. Information Flow Control and Applications – Bridging a Gap. In *Proceedings of FME 2001: Formal Methods for Increasing Software Productivity*, volume 2021 of *LNCS*, pages 153–172, Berlin, Germany, 2001.

[Man01b] H. Mantel. Preserving Information Flow Properties under Refinement. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 78–91, Oakland, CA, USA, 2001.

[Man02] H. Mantel. On the Composition of Secure Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–104, Berkeley, CA, USA, 2002.

[MC92] I. S. Moskowitz and O. L. Costich. A classical Automata Approach to Noninterference Type Problems. In *Proceedings of the 5th IEEE Computer Security Foundations Workshop*, pages 2–8, Franconia, NH, USA, 1992.

[McC87] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 161–166, Oakland, CA, USA, 1987.

[McC90] D. McCullough. A Hookup Theorem for Multilevel Security. *IEEE Transactions on Software Engineering*, 16(6), 1990.

[McL90] J. D. McLean. Security Models and Information Flow. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 180–187, Oakland, CA, USA, 1990.

[McL92] J. D. McLean. Proving Noninterference and Functional Correctness using Traces. *Journal of Computer Security*, 1(1):37–57, 1992.

[McL94a] J. D. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 79–93, Oakland, CA, USA, 1994.

[McL94b] J. D. McLean. Security Models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc., 1994.

[McL96] J. D. McLean. A General Theory of Composition for a Class of "Possibilistic" Security Properties. *IEEE Transaction on Software Engineering*, 22(1):53–67, 1996.

[MG00a] H. Mantel and F. Gärtner. A Case Study in the Mechanical Verification of Fault Tolerance. In *Proceedings of Special Track on Verification, Validation and System Certification at 13th International Florida Artificial Intelligence Research Society Conference 2000, FLAIRS-2000*, pages 341–345, Orlando, Florida, US, 2000.

[MG00b] H. Mantel and F. Gärtner. A Case Study in the Mechanical Verification of Fault Tolerance. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 12(4):473–488, 2000.

[Mil90] J. K. Millen. Hookup Security for Synchronous Machines. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 84–90, Oakland, CA, USA, 1990.

[Mil94] J. K. Millen. Unwinding Forward Correctability. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 2–10, Franconia, NH, USA, 1994.

[MK98] H. Mantel and C. Kreitz. A Matrix Characterization for MELL. In *Proceedings of Logics in Artificial Intelligence, European Workshop, JELIA '98*, volume 1489 of *LNAI*, pages 169–183, Dagstuhl, Germany, 1998.

[MO99] H. Mantel and J. Otten. linTAP: A Tableau Prover for Linear Logic. In *Proceedings of Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX'99)*, volume 1617 of *LNAI*, pages 217–231, Saratoga Springs, NY, USA, 1999.

[MS01] H. Mantel and A. Sabelfeld. A Generic Approach to the Security of Multi-threaded Programs. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 126–142, Cape Breton, Nova Scotia, Canada, 2001.

[MS03a] H. Mantel and A. Sabelfeld. A Unifying Approach to the Security of Distributed and Multi-threaded Programs. *Journal of Computer Security*, 2003. to appear.

[MS03b] H. Mantel and A. Schairer. Exploiting Generic Aspects of Security Models in Formal Developments. In *Festschrift in honour of Jörg H. Siekmann*. Springer, 2003. to appear.

[MSK+01] H. Mantel, A. Schairer, M. Kabatnik, M. Kreutzer, and A. Zugenmaier. Using Information Flow Control to Evaluate Access Protection of Location Information in Mobile Communication Networks. Technical Report 159, Computer Science Department, University of Freiburg, August 2001.

[Neu95] P. G. Neumann. *Computer Related Risks*. Addison-Wesley, ACM Press, 1995.

[O'H90] C. O'Halloran. A Calculus of Information Flow. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, pages 147–159, Toulouse, France, 1990.

[O'H92] C. O'Halloran. Refinement and Confidentiality. In *Proceedings of the 5th Refinement Workshop*, Workshops in Computing, pages 119–139, London, 1992.

[PHW02] A Di Pierro, C. Hankin, and H. Wiklicky. Approximate Non-Interference. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 3–17, Cape Breton, Nova Scotia, Canada, 2002.

[Pin95] S. Pinsky. Absorbing Covers and Intransitive Non-Interference. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 102–113, Oakland, CA, USA, 1995.

[PSW00] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic Security of Reactive Systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 32, 2000.

[PW01] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–200, Oakland, CA, USA, 2001.

[PWK96] R. V. Peri, W. A. Wulf, and D. M. Kienzle. A Logic of Composition for Information Flow Predicates. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 82–93, Kenmare, County Kerry, Ireland, 1996.

[RG99] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 228–238, Mordano, Italy, 1999.

[Ros95] A. W. Roscoe. CSP and Determinism in Security Modelling. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 114–127, Oakland, CA, USA, 1995.

[RS99] P. Y. A. Ryan and S. A. Schneider. Process Algebra and Non-interference. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 214–227, Mordano, Italy, 1999.

[Rus81a] J. M. Rushby. Design and Verification of Secure Systems. In *Proceedings of the Eighth ACM Symposium on Operating System Principles*, pages 12–21, Asimolar, CA, USA, 1981.

[Rus81b] J. M. Rushby. Verification of Secure Systems. Technical Report 166, University of Newcastle upon Tyne, UK, August 1981.

[Rus92] J. M. Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical Report CSL-92-02, SRI International, 1992.

[Rus94] J. M. Rushby. Critical System Properties: Survey and Taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.

[Rus99] J. M. Rushby. Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance. Technical report, SRI International, Menlo Park, CS, USA, March 1999.

[RW95] A. W. Roscoe and L. Wulf. Composing and Decomposing Systems under Security Properties. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 9–15, Kenmare, Ireland, 1995.

[RWW94] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-interference through Determinism. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, volume 875 of *LNCS*, pages 33–53, Brighton, UK, 1994.

[Rya91] P. Y. A. Ryan. A CSP Formulation of Non-Interference and Unwinding. *Cipher*, pages 19–30, Winter 1991. presented at CSFW'90.

[Sch99] F. B. Schneider, editor. *Trust in Cyberspace.* National Academy Press, Washington, D.C., USA, 1999. Comittee on Information Systems Trustworthyness, Computer Science Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications, National Research Council.

[Sch00a] F. B. Schneider. Enforceable security policies. *Information and System Security*, 3(1):30–50, 2000.

[Sch00b] M. Schunter. *Optimistic Fair Exchange.* PhD thesis, Technische Fakultät, Universität des Saarlandes, 2000.

[Sch01] S. A. Schneider. May Testing, Non-interference, and Compositionality. Technical Report CSD-TR-00-02, Royal Holloway, University of London, January 2001.

[Sch03] A. Schairer. Towards Using Possibilistic Information Flow Control to Design Secure Multiagent Systems. In *Proceedings of the First International Conference on Security in Pervasive Computing*, 2003. to appear.

[Sha48] C. E. Shannon. The Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27:379–423, 1948.

[SHP02] T. Santen, M. Heisel, and A. Pfitzmann. Confidentiality-Preserving Refinement is Compositional – Sometimes. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, volume 20502 of *LNCS*, pages 194–211, Zürich, Switzerland, 2002.

[SM02] A. Sabelfeld and H. Mantel. Static Confidentiality Enforcement for Distributed Programs. In *Proceedings of the 9th International Static Analysis Symposium, SAS'02*, volume 2477 of *LNCS*, pages 376–394, Madrid, Spain, 2002.

[SM03] A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *IEEE Journal on Selected Areas in Communication*, 21(1):5–19, 2003.

[Som96] I. Sommerville. *Software Engineering.* Addison-Wesley, 5th edition, 1996.

[SRS+00] G. Schellhorn, W. Reif, A. Schairer, P. Karger, V. Austel, and D. Toll. Verification of a Formal Security Model for Multiapplicative Smart Cards. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *LNCS*, pages 17–36, Toulouse, France, 2000.

[SS00] A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200–215, Cambridge, UK, 2000.

[Ste02] M. Steiner. *Secure Group Key Exchange.* PhD thesis, Technische Fakultät, Universität des Saarlandes, 2002.

[Sut86] D. Sutherland. A Model of Information. In *Proceedings of the 9th National Computer Security Conference*, 1986.

[SV98] G. Smith and D. Volpano. Secure Information Flow in a Multi-threaded Imperative Language. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 355–364, San Diego, California, 1998.

[VSI96] D. Volpano, G. Smith, and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, 4(3):1–21, 1996.

[WJ90] J. T. Wittbold and D. M. Johnson. Information Flow in Nondeterministic Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 144–161, Oakland, CA, USA, 1990.

[Zak96] A. Zakinthinos. *On the Composition of Security Properties.* PhD thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto, 1996.

[ZL95] A. Zakinthinos and E. S. Lee. The Composability of Non-Interference. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 2–8, Kenmare, Ireland, 1995.

[ZL96] A. Zakinthinos and E. S. Lee. How and Why Feedback Composition Fails. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 95–101, Kenmare, County Kerry, Ireland, 1996.

[ZL97] A. Zakinthinos and E. S. Lee. A General Theory of Security Properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, CA, USA, 1997.

[ZL98] A. Zakinthinos and E. S. Lee. Composing Secure Systems that have Emergent Properties. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 117–122, Rockport, Massachusetts, 1998.

[ZM02] S. Zdancewic and A. C. Myers. Secure Information Flow via Linear Continuations. *Higher Order and Symbolic Computation*, 15(2–3):209–234, 2002.

Index

Appendix A

Details on the Representation of Known Information Flow Properties

A.1 Representation Lemma for *IBGNI*

Lemma A.1.1. We have the following implications:

$$IBGNI(ES) \Rightarrow D_{\mathcal{HI}}(Tr) \tag{A.1}$$

$$IBGNI(ES) \Rightarrow I_{\mathcal{HI}}(Tr) \tag{A.2}$$

$$(R_{\mathcal{HI}}(Tr) \wedge I_{\mathcal{HI}}(Tr)) \Rightarrow IBGNI(ES) \tag{A.3}$$

$\diamond$

Proof. For proving $D_{\mathcal{HI}}(Tr)$ in (A.1), let $\alpha, \beta \in E^*$ and $hi \in H \cap I$ be arbitrary such that $\beta.\langle hi\rangle.\alpha \in Tr$ and $\alpha|_{H\cap I} = \langle\rangle$. Applying Definition 4.2.4 with $\tau_l = \beta.\langle hi\rangle.\alpha$, $t_{hi} = \beta|_{H\cap I}$, and $t = (\beta.\alpha)|_{L\cup(H\cap I)}$ yields that there is a trace $\tau' \in Tr$ with $\tau'|_{L\cup(H\cap I)} = (\beta.\alpha)|_{L\cup(H\cap I)}$. The trace τ' can be split into two subsequences $\alpha', \beta' \in E^*$ such that $\beta'.\alpha' = \tau'$, $\alpha'|_L = \alpha|_L$, $\alpha'|_{H\cap I} = \langle\rangle$, and $\beta'|_{L\cup(H\cap I)} = \beta|_{L\cup(H\cap I)}$. Consequently, $D_{\mathcal{HI}}(Tr)$ holds.

For proving $I_{\mathcal{HI}}(Tr)$ (A.2), let $\alpha, \beta \in E^*$ and $hi \in H \cap I$ be arbitrary such that $\beta.\alpha \in Tr$ and $\alpha|_{H\cap I} = \langle\rangle$. Applying Definition 4.2.4 with $\tau_l = \beta.\alpha$, $t_{hi} = (\beta.\langle hi\rangle)|_{H\cap I}$, and $t = (\beta.\langle hi\rangle.\alpha)|_{L\cup(H\cap I)}$ yields that there is a trace $\tau' \in Tr$ with $\tau'|_{L\cup(H\cap I)} = (\beta.\langle hi\rangle.\alpha)|_{L\cup(H\cap I)}$. The trace τ' can be split into subsequences $\alpha', \beta' \in E^*$ such that $\beta'.\langle hi\rangle.\alpha' = \tau'$, $\alpha'|_L = \alpha|_L$, $\alpha'|_{H\cap I} = \langle\rangle$, and $\beta'|_{L\cup(H\cap I)} = \beta|_{L\cup(H\cap I)}$. Consequently, $I_{\mathcal{HI}}(Tr)$ holds.

For proving (A.3), assume $R_{\mathcal{HI}}(Tr)$ and $I_{\mathcal{HI}}(Tr)$. We prove $IBGNI(ES)$ by induction on the length of t_{hi}. In the base case, we have $t_{hi} = \langle\rangle$. The implication in the definition of *IBGNI* holds because of $R_{\mathcal{HI}}(Tr)$ ($t = \tau_l|_L$ is the only choice possible). In the step case, we assume that the implication (in Definition 4.2.4) holds for all t^*_{hi} with length smaller than n and show that it also holds for t_{hi} with length n. Let $t_{hi} = \gamma.\langle hi\rangle$ where $\gamma \in (H \cap I)^*$ and $hi \in H \cap I$ are arbitrary. Let $\tau_l \in Tr$ and $t \in E^*$ be arbitrary with $t \in interleaving(t_{hi}, \tau_l|_L)$. Choose $t_1, t_2 \in E^*$ such that $t = t_1.\langle hi\rangle.t_2$ and $t_2|_{H\cap I} = \langle\rangle$. According to our induction assumption, there are traces $\alpha, \beta \in E^*$ such that $\beta.\alpha \in Tr$, $\alpha|_L = t_2|_L$, $\alpha|_{H\cap I} = \langle\rangle$, and $\beta|_{L\cup(H\cap I)} = t_1|_{L\cup(H\cap I)}$ (use Definition 4.2.4 with τ_l as is, $t_{hi} = \gamma$, and the appropriate interleaving). Since $I_{\mathcal{HI}}(Tr)$ holds, there are $\alpha', \beta' \in E^*$ such that $\beta'.\langle hi\rangle.\alpha' \in Tr$, $\alpha'|_L = \alpha|_L$, $\alpha'|_{H\cap I} = \langle\rangle$, and $\beta'|_{L\cup(H\cap I)} = \beta|_{L\cup(H\cap I)}$. With the choice $\tau' = \beta'.\langle hi\rangle.\alpha'$ the proposition holds ($\tau'|_{L\cup(H\cap I)} = (\beta'.\langle hi\rangle.\alpha')|_{L\cup(H\cap I)} = t_1.\langle hi\rangle.t_2 = t$). Consequently, $IBGNI(ES)$ holds. $\square$

A.2 Representation Lemma for *FC*

Lemma A.2.1. For $\Gamma_{FC} = (I, \emptyset, I)$, we have the following implications:

$$FC(ES) \Rightarrow BSD_{\mathcal{HI}}(Tr) \tag{A.4}$$

$$FC(ES) \Rightarrow BSI_{\mathcal{HI}}(Tr) \tag{A.5}$$

$$FC(ES) \Rightarrow FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \tag{A.6}$$

$$FC(ES) \Rightarrow FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \tag{A.7}$$

$$\left(\begin{array}{c} BSD_{\mathcal{HI}}(Tr) \wedge BSI_{\mathcal{HI}}(Tr) \\ \wedge FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \wedge FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr) \end{array}\right) \Rightarrow FC(ES) \tag{A.8}$$

◇

Proof. Implications (A.4) and (A.5) follow, respectively, from the second and first conjunct in Definition 4.2.10 if $[\langle li \rangle]$ is replaced by $\langle\rangle$ ($t_1 = \beta$, $t_2 = \alpha$, $\alpha' = t_3$).

Implications (A.6) and (A.7) follow, respectively, from the second and first conjunct in Definition 4.2.10 if $[\langle li \rangle]$ is replaced by $\langle li \rangle$ ($t_1 = \beta$, $t_2 = \alpha$, $\alpha' = t_3$).

For proving $FC(ES)$ in (A.8), assume $t_1, t_2 \in E^*$, $hi \in HI$, and $li \in L \cap I$. The first conjunct in the definition of FC follows from $BSI_{\mathcal{HI}}(Tr)$ (for $[\langle li \rangle] = \langle\rangle$) and $FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$ (for $[\langle li \rangle] = \langle li \rangle$). The second conjunct follows from $BSD_{\mathcal{HI}}(Tr)$ (for $[\langle li \rangle] = \langle\rangle$) and $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$ (for $[\langle li \rangle] = \langle li \rangle$). □

A.3 Representation Lemma for *NDO*

Lemma A.3.1. For a set $UI \subseteq I$ of user inputs and the function ρ_{UI} defined by $\rho_{UI}(V, N, C) = C \cup (V \cap UI)$, we have the following implications:

$$NDO(ES) \Rightarrow SD_{\mathcal{H}}(Tr) \tag{A.9}$$

$$NDO(ES) \Rightarrow SIA^{\rho_{UI}}_{\mathcal{H}}(Tr) \tag{A.10}$$

$$R_{\mathcal{H}}(Tr) \wedge IA^{\rho_{UI}}_{\mathcal{H}}(Tr) \Rightarrow NDO(ES) \tag{A.11}$$

◇

Proof. For proving $SD_{\mathcal{H}}(Tr)$ in (A.9), let $\alpha, \beta \in E^*$ and $h \in H$ be arbitrary such that $\beta.\langle h \rangle.\alpha \in Tr$ and $\alpha|_H = \langle\rangle$ (cf. Definition 3.4.26). From $\beta \in Tr$ and $total(ES, I)$, we obtain that $\beta.(\alpha|_{L \cap UI}) \in Tr$ holds. By applying the definition of NDO for $\tau_l = \beta.\langle h \rangle.\alpha$ and $\tau_{hlui} = \beta.(\alpha|_{L \cap UI})$, we obtain that $\beta.\alpha \in Tr$ holds because $\tau_l \in Tr$, $\tau_{hlui} \in Tr$, $(\beta.\alpha)|_L = (\beta.\langle h \rangle.\alpha)|_L = \tau_l|_L$, and $(\beta.\alpha)|_{H \cup (L \cap UI)} = (\beta.\alpha|_{L \cap UI})|_{H \cup (L \cap UI)} = \tau_{hlui}|_{H \cup (L \cap UI)}$. Hence, $SD_{\mathcal{H}}(Tr)$ holds.

For proving $SIA^{\rho_{UI}}_{\mathcal{H}}(Tr)$ in (A.10), let $\alpha, \beta, \gamma \in E^*$ and $h \in H$ be arbitrary such that $\beta.\alpha \in Tr$, $\alpha|_H = \langle\rangle$, $\gamma.\langle h \rangle \in Tr$, and $\gamma|_{H \cup (L \cap UI)} = \beta|_{H \cup (L \cap UI)}$ (cf. Definition 3.4.26). From $\gamma.\langle h \rangle \in Tr$ and $total(ES, I)$, we obtain that $\gamma.\langle h \rangle.(\alpha|_{L \cap UI}) \in Tr$ holds. By applying the definition of NDO for $\tau_l = \beta.\alpha$ and $\tau_{hlui} = \gamma.h.(\alpha|_{L \cap UI})$, we obtain that $\beta.\langle h \rangle.\alpha \in Tr$ because $\tau_l \in Tr$, $\tau_{hlui} \in Tr$, $(\beta.\langle h \rangle.\alpha)|_L = (\beta.\alpha)|_L = \tau_l|_L$, and $(\beta.\langle h \rangle.\alpha)|_{H \cup (L \cap UI)} = (\gamma.\langle h \rangle.\alpha|_{L \cap UI})|_{H \cup (L \cap UI)} = \tau_{hlui}|_{H \cup (L \cap UI)}$. Consequently, $SIA^{\rho_{UI}}_{\mathcal{H}}(Tr)$ holds.

We prove $NDO(ES)$ in (A.11) by induction on the length of $t|_H$. In the base case, $t|_H = \langle\rangle$ holds. Let $\tau_l, \tau_{hlui} \in Tr$ be arbitrary with $t|_L = \tau_l|_L$ and $t|_{H \cup (L \cap UI)} = \tau_{hlui}|_{H \cup (L \cap UI)}$. From $t|_H = \langle\rangle$, we obtain that $t = \tau_l|_L$ holds and $t \in Tr$ follows from $R_{\mathcal{H}}(Tr)$. In the step case, $t|_H$ has length $n > 0$. The induction assumption is that the implication in Definition 4.2.14 holds for all $t^* \in E^*$ for which the length of $t^*|_H$ is smaller than n. We have to show

that the implication also holds for t. Let $\tau_l, \tau_{hlui} \in Tr$ be arbitrary with $\tau_l|_L = t|_L$ and $\tau_{hlui}|_{H\cup(L\cap UI)} = t|_{H\cup(L\cap UI)}$. Let $t_1, t_2 \in E^*$ be the subsequences of t, respectively, before and after the last occurrences of a high-level event h and let $\gamma, \delta \in E^*$ be the corresponding subsequences of τ_{hlui}, i.e. $t = t_1.\langle h\rangle.t_2$, $t_2|_H = \langle\rangle$, $\tau_{hlui} = \gamma.\langle h\rangle.\delta$, and $\delta|_H = \langle\rangle$. We apply the induction hypothesis with $\tau_l^* = \tau_l$, $\tau_{hlui}^* = \gamma.(\delta|_{L\cap UI})$, $t^* = t_1.t_2$ and obtain $t_1.t_2 \in Tr$ because $\tau_l^* \in Tr$, $\tau_{hlui}^* \in Tr$ (follows from $\gamma \in Tr$ and $total(ES, I)$), $t^*|_L = \tau_l|_L$, and $t^*|_{H\cup(L\cap UI)} = \tau_{hlui}^*|_{H\cup(L\cap UI)}$. Since $t_1.t_2 \in Tr$, $t_2|_H = \langle\rangle$, $\gamma.\langle h\rangle \in Tr$, $\gamma|_{H\cup(L\cap UI)} = t_1|_{H\cup(L\cap UI)}$ (follows from $(\gamma.\langle h\rangle.\delta)|_{H\cup(L\cap UI)} = \tau_{hlui}|_{H\cup(L\cap UI)}$), and $IA_{\mathcal{H}}^{\rho_{UI}}(Tr)$, we obtain $t_1.\langle h\rangle.t_2 \in Tr$ (view $\mathcal{H}$ permits no corrections), which means $t \in Tr$ ($t = t_1.\langle h\rangle.t_2$). Consequently, $NDO(ES)$ holds. □

A.4 Representation Lemma for *NF*

Lemma A.4.1. We have the following implications:

$$\begin{aligned} NF(ES) &\Rightarrow SR_{\mathcal{H}}(Tr) \\ R_{\mathcal{H}}(Tr) &\Rightarrow NF(ES) \end{aligned} \qquad \diamond$$

Proof. The implications follow immediately from the definitions of *SR*, *R*, and *NF* (Definitions 3.4.26, 3.4.1, and 4.2.19). □

A.5 Representation Lemma for *GNF*

Lemma A.5.1. We have the following implications:

$$\begin{aligned} GNF(ES) &\Rightarrow R_{\mathcal{HI}}(Tr) \\ R_{\mathcal{HI}}(Tr) &\Rightarrow GNF(ES) \end{aligned} \qquad \diamond$$

Proof. The implications follow immediately from Definitions 3.4.1 and 4.2.21. □

A.6 Representation Lemma for *SEP*

Lemma A.6.1. For the function ρ_C defined by $\rho_C(V, N, C) = C$, we have the following implications:

$$SEP(ES) \Rightarrow SD_{\mathcal{H}}(Tr) \tag{A.12}$$

$$SEP(ES) \Rightarrow SIA_{\mathcal{H}}^{\rho_C}(Tr) \tag{A.13}$$

$$R_{\mathcal{H}}(Tr) \wedge IA_{\mathcal{H}}^{\rho_C}(Tr) \Rightarrow SEP(ES) \tag{A.14}$$

$\diamond$

Proof. For proving $SD_{\mathcal{H}}(Tr)$ in (A.12), let $\alpha, \beta \in E^*$ and $h \in H$ be arbitrary such that $\beta.\langle h\rangle.\alpha \in Tr$ and $\alpha|_H = \langle\rangle$. $\beta.\alpha$ is an interleaving of $\beta|_H$ and $(\beta.\langle h\rangle.\alpha)|_L$. Applying the definition of *SEP* for $\tau_l = \beta.\langle h\rangle.\alpha$ and $\tau_h = \beta$ yields $\beta.\alpha \in Tr$. Consequently, $SD_{\mathcal{H}}(Tr)$ holds.

For proving $SIA_{\mathcal{H}}^{\rho_C}(Tr)$ in (A.13), let $\alpha, \beta, \gamma \in E^*$ and $h \in H$ be arbitrary such that $\beta.\alpha \in Tr$, $\alpha|_H = \langle\rangle$, $\gamma.\langle h\rangle \in Tr$, and $\gamma|_H = \beta|_H$. $\beta.\langle h\rangle.\alpha$ is an interleaving of $(\gamma.\langle h\rangle)|_H$ and $(\beta.\alpha)|_L$. Applying the definition of *SEP* (for $\tau_l = \beta.\alpha$ and $\tau_h = \gamma.\langle h\rangle$) yields $\beta.\langle h\rangle.\alpha \in Tr$. Consequently, $SIA_{\mathcal{H}}^{\rho_C}(Tr)$ holds.

We prove $SEP(ES)$ in (A.14) by induction on the length of $\tau_h|_H$. In the base case, $\tau_h|_H = \langle\rangle$ holds. Let $\tau_l \in Tr$ and $t \in interleaving(\tau_h|_H, \tau_l|_L)$ be arbitrary. From $\tau_h|_H = \langle\rangle$, we obtain that $t = \tau_l|_L$ holds. $t \in Tr$ follows from $R_{\mathcal{H}}(Tr)$. Thus, $t \in \{\tau \in Tr \mid \tau|_L = \tau_l|_L\}$

holds. In the step case, $\tau_h|_H$ has length $n > 0$ and we assume that the subset relation (in Definition 4.2.23) holds for all τ_h^* for which the length of $\tau_h^*|_H$ is smaller than n. We show that the relation also holds for τ_h. Let $\tau_l \in Tr$ and $t \in interleaving(\tau_h|_H, \tau_l|_L)$ be arbitrary. Let $t_1, t_2 \in E^*$ be the subsequences of t, respectively, before and after the last occurrences of a high-level event h and let $\gamma, \delta \in E^*$ be the corresponding subsequences of τ_h, i.e. $t = t_1.\langle h\rangle.t_2$, $t_2|_H = \langle\rangle$, $\tau_h = \gamma.\langle h\rangle.\delta$, and $\delta|_H = \langle\rangle$. We apply the induction hypothesis with $\tau_l^* = \tau_l$, $\tau_h^* = \gamma$, $t^* = t_1.t_2$ and obtain $t_1.t_2 \in Tr$ because $\tau_l^* \in Tr$, $\tau_h^* \in Tr$, and $t_1.t_2 \in interleaving(\tau_h^*|_H, \tau_l^*|_L)$ hold. Since $t_1.t_2 \in Tr$, $t_2|_H = \langle\rangle$, $\gamma.\langle h\rangle \in Tr$, $\gamma|_H = t_1|_H$, and $IA_{\mathcal{H}}^{\rho_C}(Tr)$, we obtain $t_1.\langle h\rangle.t_2 \in Tr$ (view $\mathcal{H}$ permits no corrections), which means $t \in Tr$ ($t = t_1.\langle h\rangle.t_2$). Consequently, *SEP(ES)* holds. □

A.7 Representation Lemma for *PSP*

Lemma A.7.1. For the function ρ_E defined by $\rho_E(V, N, C) = E$, we have the following implications:

$$PSP(ES) \Rightarrow SD_{\mathcal{H}}(Tr) \quad (A.15)$$

$$PSP(ES) \Rightarrow SIA_{\mathcal{H}}^{\rho_E}(Tr) \quad (A.16)$$

$$R_{\mathcal{H}}(Tr) \wedge IA_{\mathcal{H}}^{\rho_E}(Tr) \Rightarrow PSP(ES) \quad (A.17)$$

◇

Proof. For proving $SD_{\mathcal{H}}(Tr)$ in (A.15), we define an auxiliary predicate *S-deletable*. Let $\tau \in Tr$ be a possible trace and $n \in \mathbb{N}$ be a natural number. *S-deletable*$_H(Tr, \tau, n)$ holds if deleting all occurrences of events in H from τ at once, with the exception of the n left-most occurrences (if they exist), yields, again, a possible trace in Tr. According to this construction, $SD_{\mathcal{H}}(Tr)$ holds if and only if $\forall \tau \in Tr.$ *S-deletable*$_H(Tr, \tau, |(\tau|_H)| - 1)$. Assume now that *PSP(ES)* holds. We prove that $\forall n \in \mathbb{N}. \forall \tau \in Tr.$ *S-deletable*$_H(Tr, \tau, n)$ holds by induction on n. In the base case holds $n = 0$. That $\forall \tau \in Tr.$ *S-deletable*$_H(Tr, \tau, 0)$ holds follows immediately from the first conjunct in Definition 4.2.25. In the step case, let $n \in \mathbb{N}$ be arbitrary with $n > 0$ and we assume that $\forall \tau \in Tr.$ *S-deletable*$_H(Tr, \tau, n^*)$ holds for all $n^* \in \mathbb{N}$ with $n^* < n$. We prove that $\forall \tau \in Tr.$ *S-deletable*$_H(Tr, \tau, n)$ holds. Let $\tau \in Tr$ be arbitrary. If the length of $\tau|_H$ is smaller than n, then the proposition follows immediately from the induction assumption. Thus, we assume that the length of $\tau|_H$ is at least n. We split τ into subsequences before and after the nth high-level event. Choose $\delta, \gamma \in E^*$ and $h \in H$ such that the length of $\delta|_H$ is $n-1$ and $\delta.\langle h\rangle.\gamma = \tau$ ($\gamma|_H = \langle\rangle$ need not hold). According to the induction assumption, $\delta.(\gamma|_L) \in Tr$ holds. That $\delta.\langle h\rangle.(\gamma|_L) \in Tr$ holds follows immediately from the second conjunct in Definition 4.2.25 (choose $\alpha = \gamma|_L$ and $\beta = \delta$). This concludes the proof of the step case, i.e. we have successfully shown that $\forall n \in \mathbb{N}. \forall \tau \in Tr.$ *S-deletable*$_H(Tr, \tau, n)$ holds. This implies that $SD_{\mathcal{H}}(Tr)$ holds.

Implication (A.16) follows immediately from the second conjunct in Definition 4.2.25.

For proving *PSP(ES)* in (A.17), assume that $R_{\mathcal{H}}(Tr)$ and $IA_{\mathcal{H}}^{\rho_E}(Tr)$ hold. The first and second conjunct in the definition of *PSP* (Definition 4.2.25) follow, respectively, from $R_{\mathcal{H}}(Tr)$ and $IA_{\mathcal{H}}^{\rho_E}(Tr)$. Thus, *PSP(ES)* holds. □

Appendix B

The Relationship between *MAKS* and the Framework of Selective Interleaving Functions

Among the previously proposed frameworks for information flow properties, McLean's framework of selective interleaving functions [McL94a, McL96] is closest to *MAKS* in the sense that it also allows for the investigation of classes of properties. In this appendix, we elaborate a rigorous relationship between the representation of information flow properties in McLean's framework and the representation in our framework.

We will proceed as follows: In Section B.1, we will show how the concepts used for representing information flow properties in McLean's framework can be defined in an event-based setting.[1] In Section B.2, we will show how the condition that *a set of traces is closed under a set of selective interleaving functions of a particular type* (i.e. the main condition for representing properties in McLean's framework) can be formulated in *MAKS*. The main outcome of our investigation will be that every property represented in the framework of selective interleaving functions can also be represented in *MAKS* and that there are information flow properties that can be represented in *MAKS* but not in the framework of selective interleaving functions (in its current form). In Section B.3, we point out the extensions of the framework of selective interleaving functions that are necessary for representing these information flow properties. The proofs of all theorems that we present are contained in Section B.4.

B.1 McLean's Framework of Selective Interleaving Functions

Selective interleaving functions are introduced in [McL94a, McL96] for a state-based system model. In an event-based system model, they can be defined as follows.

Definition B.1.1 (Selective interleaving function). Let E be a set of events, $\kappa : E \to \{0,1,2\}$ be a function, and E_0, E_1, E_2 be subsets of E defined by $E_0 = \{e \in E \mid \kappa(e) = 0\}$, $E_1 = \{e \in E \mid \kappa(e) = 1\}$, and $E_2 = \{e \in E \mid \kappa(e) = 2\}$.

A function $f : (E^* \times E^*) \to E^*$ is a *selective interleaving function* (abbreviated by *sif*) of type F_κ if and only if for all $t_1, t_2 \in E^*$ holds

$$f(t_1, t_2) = t \Rightarrow (t|_{E_1} = t_1|_{E_1} \wedge t|_{E_2} = t_2|_{E_2}) \ . \qquad \diamond$$

[1] Originally, this framework has been introduced in a state-based setting. Recasting the basic definitions of the framework in an event-based setting is necessary for formally relating it to our framework.

Hence, a *sif* f of type F_κ takes two traces as arguments and returns a trace that equals the first trace in its occurrences of events in E_1 and the second trace in its occurrences of events in E_2. If E_2 contains all events whose occurrences are visible to an observer and E_1 contains all events whose occurrences and nonoccurrences are confidential for him then closure under f means that every possible observation ($t|_{E_2}$) can occur in combination with every possible confidential behavior ($t|_{E_1}$). Intuitively, this requirement means: an observer cannot deduce from a given observation that some confidential behavior cannot have occurred. This requirement is closely related to Sutherland's nondeducibility [Sut86][2] and, hence, requiring closure under a *single* interleaving function does not result in very satisfactory security properties because this requirement shares the deficiencies of nondeducibility (cf. Section 4.2.1). In order to avoid the shortcomings of nondeducibility, McLean requires that the set of possible traces of a given system is closed under a *set of sifs* (rather than under a single *sif*).

Definition B.1.2 (Closure under set of *sifs*). Let E be a set of events, $\kappa : E \to \{0,1,2\}$ be a function, and $\mathcal{F}$ be a set of *sifs* of type F_κ. *ES* is *closed* under $\mathcal{F}$ if and only if

$$\forall f \in \mathcal{F}.\, \forall \tau_1, \tau_2 \in Tr.\, f(\tau_1, \tau_2) \in Tr\ . \qquad \diamond$$

In a state-based setting, the requirement that a set of possible traces is closed under a set of *sifs* of a particular type results in sensible security properties. However, in an event-based setting, this requirement alone is not satisfactory for expressing sensible security properties because it does not prevent that the interleavings of visible and confidential events can be narrowed down by an observer, a pitfall pointed out in [GN88]. In order to avoid this pitfall, we introduce an additional concept.

Definition B.1.3 (Covering). Let E be a set of events, $\kappa : E \to \{0,1,2\}$ be a function, and $\mathcal{F}$ be a set of *sifs* of type F_κ. Let $E_j = \{e \in E \mid \kappa(e) = j\}$ for $j \in \{0,1,2\}$. $\mathcal{F}$ *covers type* F_κ[3] if and only if

$$\forall t_1, t_2 \in E^*.\, \forall t \in \mathit{interleaving}(t_1|_{E_1}, t_2|_{E_2}).\, \exists f \in \mathcal{F}.\, f(t_1, t_2)|_{E_1 \cup E_2} = t\ . \qquad \diamond$$

To cover a type F_κ means for a set of *sifs* that for every interleaving of a possible E_1-sequence with a possible E_2-sequence there is a *sif* in the set that constructs this interleaving. Hence, if a set of possible traces is closed under a set of *sifs* that covers a type F_κ then an observer cannot narrow down the interleavings of E_1-sequences and E_2-sequences because every interleaving of these sequences is possible. Also note that if $\mathcal{F}$ covers F_κ then $\mathcal{F}$ must be nonempty because E^* is nonempty ($\langle\rangle \in E^*$ holds) and, for $t_1, t_2 \in E^*$, $\mathit{interleaving}(t_1|_{E_1}, t_2|_{E_2})$ is also nonempty ($(t_1|_{E_1}).(t_2|_{E_2}) \in \mathit{interleaving}(t_1|_{E_1}, t_2|_{E_2})$ holds). Moreover, for every type F_κ there is a covering set of *sifs* (e.g. the set of all *sifs* of type F_κ covers F_κ).

We are now ready to express the key notion for representing information flow properties in the framework of selective interleaving functions in an event-based setting, namely:

A set of traces is closed under a covering set of sifs of some type F_κ. (B.1)

Let us now illustrate how condition (B.1) can be used to express information flow properties.

Theorem B.1.4 (*SEP*). Let $ES = (E, I, O, Tr)$ be an event system. Define κ_{SEP} by:

$$\begin{aligned} \kappa_{SEP}(e) &= 1 \quad \text{if } e \text{ is a high-level event} \\ \kappa_{SEP}(e) &= 2 \quad \text{if } e \text{ is a low-level event} \end{aligned}$$

[2] If E_1 contains all high-level inputs and E_2 contains all low-level events (for a given 2-level security policy) then the two requirements are equivalent.

[3] For brevity, we also say that $\mathcal{F}$ is a *covering set of sifs of type* F_κ if $\mathcal{F}$ is a set of *sifs* that covers F_κ.

SEP(*ES*) holds if and only if there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{SEP}}$ such that *Tr* is closed under $\mathcal{F}$. ◇

Theorem B.1.5 (*IBGNI).** Let $ES = (E, I, O, Tr)$ be an event system. Define κ_{IBGNI^*} by:

$$\begin{aligned} \kappa_{IBGNI^*}(e) &= 0 \quad \text{if } e \text{ is a high-level internal or output event} \\ \kappa_{IBGNI^*}(e) &= 1 \quad \text{if } e \text{ is a high-level input event} \\ \kappa_{IBGNI^*}(e) &= 2 \quad \text{if } e \text{ is a low-level event} \end{aligned}$$

*IBGNI**(*ES*) holds if and only if there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{IBGNI^*}}$ such that *Tr* is closed under $\mathcal{F}$. ◇

The proofs of Theorems B.1.4 and B.1.5 will be presented in Section B.4.

For the representation of *NF* and *GNF*, condition (B.1) does not suffice. For representing these properties, it is necessary to introduce an additional notion, which can be used to constrain the values that *sifs* may return. An example for such a *range restriction* is that no events from E_0 must occur in the trace returned by a *sif*. This condition can be used for representing *NF*.

Theorem B.1.6 (*NF*). Let $ES = (E, I, O, Tr)$ be an event system. Define κ_{NF} by:

$$\begin{aligned} \kappa_{NF}(e) &= 0 \quad \text{if } e \text{ is a high-level event} \\ \kappa_{NF}(e) &= 2 \quad \text{if } e \text{ is a low-level event} \end{aligned}$$

NF(*ES*) holds if and only if there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{NF}}$ such that *Tr* is closed under $\mathcal{F}$ and $f(\tau_1, \tau_2)|_{E_0} = \langle\rangle$ holds for all $f \in \mathcal{F}$ and all $\tau_1, \tau_2 \in E^*$. ◇

Another example for a range restriction is that no events from $E_0 \cap I$ must occur in the trace returned by a *sif*. This condition can be used for representing *GNF*.

Theorem B.1.7 (*GNF*). Let $ES = (E, I, O, Tr)$ be an event system. Define κ_{GNF} by:

$$\begin{aligned} \kappa_{GNF}(e) &= 0 \quad \text{if } e \text{ is a high-level event} \\ \kappa_{GNF}(e) &= 2 \quad \text{if } e \text{ is a low-level event} \end{aligned}$$

GNF(*ES*) holds if and only if there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{GNF}}$ such that *Tr* is closed under $\mathcal{F}$ and $f(\tau_1, \tau_2)|_{E_0 \cap I} = \langle\rangle$ holds for all $f \in \mathcal{F}$ and all $\tau_1, \tau_2 \in E^*$. ◇

Theorem B.1.4–B.1.7 show how to represent separability, (interleaving-based) generalized noninterference, noninference, and generalized noninference in our reformulation of McLean's framework of selective interleaving functions for the event-based setting. These properties also have been investigated in [McL94a, McL96]. To the best of our knowledge, so far, no other information flow properties have been represented in the framework of selective interleaving functions. Interestingly, all of these properties can also be represented in *MAKS* (cf. Definition 4.2.8 and Theorems 4.2.20, 4.2.22, and 4.2.24).

Remark B.1.8. In [McL96], two alternative notions are suggested that both suffice to represent *NF* and *GNF*. The approach described by footnote 7 in that article corresponds to the approach that we have pursued, namely to restrict the range of *sifs*. The other approach (described in the body of that article) constrains the domains of *sifs* rather than their range. The effects are the same in both approaches [McL96]. ◇

Remark B.1.9. Another reformulation of McLean's framework of selective interleaving functions has been proposed by Zakinthinos [Zak96]. We found that this reformulation has serious shortcomings. For example, Zakinthinos requires that a set of possible traces is closed under a single *sif* of a particular type (rather than under a covering set of *sifs*). This means that deductions about the interleaving of occurrences of confidential events and occurrences of visible events are not ruled out. However, preventing such deductions is a critical issue in an event-based setting, as pointed out by Guttman and Nadel [GN88]. An even more serious problem results from the way in which *sifs* are defined in [Zak96]. E.g., rather than demanding that $f(t_1, t_2)|_{E_1} = t_1|_{E_1}$ holds, Zakinthinos require that $f(t_1, t_2)|_{E_1 \cap I} = t_1|_{E_1 \cap I}$ and $f(t_1, t_2)|_{E_1 \cap O} = t_1|_{E_1 \cap O}$ hold. Unfortunately, these two conditions do not ensure that input and output events in E_1 occur in the right order in $f(t_1, t_2)$. In particular, $f(t_1, t_2)|_{E_1} = t_1|_{E_1}$ need not hold (even if there are no internal events). As a result of this, for example, the representation of separability in Zakinthinos's reformulation of the framework of selective interleaving functions (cf. Section 6.2 in [Zak96]) is not equivalent to the original definition of separability.

Motivated by these shortcomings, we have developed the reformulation of the framework of selective interleaving functions for an event-based setting presented in this section. ◇

B.2 Expressing Closure under Set of *Sifs* in *MAKS*

The following theorem shows that a set of possible traces Tr is closed under some covering set of *sifs* of a particular type F_κ if and only if $R_\mathcal{V}(Tr)$ and $IA_\mathcal{V}^{\rho_C}(Tr)$ hold for some choice of $\mathcal{V}$. The choices of F_κ and $\mathcal{V}$ closely depend on each other. If one of F_κ or $\mathcal{V}$ is given then the other one can be calculated.

Theorem B.2.1. Let E be a set of events, $\mathcal{V} = (V, N, C)$ be a view in E, and ρ_C be a function from views in E to subsets of E with $\rho_C(\mathcal{V}) = C$. Moreover, let $\kappa : E \to \{0, 1, 2\}$ be a function. If $\forall v \in V.\, \kappa(v) = 2$, $\forall n \in N.\, \kappa(n) = 0$, and $\forall c \in C.\, \kappa(c) = 1$ then the following two propositions are equivalent:

1. $R_\mathcal{V}(Tr)$ and $IA_\mathcal{V}^{\rho_C}(Tr)$ hold.
2. Tr is closed under some covering set of *sifs* of type F_κ. ◇

Theorem B.2.1 implies that every information flow property representable in the framework of selective interleaving functions (by the requirement that the set of traces is closed under a covering set of *sifs*) can also be represented in *MAKS*. The somewhat involved proof of this theorem will be presented in Section B.4.

B.3 Lessons Learned

Theorem B.2.1 is the main technical result of this appendix. It shows that the requirement for a set of traces to be closed under a covering set is equivalent to an instance of the (schematic) statement $R_\mathcal{V}(Tr) \wedge IA_\mathcal{V}^{\rho_C}(Tr)$ where the particular instance (i.e. the choice of $\mathcal{V}$) depends on the type of the selective interleaving functions.

As simple consequences of this theorem (and our results in Section 4.2), we obtain that *SEP* and *IBGNI** can be represented in the framework of selective interleaving functions

(thereby proving Theorems B.1.4 and B.1.5). This is because *SEP* can be represented in *MAKS* by $R_{\mathcal{H}}(Tr) \wedge IA^{\rho_C}_{\mathcal{H}}(Tr)$ and *IBGNI** by $R_{\mathcal{HI}}(Tr) \wedge IA^{\rho_C}_{\mathcal{HI}}(Tr)$. That is, both properties can be represented by instances of the pattern $R_{\mathcal{V}}(Tr) \wedge IA^{\rho_C}_{\mathcal{V}}(Tr)$ (cf. the proofs of Theorems B.1.4 and B.1.5 in Section B.4 for the detailed argument).

Another consequence of Theorem B.2.1 is that *GNI*, *IBGNI*, *GNI**, *FC*, *FC**, *NDO*, *NDO**, *NF*, *GNF*, and *PSP* cannot be easily represented in the framework of selective interleaving functions. This is because their representations in *MAKS* do not comply with the pattern $R_{\mathcal{V}}(Tr) \wedge IA^{\rho_C}_{\mathcal{V}}(Tr)$. However, by introducing the notion of range restrictions it becomes possible to represent (at least) *NF* and *GNF* in this framework.[4]

Unfortunately, adding the notion of a range restriction does not suffice for representing *GNI*, *IBGNI*, *GNI**, *FC*, *FC**, *NDO*, *NDO**, or *PSP*. In order to represent any of these properties, it appears necessary to enrich the framework of selective interleaving functions with further concepts. Based on the representation of these properties in *MAKS* (from Section 4.2), we will now elaborate more closely what kind of concepts are missing. For each of *GNI*, *IBGNI*, *GNI**, *FC*, *FC**, *NDO*, *NDO**, and *PSP*, Table B.1 gives a representation of this property in *MAKS* and lists the concepts used in this representation that have no corresponding counterparts in the framework of selective interleaving functions.

property	modular representation	needed additional concepts
IBGNI(*ES*)	$R_{\mathcal{HI}}(Tr)$, $I_{\mathcal{HI}}(Tr)$	lack of admissibility assumption (I instead of IA)$^\times$
GNI(*ES*)	$BSD_{\mathcal{HI}}(Tr)$, $BSI_{\mathcal{HI}}(Tr)$	lack of admissibility assumption$^\times$, backwards strictness
GNI*(*ES*)	$BSD_{\mathcal{HI}}(Tr)$, $BSIA^{\rho_C}_{\mathcal{HI}}(Tr)$	backwards strictness
FC(*ES*)	$BSD_{\mathcal{HI}}(Tr)$, $BSI_{\mathcal{HI}}(Tr)$, $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$, $FCI^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$	lack of admissibility assumption$^\times$, backwards strictness, forward correctability
FC*(*ES*)	$BSD_{\mathcal{HI}}(Tr)$, $BSIA^{\rho_C}_{\mathcal{HI}}(Tr)$ $FCD^{\Gamma_{FC}}_{\mathcal{HI}}(Tr)$, $FCIA^{\rho_C,\Gamma_{FC}}_{\mathcal{HI}}(Tr)$	backwards strictness, forward correctability
NDO(*ES*), *NDO**(*ES*)	$R_{\mathcal{H}}(Tr)$, $IA^{\rho_{UI}}_{\mathcal{H}}(Tr)$	different choice of ρ (ρ_{UI} instead of ρ_C)
PSP(*ES*)	$R_{\mathcal{H}}(Tr)$, $IA^{\rho_E}_{\mathcal{H}}(Tr)$	different choice of ρ (ρ_E instead of ρ_C)

$^\times$ presence or lack of the admissibility assumption is irrelevant for systems total in high-level inputs

$\Gamma_{FC} = (I, \emptyset, I)$

Table B.1: Additional concepts that would be necessary in the framework of selective interleaving functions to represent certain information flow properties

For example, $I_{\mathcal{HI}}(Tr)$ occurs in the representation of *IBGNI* but this requirement cannot be expressed in the framework of selective interleaving functions. The closest requirement that can be expressed is $IA^{\rho_C}_{\mathcal{HI}}(Tr)$. Hence, in order to represent $I_{\mathcal{HI}}(Tr)$ one needs to introduce a notion that corresponds to dropping the assumption of ρ_C-admissibility in the definition

[4]Separability, (interleaving-based) generalized noninterference, noninference, and generalized noninference can be represented in the framework of selective interleaving functions. This has been demonstrated in [McL94a, McL96]. The novel aspect of our result is that it is concerned with our reformulation of this framework for an event-based system model.

of $IA^{\rho_C}_{\mathcal{HI}}(Tr)$. The gain of such an extension would be that *IBGNI* could be represented (also cf. first row in Table B.1). Moreover, the representation of GNI^* involves backwards-strict BSPs, i.e. *BSD* and $BSIA^{\rho_C}$, but the concepts provided by the framework of selective interleaving functions can only express the non-strict BSPs R and IA^{ρ_C}. Hence, in order to express this property one would have to find a way to limit corrections to causal ones in this framework. Furthermore, IA^{ρ_E} occurs in the representation of *PSP* but only IA^{ρ_C} can be expressed in the framework of selective interleaving functions. In order to represent *PSP* in this framework one would have to find a way to replace the ρ_C-admissibility assumption made by IA^{ρ_C} with a ρ_E-admissibility assumption (made by IA^{ρ_E}).

For each of *IBGNI*, *GNI*, GNI^*, *FC*, FC^*, *NDO*, NDO^* and *PSP*, Table B.1 points out what kinds of concepts are missing in the framework of selective interleaving functions in order to represent this information flow property. The entries of the table can also be regarded as hints of how the framework of selective interleaving functions should be extended. However, to elaborate such extensions in detail goes beyond the objectives of this appendix (i.e. to rigorously relate the framework of selective interleaving functions with *MAKS*). This remains an open task for future work.

B.4 Proofs of all Theorems in this Appendix

Proof (of Theorem B.1.4). $SEP(ES)$ is equivalent to $R_{\mathcal{H}}(Tr) \wedge IA^{\rho_C}_{\mathcal{H}}(Tr)$ (cf. Lemma A.6.1 and Theorems 3.5.3 and 3.5.12). According to Theorem B.2.1, this statement is equivalent to the requirement that Tr is closed under some covering set of *sifs* of type $F_{\kappa_{SEP}}$. □

Proof (of Theorem B.1.5). $IBGNI^*(ES)$ holds iff $D_{\mathcal{HI}}(Tr) \wedge IA^{\rho_C}_{\mathcal{HI}}(Tr)$ (cf. Definition 4.2.8). This statement is logically equivalent to the statement $R_{\mathcal{HI}}(Tr) \wedge IA^{\rho_C}_{\mathcal{HI}}(Tr)$. According to Theorem B.2.1, the latter statement is logically equivalent to the requirement that Tr is closed under some covering set of *sifs* of type $F_{\kappa_{IBGNI^*}}$. □

Proof (of Theorem B.1.6). Firstly, assume that there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{NF}}$ such that Tr is closed under $\mathcal{F}$ and $f(\tau_1, \tau_2)|_{E_0} = \langle\rangle$ holds for all $f \in \mathcal{F}$ and all $\tau_1, \tau_2 \in E^*$. From this, we will show that $NF(ES)$ holds.

Let $\tau \in Tr$ (if $Tr = \emptyset$ then the statement holds trivially) and $f \in \mathcal{F}$ ($\mathcal{F}$ is nonempty because it covers $F_{\kappa_{NF}}$) be arbitrary. Since f is a *sif* and $E_2 = L$, we have $f(\langle\rangle, \tau)|_L = \tau|_L$. Since $E_0 = H$, we have $f(\langle\rangle, \tau)|_H = \langle\rangle$ according to our assumptions about $\mathcal{F}$. From $f(\langle\rangle, \tau)|_L = \tau|_L$, $f(\langle\rangle, \tau)|_H = \langle\rangle$, and $E_1 = \emptyset$, we obtain $f(\langle\rangle, \tau) = \tau|_L$. Since Tr is closed under $\mathcal{F}$, $\tau|_L$ holds. Since τ was chosen arbitrarily, $NF(ES)$ holds.

Secondly, assume that $NF(ES)$ holds. From this, we will show that there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{NF}}$ such that Tr is closed under $\mathcal{F}$ and $f(\tau_1, \tau_2)|_{E_0} = \langle\rangle$ holds for all $f \in \mathcal{F}$ and all $\tau_1, \tau_2 \in E^*$. We define $\mathcal{F} = \{f \text{ is a } sif \text{ of type } F_{\kappa_{NF}} \mid \forall \tau_1, \tau_2 \in E^*.\, f(\tau_1, \tau_2)|_{E_0} = \langle\rangle\}$.

We have to show that Tr is closed under $\mathcal{F}$. Let $f \in \mathcal{F}$ and $\tau_1, \tau_2 \in Tr$ be arbitrary. Since $f(\tau_1, \tau_2) = \tau_2|_L$ (follows from $E_1 = \emptyset$ and our definition of $\mathcal{F}$) and $\tau_2|_L \in Tr$ (follows from $NF(ES)$), we have $f(\tau_1, \tau_2) \in Tr$. Since f, τ_1, and τ_2 are arbitrary, Tr is closed under $\mathcal{F}$.

It remains to show that $\mathcal{F}$ covers $F_{\kappa_{NF}}$. Let $t_1, t_2 \in E^*$ and $t \in interleaving(t_1|_{E_1}, t_2|_{E_2})$ be arbitrary. Since $E_1 = \emptyset$, we have $t = t_2|_{E_2}$. Define f by: $\forall t'_1, t'_2 \in E^*.\, f(t'_1, t'_2) = t'_2|_{E_2}$. Obviously, $f \in \mathcal{F}$ and $f(t_1, t_2) = t_2|_{E_2} = t$ hold. Hence, $\mathcal{F}$ covers $F_{\kappa_{NF}}$. □

Proof (of Theorem B.1.7). Firstly, assume that there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{GNF}}$

such that Tr is closed under $\mathcal{F}$ and $f(\tau_1, \tau_2)|_{E_0 \cap I} = \langle\rangle$ holds for all $f \in \mathcal{F}$ and all $\tau_1, \tau_2 \in E^*$. From this, we will show that $GNF(ES)$ holds.

Let $\tau \in Tr$ and $f \in \mathcal{F}$ be arbitrary. Since f is a *sif* and $E_2 = L$, we have $f(\langle\rangle, \tau)|_L = \tau|_L$. Since $E_0 = H$, $f(\langle\rangle, \tau)|_{H \cap I} = \langle\rangle$ follows from our assumptions about $\mathcal{F}$. Since Tr is closed under $\mathcal{F}$, $f(\langle\rangle, \tau) \in Tr$ holds. For $\tau' = f(\langle\rangle, \tau)$, we have $\tau' \in Tr$, $\tau'|_L = \tau|_L$, and $\tau'|_{H \cap I} = \langle\rangle$. Hence, $GNF(ES)$ holds.

Secondly, assume that $GNF(ES)$ holds. From this, we will show that there is a covering set $\mathcal{F}$ of *sifs* of type $F_{\kappa_{GNF}}$ such that Tr is closed under $\mathcal{F}$ and $f(\tau_1, \tau_2)|_{E_0 \cap I} = \langle\rangle$ holds for all $f \in \mathcal{F}$ and all $\tau_1, \tau_2 \in E^*$. We define $\mathcal{F} = \{f\}$ where f is defined by

$$f(t_1, t_2) = \begin{cases} t_2|_{E_2} & \text{if } t_2 \notin Tr \\ t_2' & \text{if } t_2 \in Tr, \text{ where } t_2' \in E^* \text{ is some trace with } t_2' \in Tr, \\ & \quad t_2'|_L = t_2|_L, \text{ and } t_2'|_{H \cap I} = \langle\rangle \\ & \quad (t_2' \text{ with these properties exists because } GNF(ES) \text{ holds}) \end{cases}$$

That Tr is closed under $\mathcal{F}$ follows immediately from the definition of f. It remains to show that $\mathcal{F}$ covers $F_{\kappa_{GNF}}$. Let $t_1, t_2 \in E^*$ and $t \in interleaving(t_1|_{E_1}, t_2|_{E_2})$ be arbitrary. Since $E_1 = \emptyset$, we have $t = t_2|_{E_2}$. Since $f(t_1, t_2)|_{E_1 \cup E_2} = t_2|_{E_2} = t$ and t_1, t_2, t were chosen arbitrarily, we conclude that $\mathcal{F}$ covers $F_{\kappa_{GNF}}$. □

The following lemma states that if $R_{\mathcal{V}}(Tr)$ and $IA_{\mathcal{V}}^{\rho_C}(Tr)$ hold then there is a set $\mathcal{F}^n$ of functions that has three properties:

1. Each function in $\mathcal{F}^n$ interleaves a trace that has at most length n with another trace that may have arbitrary (finite) length. Intuitively, this can be understood as: Every function in $\mathcal{F}^n$ approximates a *sif* (for the restricted domain $(\bigcup_{n' \leq n} E^{n'}) \times E^*$).

2. The set of possible traces is closed under all functions in $\mathcal{F}^n$.

3. For each possibility to construct interleavings of E_1-sequences (of at most length n) with E_2-sequences (of arbitrary length) there is a function f in $\mathcal{F}^n$ that performs this construction. More precisely, for every function ι that takes a trace t_1' (of at most length n) and a trace t_2' (of arbitrary length) and that returns an interleaving of the E_1-sequence in t_1' with the E_2-sequence in t_2' there is a function f in $\mathcal{F}^n$ such that $f(t_1', t_2')|_{E_1 \cup E_2} = \iota(t_1', t_2')$ holds. This requirement can be understood as: The set $\mathcal{F}^n$ approximates the coverage requirement (for the restricted domain $(\bigcup_{n' \leq n} E^{n'}) \times E^*$).

This lemma will be helpful in the proof of Theorem B.2.1.

Lemma B.4.1. Let E be a set of events, $\mathcal{V} = (V, N, C)$ be a view in E, and ρ_C be a function from views in E to subsets of E with $\rho_C(\mathcal{V}) = C$. Moreover, let $\kappa : E \to \{0, 1, 2\}$ be defined by $\forall v \in V.\, \kappa(v) = 2$, $\forall n \in N.\, \kappa(n) = 0$, and $\forall c \in C.\, \kappa(c) = 1$. For $j \in \{0, 1, 2\}$ the set E_j shall be defined by $E_j = \{e \in E \mid \kappa(e) = j\}$.

If $R_{\mathcal{V}}(Tr) \wedge IA_{\mathcal{V}}^{\rho_C}(Tr)$ holds then, for every $n \in \mathbb{N}$, there is a set $\mathcal{F}^n \subseteq ((\bigcup_{n' \leq n} E^{n'}) \times E^*) \to E^*$ of functions for which the following three propositions hold:

1. Every $f \in \mathcal{F}^n$ approximates a *sif*, i.e.

$$\forall f \in \mathcal{F}^n.\, \forall t_1 \in \bigcup_{n' \leq n} E^{n'}.\, \forall t_2 \in E^*. \quad (B.2)$$
$$[(f(t_1, t_2))|_{E_1} = t_1|_{E_1} \wedge (f(t_1, t_2))|_{E_2} = t_2|_{E_2}]$$

2. Tr is closed under every $f \in \mathcal{F}^n$, i.e.

$$\begin{aligned} &\forall f \in \mathcal{F}^n.\, \forall t_1 \in \textstyle\bigcup_{n' \leq n} E^{n'}.\, \forall t_2 \in E^*. \\ &\quad [(t_1 \in Tr \wedge t_2 \in Tr) \Rightarrow f(t_1, t_2) \in Tr] \end{aligned} \tag{B.3}$$

3. $\mathcal{F}^n$ approximates the coverage requirement, i.e.

$$\begin{aligned} &\forall \iota \in ((\textstyle\bigcup_{n' \leq n} E^{n'}) \times E^*) \to (E_1 \cup E_2)^*. \\ &\quad [(\forall t_1' \in \textstyle\bigcup_{n' \leq n} E^{n'}.\, \forall t_2' \in E^*.\, \iota(t_1', t_2') \in \mathit{interleaving}(t_1'|_{E_1}, t_2'|_{E_2})) \\ &\qquad \Rightarrow \exists f \in \mathcal{F}^n.\, \forall t_1 \in \textstyle\bigcup_{n' \leq n} E^{n'}.\, \forall t_2 \in E^*.\, f(t_1, t_2)|_{E_1 \cup E_2} = \iota(t_1, t_2)] \end{aligned} \tag{B.4}$$

◇

Proof. The proof proceeds by induction on n.

Base case: $n = 0$ holds. Since $n = 0$, we have $\bigcup_{n' \leq n} E^{n'} = \{\langle\rangle\}$. We construct $\mathcal{F}^0$ by $\mathcal{F}^0 = \{f^0\}$ where f^0 is defined as follows for $t_1 \in \bigcup_{n' \leq n} E^{n'}$ and $t_2 \in E^*$:

$$f^0(t_1, t_2) = f^0(\langle\rangle, t_2) = \begin{cases} t_2|_{E_2} & \text{if } t_2 \notin Tr \\ t_2' & \text{if } t_2 \in Tr, \text{ where } t_2' \in E^* \text{ is some trace} \\ & \text{with } t_2' \in Tr,\ t_2'|_{E_2} = t_2|_{E_2}, \text{ and} \\ & t_2'|_{E_1} = \langle\rangle \ (t_2' \text{ with these properties exists} \\ & \text{because } R_{\mathcal{V}}(Tr) \text{ holds}) \end{cases}$$

For $n = 0$, (B.2) and (B.3) hold according to the construction of f^0.

The only function $\iota^0 \in ((\bigcup_{n' \leq 0} E^{n'}) \times E^*) \to (E_1 \cup E_2)^*$ with $\forall t \in \bigcup_{n' \leq n} E^{n'}.\, \forall t' \in E^*.\, \iota^0(t, t') \in \mathit{interleaving}(t|_{E_1}, t'|_{E_2})$ can be defined by $\iota^0(t_1', t_2') = t_2|_{E_2}$. According to our construction of f^0,

$$f^0(t_1, t_2)|_{E_1 \cup E_2} = f^0(\langle\rangle, t_2)|_{E_1 \cup E_2} = t_2|_{E_2} = \iota^0(t_1, t_2)$$

holds for all $t_1 \in E^0$ and $t_2 \in E^*$. Hence, (B.4) holds for $n = 0$.

Step case: $n > 0$ holds. We construct $\mathcal{F}^n$ as follows:

$$\mathcal{F}^n = \left\{ f_\iota^n \;\middle|\; \begin{array}{l} f_\iota^n : ((\bigcup_{n' \leq n} E^{n'}) \times E^*) \to E^* \\ \wedge\ \iota : ((\bigcup_{n' \leq n} E^{n'}) \times E^*) \to (E_1 \cup E_2)^* \\ \wedge\ (\forall t_1', t_2' \in E^*.\, \iota(t_1', t_2') \in \mathit{interleaving}(t_1'|_{E_1}, t_2'|_{E_2})) \\ \wedge\ \mathrm{Def}_{f_\iota^n} \end{array} \right\}$$

where $\mathrm{Def}_{f_\iota^n}$ defines f_ι^n as follows for $t_1 \in \bigcup_{n' \leq n} E^{n'}$ and $t_2 \in E^*$:

$$\mathrm{Def}_{f_\iota^n} \equiv f_\iota^n(t_1, t_2) = \begin{cases} \iota(t_1, t_2) & \text{if } t_1 \notin Tr \text{ or } t_2 \notin Tr \\ t_{t_1,t_2} & \text{if } t_1, t_2 \in Tr, \text{ where } t_{t_1,t_2} \in E^* \text{ is some trace} \\ & \text{with } t_{t_1,t_2} \in Tr \text{ and } t_{t_1,t_2}|_{E_1 \cup E_2} = \iota(t_1, t_2) \end{cases}$$

The validity of (B.2), (B.3), and (B.4) follows immediately from this construction. It only remains to prove that $f_\iota^n(t_1, t_2)$ is well defined, i.e. that there always is a trace $t_{t_1,t_2} \in E^*$

such that $t_{t_1,t_2} \in Tr$ and $t_{t_1,t_2}|_{E_1 \cup E_2} = \iota(t_1,t_2)$ hold. For proving the existence of t_{t_1,t_2}, we make a case distinction on t_1: either (1) $t_1 \in \bigcup_{n'<n} E^{n'}$ or (2) $t_1 \in E^n$ holds.

Case 1: $t_1 \in \bigcup_{n'<n} E^{n'}$ holds. There is a function $\iota' : ((\bigcup_{n'<n} E^{n'}) \times E^*) \to (E_1 \cup E_2)^*$ that equals ι on the restricted domain $(\bigcup_{n'<n} E^{n'}) \times E^*$, i.e. $\iota'(t_3,t_4) = \iota(t_3,t_4)$ holds for all $t_3 \in \bigcup_{n'<n} E^{n'}$ and all $t_4 \in E^*$. According to the induction assumption, there is a function $f^{n-1}_{\iota'} \in \mathcal{F}^{n-1}$ such that $f^{n-1}_{\iota'}(t_3,t_4)|_{E_1 \cup E_2} = \iota'(t_3,t_4)$ holds for all $t_3 \in \bigcup_{n'<n} E^{n'}$ and all $t_4 \in E^*$ and such that if $t_3, t_4 \in Tr$ then $f^{n-1}_{\iota'}(t_3,t_4) \in Tr$ holds. We choose $t_{t_1,t_2} = f^{n-1}_{\iota'}(t_1,t_2)$. Consequently, $t_{t_1,t_2} \in Tr$ holds. Moreover, $t_{t_1,t_2}|_{E_1 \cup E_2} = \iota(t_1,t_2)$ holds because $t_{t_1,t_2}|_{E_1 \cup E_2} = \iota'(t_1,t_2)$ and ι' equals ι for the restricted domain $(\bigcup_{n'<n} E^{n'}) \times E^*$.

Case 2: $t_1 \in E^n$ holds. Let $t'_1 \in E^{n-1}$ and $e \in E$ be defined by $t_1 = t'_1.\langle e \rangle$. We make another case distinction on e: (2a) $e \notin E_1$ and (2b) $e \in E_1$.

Case 2a: $e \notin E_1$ holds. There is a function $\iota' : ((\bigcup_{n'<n} E^{n'}) \times E^*) \to (E_1 \cup E_2)^*$ such that $\iota'(t'_1,t_2) = \iota(t_1,t_2)$ holds and $\iota'(t_3,t_4) \in \mathit{interleaving}(t_3|_{E_1}, t_4|_{E_2})$ holds for all $t_3 \in \bigcup_{n'<n} E^{n'}$ and all $t_4 \in E^*$. According to the induction assumption, there is a function $f^{n-1}_{\iota'} \in \mathcal{F}^{n-1}$ such that $f^{n-1}_{\iota'}(t_3,t_4)|_{E_1 \cup E_2} = \iota'(t_3,t_4)$ holds for all $t_3 \in \bigcup_{n'<n} E^{n'}$ and all $t_4 \in E^*$ and such that if $t_3, t_4 \in Tr$ then $f^{n-1}_{\iota'}(t_3,t_4) \in Tr$ holds. We choose $t_{t_1,t_2} = f^{n-1}_{\iota'}(t'_1,t_2)$ and obtain $t_{t_1,t_2} \in Tr$ and $t_{t_1,t_2}|_{E_1 \cup E_2} = \iota'(t'_1,t_2) = \iota(t_1,t_2)$.

Case 2b: $e \in E_1$ holds. Let $\alpha, \beta \in E^*$ be defined by $\iota(t'_1.\langle e \rangle, t_2) = \beta.\langle e \rangle.\alpha$ and $\alpha|_{E_1} = \langle\rangle$. There is a function $\iota' : ((\bigcup_{n'<n} E^{n'}) \times E^*) \to (E_1 \cup E_2)^*$ such that $\iota'(t'_1,t_2) = \beta.\alpha$ holds and $\iota(t_3,t_4) \in \mathit{interleaving}(t_3|_{E_1}, t_4|_{E_2})$ holds for all $t_3 \in \bigcup_{n'<n} E^{n'}$ and all $t_4 \in E^*$. According to the induction assumption, there is a function $f^{n-1}_{\iota'} \in \mathcal{F}^{n-1}$ such that $f^{n-1}_{\iota'}(t_3,t_4)|_{E_1 \cup E_2} = \iota'(t_3,t_4)$ holds for all $t_3 \in \bigcup_{n'<n} E^{n'}$ and all $t_4 \in E^*$ and such that if $t_3, t_4 \in Tr$ then $f^{n-1}_{\iota'}(t_3,t_4) \in Tr$ holds. Since $f^{n-1}_{\iota'}(t'_1,t_2)|_{E_1 \cup E_2} = \iota'(t'_1,t_2) = \beta.\alpha$, there are traces $\alpha', \beta' \in E^*$ with $\beta'.\alpha' = f^{n-1}_{\iota'}(t'_1,t_2)$, $\beta'|_{E_1 \cup E_2} = \beta|_{E_1 \cup E_2}$, $\alpha'|_{E_2} = \alpha|_{E_2}$, and $\alpha'|_{E_1} = \langle\rangle$. Since $t'_1|_{E_1} = \beta'|_{E_1}$ and $t'_1.\langle e \rangle \in Tr$ hold, we obtain that $Adm^{\rho C}_{\mathcal{V}}(Tr, \beta', e)$ holds. From $IA^{\rho C}_{\mathcal{V}}(Tr)$ we conclude that there are $\alpha'', \beta'' \in E^*$ with $\beta''.\langle e \rangle.\alpha'' \in Tr$, $\beta''|_{E_1 \cup E_2} = \beta'|_{E_1 \cup E_2}$, $\alpha''|_{E_2} = \alpha'|_{E_2}$, and $\alpha''|_{E_1} = \langle\rangle$. We choose $t_{t_1,t_2} = \beta''.\langle e \rangle.\alpha''$ and obtain $t_{t_1,t_2} \in Tr$ and $t_{t_1,t_2}|_{E_1 \cup E_2} = (\beta''.\langle e \rangle.\alpha'')|_{E_1 \cup E_2} = (\beta'.\langle e \rangle.\alpha')|_{E_1 \cup E_2} = (\beta.\langle e \rangle.\alpha)|_{E_1 \cup E_2} = \iota(t_1,t_2)$. □

Note in the proof of Lemma B.4.1 that $R_{\mathcal{V}}(Tr)$ is applied only in the base case (for defining f^0) and that $IA^{\rho C}_{\mathcal{V}}(Tr)$ is applied only in Case 2b of the step case (for proving the existence of a particular trace t_{t_1,t_2} if $t_1 \in E^n$, $t_1 = t'_1.\langle e \rangle$, and $e \in E_1$).

We are now ready to prove the main theorem of this appendix.

Proof (of Theorem B.2.1). Firstly, assume that $R_{\mathcal{V}}(Tr)$ and $IA^{\rho C}_{\mathcal{V}}(Tr)$ hold. We have to show that Tr is closed under some covering set $\mathcal{F}$ of *sifs* of type F_κ.

We choose $\mathcal{F} = \bigcup_{n \in \mathbb{N}} \mathcal{F}^n$ where $\mathcal{F}^n$ is defined like in the proof of Lemma B.4.1. Hence, Tr is closed under $\mathcal{F}$. It remains to prove that $\mathcal{F}$ covers F_κ. Let $t_1, t_2 \in E^*$ and $t \in \mathit{interleaving}(t_1|_{E_1}, t_2|_{E_2})$ be arbitrary. There is a number $n \in \mathbb{N}$ with $t_1 \in E^n$ (n is the length of t_1). Choose a function $\iota \in ((\bigcup_{n' \leq n} E^{n'}) \times E^*) \to (E_1 \cup E_2)^*$ with $\forall t'_1 \in \bigcup_{n' \leq n} E^{n'}. \forall t'_2 \in E^*. \iota(t'_1,t'_2) \in \mathit{interleaving}(t'_1|_{E_1}, t'_2|_{E_2})$ and $\iota(t_1,t_2) = t$. According to Lemma B.4.1 there is a function $f \in \mathcal{F}$ with $f(t_1,t_2)|_{E_1 \cup E_2} = \iota(t_1,t_2) = t$. Since t_1, t_2, t were chosen arbitrarily and $f \in \mathcal{F}$ ($\mathcal{F}^n \subseteq \mathcal{F}$ holds), we conclude that $\mathcal{F}$ covers F_κ.

Secondly, assume that Tr is closed under some covering set $\mathcal{F}$ of *sifs* of type F_κ. We have to show that $R_{\mathcal{V}}(Tr)$ and $IA^{\rho C}_{\mathcal{V}}(Tr)$ hold.

For proving that $R_{\mathcal{V}}(Tr)$ holds, let $\tau \in Tr$ be arbitrary (if $Tr = \emptyset$ then the proposition holds trivially). We choose $\tau' = f(\langle\rangle, \tau)$ where $f \in \mathcal{F}$ is arbitrary. $\tau' \in Tr$ holds because

Tr is closed under $\mathcal{F}$ and $\langle\rangle, \tau \in Tr$ holds ($\langle\rangle \in Tr$ because Tr is closed under prefixes). $\tau'|_C = \tau'|_{E_1} = f(\langle\rangle, \tau)|_{E_1} = \langle\rangle$ and $\tau'|_V = \tau'|_{E_2} = f(\langle\rangle, \tau)|_{E_2} = \tau|_{E_2} = \tau|_V$ because f is a *sif* of type F_κ. Thus, $R_{\mathcal{V}}(Tr)$ holds.

For proving that $IA_{\mathcal{V}}^{\rho_C}(Tr)$ holds, let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary. Assume $\beta.\alpha \in Tr$, $\alpha|_C = \langle\rangle$, and $Adm_{\mathcal{V}}^{\rho_C}(Tr, \beta, c)$. Hence $\alpha|_{E_1} = \langle\rangle$ and $c \in E_1$. Since $Adm_{\mathcal{V}}^{\rho_C}(Tr, \beta, c)$ there is a trace $\gamma \in E^*$ such that $\gamma.\langle c\rangle \in Tr$ and $\gamma|_{E_1} = \beta|_{E_1}$ hold. Consequently, $(\beta.\langle c\rangle.\alpha)|_{E_1 \cup E_2} \in$ *interleaving*$((\gamma.\langle c\rangle)|_{E_1}, (\beta.\alpha)|_{E_2})$ holds. Since $\mathcal{F}$ covers F_κ, there is a *sif* $f \in \mathcal{F}$ such that $(f(\gamma.\langle c\rangle, \beta.\alpha))|_{E_1 \cup E_2} = (\beta.\langle c\rangle.\alpha)|_{E_1 \cup E_2}$. Let $\beta', \alpha' \in E^*$ be the subsequences of $f(\gamma.\langle c\rangle, \beta.\alpha)$, respectively, before and after the last occurrence of c, i.e. $f(\gamma.\langle c\rangle, \beta.\alpha) = \beta'.\langle c\rangle.\alpha'$ and $\alpha'|_{E_1} = \langle\rangle$. Since Tr is closed under $\mathcal{F}$, we obtain $\beta'.\langle c\rangle.\alpha' \in Tr$. From $\beta'.\langle c\rangle.\alpha' \in Tr$, $\alpha'|_C = \langle\rangle$, and $(\beta'.\langle c\rangle.\alpha')|_{V \cup C} = (\beta.\langle c\rangle.\alpha)|_{V \cup C}$ we conclude that $IA_{\mathcal{V}}^{\rho_C}(Tr)$ holds. □

Appendix C

Verifying Information Flow Properties by Simulation

Simulation techniques are similar in spirit to unwinding in the sense that they reduce the verification of a global property to the verification of local conditions. However, an essential difference between these two techniques is that they have been developed for different properties. While the unwinding technique has been developed to simplify the verification of information flow properties, simulation techniques have been developed to simplify the verification that some system refines another.

However, we discovered a possibility to apply simulation techniques also for verifying information flow properties. The key observation is that BSPs can be expressed by refinement statements. After reformulating BSPs by refinement statements, simulation techniques become immediately applicable for their verification. Thereby, we obtain verification techniques for information flow properties that provide alternatives to the unwinding technique. The resulting verification techniques are *sound*. Moreover, some of them, namely forward-backward simulation and backward-forward simulation, are also complete.

Overview. In Section C.1, we recall four well known simulation techniques from the literature. In Section C.2, we show how backwards-strict BSPs and forward-correctable BSPs can be expressed in terms of refinement statements. In Section C.3, we illustrate by a concrete example how the simulation techniques can be employed during the verification of information flow properties. We conclude in Section C.4 with a brief summary of our main results.

Conventions. $ES^c = (E, I, O, Tr^c)$ and $ES^a = (E, I, O, Tr^a)$ denote event systems. $SES^a = (S^a, s_0^a, E, I, O, T^a)$, $SES^c = (S^c, s_0^c, E, I, O, T^c)$, and $SES = (S, s_0, E, I, O, T)$ denote state-event systems. Let $\mathcal{V} = (V, N, C)$ denote a view in E, let ρ denote a function from views in E to subsets of E, and let Γ denote a triple $(\nabla, \Delta, \Upsilon)$ of subsets of E. Moreover, let α and β denote finite traces, i.e. $\alpha, \beta \in E^*$, and let c denote a confidential event, i.e. $c \in C$.

C.1 Preliminaries on Simulation Techniques

Various simulation techniques have been proposed for proving that one system specification is a trace refinement of another one [LV95]. We refer to the particular notion of trace refinement considered in this setting by *observational trace refinement*.

Definition C.1.1 (Trace refinement). The event system ES^c is a *trace refinement* of ES^a (denoted by $ES^c \leq_T ES^a$) iff $Tr^c \subseteq Tr^a$ holds. ◇

Definition C.1.2 (Observational trace refinement). The event system ES^c is an *observational trace refinement* of ES^a (denoted by $ES^c \leq_{OT} ES^a$) iff $Tr^c|_{I \cup O} \subseteq Tr^a|_{I \cup O}$ holds where $Tr^c|_{I \cup O} = \{\tau|_{I \cup O} \mid \tau \in Tr^c\}$ and $Tr^a|_{I \cup O} = \{\tau|_{I \cup O} \mid \tau \in Tr^a\}$ are the observable traces of, respectively, ES^c and ES^a.

The state-event system SES^c is an observational trace refinement of SES^a (denoted by $SES^c \leq_{OT} SES^a$) iff $ES_{SES^c} \leq_{OT} ES_{SES^a}$ holds. ◇

If $ES^c \leq_{OT} ES^a$ holds then all observations that are possible at the interface of ES^c must also be possible at the interface of ES^a.

Various simulation techniques have been developed to prove that one system is an observational trace refinement of another system. These simulation techniques have in common that they reduce the verification of (global) refinement statements that involve the sets of all possible traces of both systems to more local conditions. However, they differ in which local verification conditions are used.

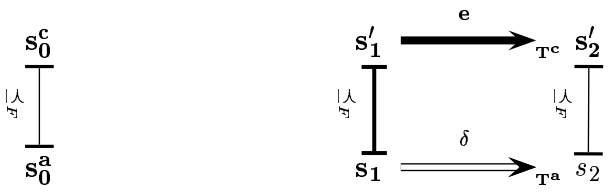
Below, we briefly recall four well known simulation techniques from the literature as well as corresponding soundness and completeness results. Our presentation is based on the overview article [LV95] with only a few minor changes in technical details.

Forward Simulation. The following definition is derived from Section 3.2 in [LV95].

Definition C.1.3 (Forward simulation). A relation $\preceq_F \subseteq S^c \times S^a$ is a *forward simulation* from SES^c to SES^a (denoted by $SES^c \preceq_F SES^a$) if it satisfies the following two conditions:

1. $s_0^c \preceq_F s_0^a$
2. $\forall s_1 \in S^a. \forall s_1', s_2' \in S^c. \forall e \in E.$
$$\left(\begin{array}{l} (s_1' \xrightarrow{e}_{T^c} s_2' \wedge s_1' \preceq_F s_1) \\ \Rightarrow \exists s_2 \in S^a. \exists \delta \in E^*. (\delta|_{I \cup O} = \langle e \rangle|_{I \cup O} \wedge s_1 \overset{\delta}{\Longrightarrow}_{T^a} s_2 \wedge s_2' \preceq_F s_2) \end{array} \right)$$ ◇

The conditions in Definition C.1.3 are also viewed by the two diagrams in Figure C.1 where we use the same conventions like in Chapter 5.



where $e \in E$, $\delta \in E^*$, and $\delta|_{I \cup O} = \langle e \rangle|_{I \cup O}$

Figure C.1: Verification conditions for forward simulation

Theorem C.1.4 (Soundness of forward simulation). If $\preceq_F \subseteq S^c \times S^a$ is a forward simulation from SES^c to SES^a then $SES^c \leq_{OT} SES^a$ holds. ◇

Proof. Follows from Theorem 3.10 in [LV95]. □

Forward simulation is a sound proof technique but it is, in general, not complete. For a conditional completeness result, we refer to Theorem 3.11 in [LV95].

Backward Simulation. The following definition of backward simulation is derived from Section 3.3 in [LV95]. Because of our assumption on state-event systems, i.e. that there is only one initial state and that T is functional in its first two arguments, the soundness result that we present is a specialization of the one in that article.

Definition C.1.5 (Backward simulation). A relation $\preceq_B \subseteq S^c \times S^a$ is a *backward simulation* from SES^c to SES^a (denoted by $SES^c \preceq_B SES^a$) if it is *total*[1] and if

1. $\forall s' \in S^a.\, (s_0^c \preceq_B s' \Rightarrow s' = s_0^a)$
2. $\forall s_2 \in S^a.\, \forall s_1', s_2' \in S^c.\, \forall e \in E.$
$$\left(\begin{array}{l} (s_1' \xrightarrow{e}_{T^c} s_2' \wedge s_2' \preceq_B s_2) \\ \Rightarrow \exists s_1 \in S^a.\, \exists \delta \in E^*.\, (\delta|_{I \cup O} = \langle e \rangle|_{I \cup O} \wedge s_1 \overset{\delta}{\Longrightarrow}_{T^a} s_2 \wedge s_1' \preceq_B s_1) \end{array} \right) \qquad \diamond$$

where $e \in E$, $\delta \in E^*$, and $\delta|_V = \langle e \rangle|_V$

Figure C.2: Verification conditions for backward simulation

Unlike forward simulation, backward simulation requires that the simulation relation is *total* in S^c. The two conditions of Definition C.1.5 are also shown in Figure C.2. Note that the diagram on the right hand side has the same shape like the one in Figure C.1.3. However, it differs in which parts are fat or thin, e.g., the quantification of s_1 and s_2 differs.

Theorem C.1.6 (Soundness of backward simulation). If $\preceq_B \subseteq S^c \times S^a$ is a backward simulation from SES^c to SES^a then $SES^c \leq_{OT} SES^a$ holds. $\diamond$

Proof. Follows from Theorem 3.17 in [LV95]. $\square$

Backward simulation is, in general, not complete. For a conditional completeness result, we refer to Theorem 3.18 in [LV95].

Remark C.1.7. Although forward simulation and backward simulation are, in general, not complete it is possible to obtain a complete verification technique by combining them. Theorem 3.22 in [LV95] states that if there are specifications A and B such that $A \leq_{OT} B$ then there is a specification C, a forward simulation $\preceq_F$, and a backward simulation $\preceq_B$ such that $A \preceq_F C$ and $C \preceq_B B$. Similar results can be found in [HHS87, He89, Jon89, Jon91, Jos88, KS93]. $\diamond$

Hybrid Simulations. Simulation techniques that are sound as well as complete, without having to construct an intermediate system specification (cf. Remark C.1.7), have been constructed by merging forward simulation with backward simulation. The following definitions are derived from Section 4 in [LV95].

[1] A relation $R \subseteq X \times Y$ is *total* over X if $domain(R) = X$ where $domain(R) = \{x \in X \mid \exists y \in Y.\, (x, y) \in R\}$.

One technical difference of these hybrid simulation techniques to forward simulation or backward simulation is that the simulation relation relates a state of the concrete system with a *set of states* of the abstract system (rather than with a single state of the abstract system).

Definition C.1.8 (Forward-backward simulation). A relation $\preceq_{FB} \subseteq S^c \times \mathcal{N}(S^a)^2$ is a *forward-backward simulation* from SES^c to SES^a (denoted by $SES^c \preceq_{FB} SES^a$) if it satisfies the following two conditions.

1. $s_0^c \preceq_{FB} \{s_0^a\}$
2. $\forall S_1 \in \mathcal{N}(S^a). \forall s_1', s_2' \in S^c. \forall e \in E.$
$$\left((s_1' \xrightarrow{e}_{T^c} s_2' \wedge s_1' \preceq_{FB} S_1) \Rightarrow \exists S_2 \in \mathcal{N}(S^a). \forall s_2 \in S_2. \exists \gamma \in E^*. \exists s_1 \in S_1. \right.$$
$$\left. (s_2' \preceq_{FB} S_2 \wedge \gamma|_{I \cup O} = \langle e \rangle|_{I \cup O} \wedge s_1 \overset{\gamma}{\Longrightarrow}_{T^a} s_2) \right) \quad \diamond$$

Definition C.1.9 (Backward-forward simulation). A relation $\preceq_{BF} \subseteq S^c \times \mathcal{P}(S^a)^3$ is a *backward-forward simulation* from SES^c to SES^a (denoted by $SES^c \preceq_{BF} SES^a$) if it is *total* in S^c and if it satisfies the following two conditions.

1. $\forall S \in \mathcal{P}(S^a). (s_0^c \preceq_{BF} S \Rightarrow s_0^a \in S)$
2. $\forall S_2 \in \mathcal{P}(S^a). \forall s_1', s_2' \in S^c. \forall e \in E.$
$$\left((s_1' \xrightarrow{e}_{T^c} s_2' \wedge s_2' \preceq_{BF} S_2) \Rightarrow \exists S_1 \in \mathcal{P}(S^a). \forall s_1 \in S_1. \exists \gamma \in E^*. \exists s_2 \in S_2. \right.$$
$$\left. (s_1' \preceq_{BF} S_1 \wedge \gamma|_{I \cup O} = \langle e \rangle|_{I \cup O} \wedge s_1 \overset{\gamma}{\Longrightarrow}_{T^a} s_2) \right) \quad \diamond$$

Theorem C.1.10 (Soundness of hybrid simulations). Let $\preceq_{FB} \subseteq S^c \times \mathcal{N}(S^a)$ and $\preceq_{BF} \subseteq S^c \times \mathcal{P}(S^a)$ be two relations.

- If $SES^c \preceq_{FB} SES^a$ then $SES^c \leq_{OT} SES^a$ holds.
- If $SES^c \preceq_{BF} SES^a$ then $SES^c \leq_{OT} SES^a$ holds. $\diamond$

Proof. Follows from Theorems 4.5 and 4.13 in [LV95]. $\square$

Unlike forward simulation or backward simulation, which are only complete under certain assumptions, forward-backward simulation and backward-forward simulation are both, in general, complete verification techniques for observational trace refinement.

Theorem C.1.11 (Completeness of hybrid simulations).

- If $SES^c \leq_{OT} SES^a$ then $SES^c \preceq_{FB} SES^a$ holds for some relation $\preceq_{FB} \subseteq S^c \times \mathcal{N}(S^a)$.
- If $SES^c \leq_{OT} SES^a$ then $SES^c \preceq_{BF} SES^a$ holds for some relation $\preceq_{BF} \subseteq S^c \times \mathcal{P}(S^a)$. $\diamond$

Proof. Follows from Theorems 4.5 and 4.13 in [LV95]. $\square$

[2]Given a set X, $\mathcal{N}(X)$ denotes the powerset of X without $\emptyset$, i.e. $\mathcal{N}(X) = \{X' \mid X' \subseteq X \wedge X' \neq \emptyset\}$.
[3]Given a set X, $\mathcal{P}(X)$ denotes the *powerset* of X, i.e. $\mathcal{P}(X) = \{X' \mid X' \subseteq X\}$.

C.2 Correspondence to Trace Refinement

We now elaborate the correspondence between BSPs and observational trace refinement.

The following theorem shows how our backwards-strict BSPs can be reformulated by refinement statements. This theorem provides the basis for employing simulation techniques in the verification of BSPs.

Theorem C.2.1. Let $ES_{SES} = (E, I, O, Tr_{SES})$ be the event system for *SES*. The following equivalences are valid where $T' = \{(s, e, s') \in T \mid e \notin C\}$:

$$BSD_{\mathcal{V}}(Tr_{SES}) \iff \forall \beta \in E^*.\, \forall c \in C.\, \forall s, s' \in S. \quad \left(\begin{array}{l} (s_0 \stackrel{\beta}{\Longrightarrow}_T s \wedge s \stackrel{c}{\longrightarrow}_T s') \\ \Rightarrow (S, s', E, \emptyset, V, T') \leq_{OT} (S, s, E, \emptyset, V, T') \end{array} \right) \tag{C.1}$$

$$BSI_{\mathcal{V}}(Tr_{SES}) \iff \forall \beta \in E^*.\, \forall c \in C.\, \forall s \in S. \quad \left(\begin{array}{l} s_0 \stackrel{\beta}{\Longrightarrow}_T s \\ \Rightarrow \exists s' \in S.\, (s \stackrel{c}{\longrightarrow}_T s' \wedge \\ \qquad (S, s, E, \emptyset, V, T') \leq_{OT} (S, s', E, \emptyset, V, T')) \end{array} \right) \tag{C.2}$$

$$BSIA^{\rho}_{\mathcal{V}}(Tr_{SES}) \iff \forall \beta \in E^*.\, \forall c \in C.\, \forall s \in S. \quad \left(\begin{array}{l} (s_0 \stackrel{\beta}{\Longrightarrow}_T s \wedge En^{\rho}_{\mathcal{V}}(T, s, c)) \\ \Rightarrow \exists s' \in S.\, (s \stackrel{c}{\longrightarrow}_T s' \wedge \\ \qquad (S, s, E, \emptyset, V, T') \leq_{OT} (S, s', E, \emptyset, V, T')) \end{array} \right) \tag{C.3}$$

◇

Proof. We only prove (C.1) in detail.

$\Longrightarrow$: Assume that $BSD_{\mathcal{V}}(Tr_{SES})$ holds. Let $\beta \in E^*$, $c \in C$, and $s, s' \in S$ be arbitrary such that $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $s \stackrel{c}{\longrightarrow}_T s'$ hold. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that there is a state s'' with $s' \stackrel{\alpha}{\Longrightarrow}_{T'} s''$, i.e. α is a possible trace of $(S, s', E, \emptyset, V, T')$. Consequently, $\beta.\langle c \rangle.\alpha \in Tr_{SES}$ holds. From $BSD_{\mathcal{V}}(Tr_{SES})$ we obtain that there is a trace $\alpha' \in (E \setminus C)^*$ with $\beta.\alpha' \in Tr_{SES}$ and $\alpha'|_V = \alpha|_V$. Hence, there must be a state $s''' \in S$ for which $s \stackrel{\alpha'}{\Longrightarrow}_{T'} s'''$ holds, i.e. α' is a possible trace of $(S, s, E, \emptyset, V, T')$ that yields the same observation like α.

$\Longleftarrow$: Assume that, for all $\beta \in E^*$, $c \in C$, $s, s' \in S$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $s \stackrel{c}{\longrightarrow}_T s'$, $(S, s', E, \emptyset, V, T') \leq_{OT} (S, s, E, \emptyset, V, T')$ holds. Let $\beta \in E^*$ and $c \in C$ be arbitrary such that $\beta.\langle c \rangle \in Tr_{SES}$ holds. Hence, there are states $s, s' \in S$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $s \stackrel{c}{\longrightarrow}_T s'$. According to our assumption $(S, s', E, \emptyset, V, T') \leq_{OT} (S, s, E, \emptyset, V, T')$ holds. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $\beta.\langle c \rangle.\alpha \in Tr_{SES}$ holds. Hence, there is a state $s'' \in S$ with $s' \stackrel{\alpha}{\Longrightarrow}_{T'} s''$. Consequently, α is a possible trace of $(S, s', E, \emptyset, V, T')$. From the refinement statement, we obtain that there is a possible trace $\alpha' \in (E \setminus C)^*$ of $(S, s, E, \emptyset, V, T')$ that yields the same observation like α, i.e. there is a state $s''' \in S$ with $s \stackrel{\alpha'}{\Longrightarrow}_{T'} s'''$ (implies $\beta.\alpha' \in Tr_{SES}$) and $\alpha'|_V = \alpha|_V$. Consequently, $BSD_{\mathcal{V}}(Tr_{SES})$ holds.

The proofs of (C.2) and (C.3) are along the same lines. In the proof of (C.3), Theorem 5.4.7 needs to be applied in order to move between ρ-enabledness and ρ-admissibility. □

Note that the two state-event systems in each refinement statement of Theorem C.2.1 differ only in their initial state (either s or s'). In the refinement statements, the classification of

events as external or internal events and the choice of T' are crucial. For example, in the equivalence for $BSD_{\mathcal{V}}(Tr_{SES})$, events in $N \cup C$ become internal events and events in V become external events according to $(S, s, E, \emptyset, V, T') \leq_{OT} (S, s', E, \emptyset, V, T')$.[4] This reflects that α and α' in Definition 3.4.24 must be equal in events from V but may differ in events from $N \cup C$. However, in order to prevent that they differ in events from C, it is crucial that events from C are disabled in T'. This also ensures $\alpha \in (E \setminus C)^*$ and $\alpha' \in (E \setminus C)^*$.

The following theorem shows how our forward-correctable BSPs can be reformulated based on observational trace refinement.

Theorem C.2.2. Let $ES_{SES} = (E, I, O, Tr_{SES})$ be the event system for *SES*. The following implications are valid where $T' = \{(s, e, s') \in T \mid e \notin C\}$:

$$FCD^{\Gamma}_{\mathcal{V}}(Tr_{SES}) \Leftarrow \forall \beta \in E^*.\, \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla.\, \forall s, s' \in S. \quad \text{(C.4)}$$
$$\left(\begin{array}{l} (s_0 \stackrel{\beta}{\Longrightarrow}_T s \wedge s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s') \\ \Rightarrow \exists s'' \in S.\, \exists \delta \in (N \cap \Delta)^*. \\ \quad (s \stackrel{\delta.\langle v \rangle}{\Longrightarrow}_T s'' \wedge (S, s', E, \emptyset, V, T') \leq_{OT} (S, s'', E, \emptyset, V, T')) \end{array} \right)$$

$$FCI^{\Gamma}_{\mathcal{V}}(Tr_{SES}) \Leftarrow \forall \beta \in E^*.\, \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla.\, \forall s, s'' \in S. \quad \text{(C.5)}$$
$$\left(\begin{array}{l} (s_0 \stackrel{\beta}{\Longrightarrow}_T s \wedge s \stackrel{v}{\longrightarrow}_T s'') \\ \Rightarrow \exists s' \in S.\, \exists \delta \in (N \cap \Delta)^*. \\ \quad (s \stackrel{\langle c \rangle.\delta.\langle v \rangle}{\Longrightarrow}_T s' \wedge (S, s'', E, \emptyset, V, T') \leq_{OT} (S, s', E, \emptyset, V, T')) \end{array} \right)$$

$$FCIA^{\rho,\Gamma}_{\mathcal{V}}(Tr_{SES}) \Leftarrow \forall \beta \in E^*.\, \forall c \in C \cap \Upsilon.\, \forall v \in V \cap \nabla.\, \forall s, s'' \in S. \quad \text{(C.6)}$$
$$\left(\begin{array}{l} (s_0 \stackrel{\beta}{\Longrightarrow}_T s \wedge s \stackrel{v}{\longrightarrow}_T s'' \wedge En^{\rho}_{\mathcal{V}}(T, s, c)) \\ \Rightarrow \exists s' \in S.\, \exists \delta \in (N \cap \Delta)^*. \\ \quad (s \stackrel{\langle c \rangle.\delta.\langle v \rangle}{\Longrightarrow}_T s' \wedge (S, s'', E, \emptyset, V, T') \leq_{OT} (S, s', E, \emptyset, V, T')) \end{array} \right) \quad \diamond$$

Proof. We only prove (C.4) in detail.

Assume that for all $\beta \in E^*$, $c \in C \cap \Upsilon$, $v \in V \cap \nabla$, $s, s' \in S$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s'$ there are a state $s'' \in S$ and a sequence $\delta \in (N \cap \Delta)^*$ such that $s \stackrel{\delta.\langle v \rangle}{\Longrightarrow}_T s''$ and $(S, s', E, \emptyset, V, T') \leq_{OT} (S, s'', E, \emptyset, V, T')$ hold. Let $\beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary such that $\beta.\langle c.v \rangle \in Tr_{SES}$ holds. Hence, there are states $s, s' \in S$ with $s_0 \stackrel{\beta}{\Longrightarrow}_T s$ and $s \stackrel{\langle c.v \rangle}{\Longrightarrow}_T s'$. According to our assumption there are a state $s'' \in S$ and a sequence $\delta \in (N \cap \Delta)^*$ such that $s \stackrel{\delta.\langle v \rangle}{\Longrightarrow}_T s''$ and $(S, s', E, \emptyset, V, T') \leq_{OT} (S, s'', E, \emptyset, V, T')$ hold. Let $\alpha \in (E \setminus C)^*$ be arbitrary such that $\beta.\langle c.v \rangle.\alpha \in Tr_{SES}$ holds. Hence, there is a state $s''' \in S$ with $s' \stackrel{\alpha}{\Longrightarrow}_{T'} s'''$. Consequently, α is a possible trace of the state-event system $(S, s', E, \emptyset, V, T')$. From the refinement statement, we obtain that there is a possible trace $\alpha' \in (E \setminus C)^*$ of the state-event system $(S, s'', E, \emptyset, V, T')$ that yields the same observation like α, i.e. there is a state $s'''' \in S$ with $s'' \stackrel{\alpha'}{\Longrightarrow}_{T'} s''''$ (implies $\beta.\delta.\langle v \rangle.\alpha' \in Tr_{SES}$) and $\alpha'|_V = \alpha|_V$.

The proofs of (C.5) and (C.6) are along the same lines. In the proof of (C.6), Theorem 5.4.7 needs to be applied in order to move between ρ-admissibility and ρ-enabledness. $\square$

[4] However, it does not matter whether events in V are viewed as input events or as output events. We view these events as output events (cf. Theorem C.2.1).

Remark C.2.3. According to Theorem C.2.1, the backwards-strict BSPs are equivalent to their respective reformulations by refinement statements. In contrast to this, our reformulations of forward-correctable BSPs are not equivalent to the respective BSPs (cf. Theorem C.2.2), i.e. these BSPs are implied by the respective reformulation but they do not imply this reformulation. This is due to the quantifier structure, i.e. $\forall\alpha\ldots\exists\delta\ldots$ in the definitions of *FCD*, *FCI*, or *FCIA* and the quantifier structure $\ldots\exists\delta\ldots\forall\alpha\ldots$ in the reformulations (universal quantification of α is implicit in the refinement statement). ◇

C.3 Verifying BSPs by Simulation

In Theorems C.2.1 and C.2.2, we have shown how backwards-strict BSPs and forward-correctable BSPs can be expressed by refinement statements. The benefit of having these reformulations is that simulation techniques are applicable to them. This means, we can employ any of the simulation techniques in Section C.1 for proving these reformulations. According to the soundness theorems (Theorems C.1.4, C.1.6, and C.1.10), the application of forward simulation, backward simulation, forward-backward simulation, and backward-forward simulation during the verification of information flow properties is, indeed, sound. Interestingly, after having shown Theorems C.2.1 and C.2.2, we obtain these soundness results "for free" from [LV95]. Moreover, simulation techniques that are complete proof techniques for refinement statements are also complete proof techniques for our backwards-strict BSPs (because these BSPs are equivalent to their respective reformulations).

Let us now illustrate by a concrete example how simulation techniques can be applied during the verification of BSPs. We start with forward simulation and then present an example for backward simulation. For both examples, we employ the same system that we have also used to illustrate the application of our unwinding results in Section 5.6.1.

Example C.3.1. Let $SES = (S, s_0, E, I, O, T)$ and $\mathcal{V} = (V, N, C)$ be defined like in Example 5.6.1. The transition relation T is also viewed in Figure C.3. Let us now verify that $BSD_{\mathcal{V}}(Tr_{SES})$ holds with the help of forward simulation.

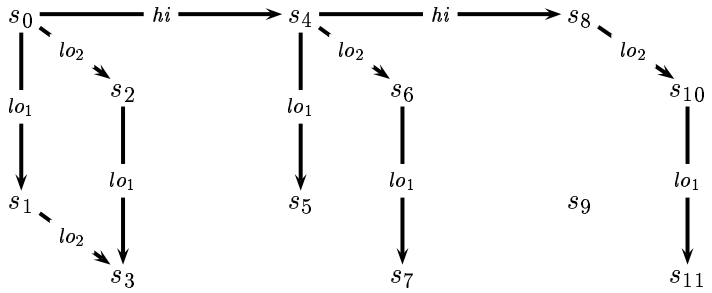


Figure C.3: Transition relation from Example C.3.1

According to Theorem C.2.1, we have to prove two refinement statements, namely

$$\begin{array}{lll}(S, s_4, E, \emptyset, V, T') & \leq_{OT} & (S, s_0, E, \emptyset, V, T') \\ (S, s_8, E, \emptyset, V, T') & \leq_{OT} & (S, s_4, E, \emptyset, V, T')\end{array}$$

where T' is defined by $T' = \{(s, e, s') \in T \mid e \neq hi\}$. We now construct a forward simulation $\preceq_{F1} \subseteq S \times S$ for the first of these statements by starting with the empty relation and iteratively

adding elements to this relation in order to satisfy the requirements of a forward simulation. The first condition in the definition of forward simulations (cf. Definition C.1.3) requires $s_4 \preceq_{F1} s_0$. Since $(s_4, lo_1, s_5) \in T'$ and $s_4 \preceq_{F1} s_0$, the second condition in Definition C.1.3 requires $s_5 \preceq_{F1} s_1$. By repeatedly applying this second condition, we obtain that $s_6 \preceq_{F1} s_2$ and $s_7 \preceq_{F1} s_3$ have to hold as well. It is easy to check that $\preceq_F = \{(s_4, s_0), (s_5, s_1), (s_6, s_2), (s_7, s_3)\}$, indeed, is a forward simulation for the first of the two refinement statements above. A forward simulation $\preceq_{F2} \subseteq S \times S$ for the second refinement statement can be constructed along the same lines. The forward simulations $\preceq_{F1}$ and $\preceq_{F2}$ are depicted in Figure C.4. $\diamond$

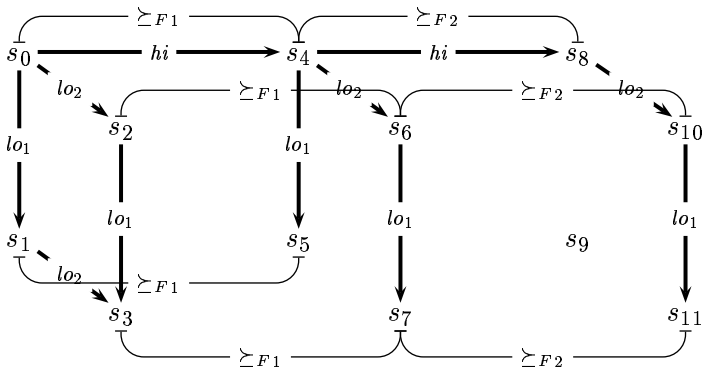


Figure C.4: Construction of a forward simulation in Example C.3.1

The difference between the above example and our example for the unwinding technique (Example 5.6.1) is that there are two simulation relations $\preceq_{F1}$ and $\preceq_{F2}$ but only a single unwinding relation $\ltimes$. Interestingly, the forward simulations selected in the example are not the only ones that are suitable. For example, the verification of the conditions for forward simulations would have also succeeded for the choices $\preceq_{F1} = \ltimes$ and $\preceq_{F2} = \ltimes$ (where $\ltimes$ is like in Example 5.6.1).

While the construction of a forward simulation is similar to the construction of an unwinding relation, the construction of a backward simulation is quite different. The main reason for this difference is the requirement that a backward simulation must be *total* in the states of the concrete system, i.e. that *every* state of the concrete system must be related to some state of the abstract system.

Example C.3.2. Let $SES = (S, s_0, E, I, O, T)$ and $\mathcal{V} = (V, N, C)$ be defined like in Example 5.6.1. Let us now verify that $BSD_{\mathcal{V}}(Tr_{SES})$ holds with the help of backward simulation.

Like in Example C.3.1, we have to prove the following two refinement statements:

$$\begin{array}{lll} (S, s_4, E, \emptyset, V, T') & \leq_{OT} & (S, s_0, E, \emptyset, V, T') \\ (S, s_8, E, \emptyset, V, T') & \leq_{OT} & (S, s_4, E, \emptyset, V, T') \end{array}$$

where T' is defined by $T' = \{(s, e, s') \in T \mid e \neq hi\}$. We now construct a backward simulation $\preceq_B \subseteq S \times S$ for the first of these refinement statements by starting with the empty relation and iteratively adding elements to this relation in order to satisfy the requirements of a backward simulation. Since $\preceq_B$ must be total in S, s_4 (the initial state of the concrete system) must be related to *some* state. According to the first condition in the definition of backward simulation (cf. Definition C.1.5), s_0 is the only possibility, i.e. $s_4 \preceq_B s_0$ must hold. Since $s_4 \preceq_B s_0$, $(s_4, lo_1, s_5) \in T'$, and $(s_0, lo_1, s_1) \in T'$, the second condition in the definition of backward simulation is satisfied if we demand that $s_5 \preceq_B s_1$ must hold. By

repeatedly, extending the relation, we obtain that $s_6 \preceq_B s_2$ and $s_7 \preceq_B s_3$ hold also. So far, the constructed relation is identical to the forward simulation $\preceq_F$. However, the requirements of a backward simulation are not yet satisfied because the relation constructed so far is not total in S, i.e. the states $s_0, s_1, s_2, s_3, s_8, s_9, s_{10}, s_{11}$ are not related to any other states. We relate s_8, s_{10}, s_{11} like in $\preceq_{F2}$ (cf. Example C.3.1), i.e. $s_8 \preceq_B s_4$, $s_{10} \preceq_B s_6$, $s_{11} \preceq_B s_7$ and relate s_0, s_1, s_2, s_3, s_9 to themselves, i.e. $s_0 \preceq_B s_0$, $s_1 \preceq_B s_1$, $s_2 \preceq_B s_2$, $s_3 \preceq_B s_3$, $s_9 \preceq_B s_9$. It is easy to check that the resulting relation (depicted in Figure C.5) is a backward simulation for the first refinement statement. Moreover, it is also a backward simulation for the second refinement statement. ◇

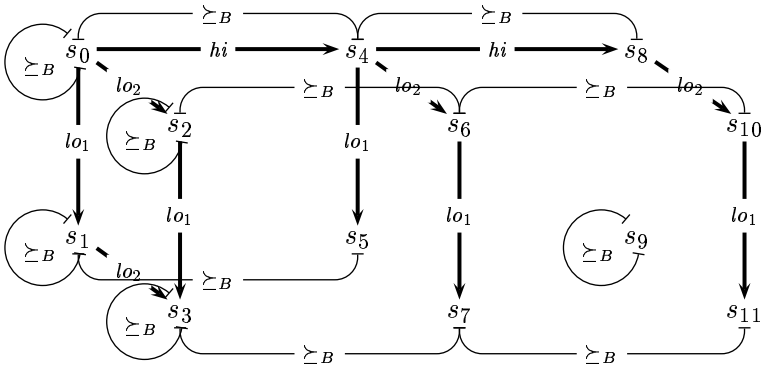


Figure C.5: Construction of a backward simulation in Example C.3.2

C.4 Summary

As an alternative to unwinding for the verification of information flow properties, we have adopted simulation techniques. Simulation techniques can be applied for this purpose despite they have been originally developed for a different purpose, i.e. for proving that one system refines another system. The adoptation of simulation techniques like, e.g., forward simulation or backward simulation has become possible after we identified a close relationship between our BSPs and a particular variant of trace refinement (referred to by *observational trace refinement*). The discovery of this correspondence and the adoptation of simulation techniques for verifying information flow properties are original contributions of our work.

Using simulation techniques, the verification of information flow properties proceeds as follows: First, the verification of information flow properties is reduced to the verification of BSPs (based on the modular representation in *MAKS*). Then, the BSPs are reformulated by refinement statements and, finally, they can be verified with the help of the simulation techniques explained in this chapter.

The simulation techniques considered here are already established for verifying trace refinement and our definitions of these techniques are simple re-formulations (in our formalism) of definitions from the literature [LV95]. The benefit of our adaptation of these simulation techniques is that we obtained alternatives to the unwinding technique for verifying BSPs. Since simulation techniques have been thoroughly investigated in the context of trace refinement, we could obtain soundness and completeness results for these verification techniques "for free". Concerning completeness, there seems to be a trade-off with the complexity of the local verification conditions. On the one hand, forward simulation and backward simulation

are based on relatively simple local conditions that are easy to handle during verification but these techniques are not complete, in general. On the other hand, forward-backward simulation and backward-forward simulation are complete verification techniques but the corresponding local verification conditions are quite complex (e.g. they involve relations between sets of states) and, hence, they are more difficult to handle during verification than the verification conditions for forward simulation or backward simulation.

Appendix D

Proofs of Compositionality Results

D.1 Detailed Proof of the Generalized Zipping Lemma

This section contains the detailed proof of the generalized zipping lemma (Lemma 6.4.4) that has been sketched in Section 6.4.2 already.

Proof (of Lemma 6.4.4). According to our assumptions, the composition of ES_1 and ES_2 is well behaved wrt. $\mathcal{V}_1$ and $\mathcal{V}_2$. We make a case distinction on conditions 1–4 in Definition 6.3.6 and prove each of the resulting cases. However, first, note the following facts:

- $N_1 \cap V_2 = \emptyset$ and $N_2 \cap V_1 = \emptyset$ hold because
  $N_1 \cap V_2 = (N_1 \cap E_1) \cap (V \cap E_2) = (N_1 \cap V_1) \cap E_2 = \emptyset$ and
  $N_2 \cap V_1 = (N_2 \cap E_2) \cap (V \cap E_1) = (N_2 \cap V_2) \cap E_1 = \emptyset$.
- If $N_1 \cap E_2 = \emptyset$ then $t_1|_{E_2} = \lambda|_{E_1 \cap E_2}$ holds because
  $t_1|_{E_2} = t_1|_{E_1 \cap E_2} = t_1|_{(V_1 \cup N_1 \cup C_1) \cap E_2} = t_1|_{V_1 \cap E_2} = \lambda|_{E_1 \cap E_2}$ ($t_1|_{C_1} = \langle\rangle$ holds).
- If $N_2 \cap E_1 = \emptyset$ then $t_2|_{E_1} = \lambda|_{E_1 \cap E_2}$ holds because
  $t_2|_{E_1} = t_2|_{E_1 \cap E_2} = t_2|_{E_1 \cap (V_2 \cup N_2 \cup C_2)} = t_2|_{E_1 \cap V_2} = \lambda|_{E_1 \cap E_2}$ ($t_2|_{C_2} = \langle\rangle$ holds).

Let us, firstly, assume that condition 1 in Definition 6.3.6 holds, i.e. that $N_1 \cap E_2 = \emptyset$ and $N_2 \cap E_1 = \emptyset$ hold. According to the above facts, $t_1|_{E_2} = \lambda|_{E_1 \cap E_2} = t_2|_{E_1}$ holds. From $t_1|_{C_1} = \langle\rangle$, $t_2|_{C_2} = \langle\rangle$, $C \cap E_1 \subseteq C_1$, and $C \cap E_2 \subseteq C_2$, we obtain that $t_1|_C = \langle\rangle$ and $t_2|_C = \langle\rangle$ hold. Since $t_1|_{E_2} = t_2|_{E_1}$, $t_1|_V = \lambda|_{E_1}$, $t_2|_V = \lambda|_{E_2}$, $t_1|_C = \langle\rangle$, and $t_2|_C = \langle\rangle$ there is a trace $t \in E^*$ such that $t|_{E_1} = t_1$, $t|_{E_2} = t_2$, $t|_V = \lambda$, and $t|_C = \langle\rangle$. This means, t results from λ by inserting the sequences $t_1|_{N_1}$ and $t_2|_{N_2}$. According to the definition of the composition operator, $\tau.t \in Tr$ holds because $t|_{E_1} = t_1$, $t|_{E_2} = t_2$, $\tau|_{E_1}.t_1 \in Tr_1$, and $\tau|_{E_2}.t_2 \in Tr_2$ hold.

Let us, secondly, assume that condition 2 in Definition 6.3.6 holds, i.e. that $N_1 \cap E_2 = \emptyset$, $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr_1)$, and $total(ES_1, C_1 \cap N_2)$ hold. From $N_1 \cap E_2 = \emptyset$, we obtain $t_1|_{E_2} = \lambda|_{E_1 \cap E_2}$. The projections $t_1|_{E_2}$ and $t_2|_{E_1}$ differ only in events from $N_2 \cap E_1$ because $t_1|_{E_2} = \lambda|_{E_1 \cap E_2}$, $t_2 \in (V_2 \cup N_2 \cup C_2)^*$, $t_2|_{C_2} = \langle\rangle$, and $t_2|_{V_2} = \lambda|_{E_2}$. Since $N_2 \cap N_1 = \emptyset$, $N_2 \cap V_1 = \emptyset$, and $E_1 = V_1 \cup N_1 \cup C_1$, we have $N_2 \cap E_1 = N_2 \cap C_1$. By an inductive argument on the length of $t_2|_{N_2 \cap C_1}$, we obtain from $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr_1)$, $total(ES_1, C_1 \cap N_2)$ and $N_1 \cap E_2 = \emptyset$ that there is a trace $t'_1 \in E_1^*$ with $\tau|_{E_1}.t'_1 \in Tr_1$, $t'_1|_{V_1} = t_1|_{V_1}$, $t'_1|_{C_1} \in (C_1 \cap N_2)^*$, and $t'_1|_{E_2} = t_2|_{E_1}$ (events in $t_2|_{N_2 \cap C_1}$ are inserted from left to right into t_1). From $t'_1|_{C_1} \in (C_1 \cap N_2)^*$, $t_2|_{C_2} = \langle\rangle$, $C \cap E_1 \subseteq C_1$, and $C \cap E_2 \subseteq C_2$, we obtain that $t'_1|_C = \langle\rangle$ and $t_2|_C = \langle\rangle$ hold. Since $t'_1|_{E_2} = t_2|_{E_1}$, $t'_1|_V = \lambda|_{E_1}$ (follows from $t'_1|_{V_1} = t_1|_{V_1}$ and $t_1|_V = \lambda|_{E_1}$), $t_2|_V = \lambda|_{E_2}$, $t'_1|_C = \langle\rangle$, and $t_2|_C = \langle\rangle$ there is a

trace $t \in E^*$ with $t|_{E_1} = t'_1$, $t|_{E_2} = t_2$, $t|_V = \lambda$, and $t|_C = \langle\rangle$. According to the definition of the composition operator, $\tau.t \in Tr$ holds ($\tau|_{E_1}.t'_1 \in Tr_1$, $\tau|_{E_2}.t_2 \in Tr_2$, $t|_{E_1} = t'_1$, and $t|_{E_2} = t_2$).

Let us, thirdly, assume that condition 3 in Definition 6.3.6 holds, i.e. that $N_2 \cap E_1 = \emptyset$, $BSIA^{\rho_2}_{\mathcal{V}_2}(Tr_2)$, and $total(ES_2, C_2 \cap N_1)$ hold. These assumptions are symmetric to the one in the second case (where we assumed that condition 2 holds). Hence, this case can be handled like the second case (where indices 1 and 2 have to be exchanged).

Let us, fourthly, assume that condition 4 in Definition 6.3.6 holds, i.e. that there are a function ρ_1 from views in E_1 to subsets of E_1, a function ρ_2 from views in E_2 to subsets of E_2, and triples $\Gamma_1 = (\nabla_1, \Delta_1, \Upsilon_1)$ and $\Gamma_2 = (\nabla_2, \Delta_2, \Upsilon_2)$ (where $\nabla_1, \Delta_1, \Upsilon_1 \subseteq E_1$ and $\nabla_2, \Delta_2, \Upsilon_2 \subseteq E_2$) such that $BSIA^{\rho_1}_{\mathcal{V}_1}(Tr_1)$, $BSIA^{\rho_2}_{\mathcal{V}_2}(Tr_2)$, $total(ES_1, C_1 \cap N_2)$, $total(ES_2, C_2 \cap N_1)$, $FCIA^{\rho_1,\Gamma_1}_{\mathcal{V}_1}(Tr_1)$, $FCIA^{\rho_2,\Gamma_2}_{\mathcal{V}_2}(Tr_2)$, $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$, $C_1 \cap N_2 \subseteq \Upsilon_1$, $C_2 \cap N_1 \subseteq \Upsilon_2$, $N_1 \cap \Delta_1 \cap E_2 = \emptyset$, and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ hold.

We prove the lemma by induction on the length of λ. This induction resembles the process of closing a zipper in a stepwise manner if the events in λ are viewed as the teeth of the zipper.

In the *base case* ($\lambda = \langle\rangle$), $\tau.t \in Tr$, $t|_V = \lambda$, and $t|_C = \langle\rangle$ hold for $t = \langle\rangle$.

In the *step case* ($\lambda = \langle v\rangle.\lambda'$ for some $v \in V$, $\lambda' \in V^*$), we make a case distinction on v.

1. $v \in V_1 \cap V_2 \cap \nabla_1$ 2. $v \in V_1 \cap V_2 \cap \nabla_2$ 3. $v \in V_1 \setminus E_2$ 4. $v \in V_2 \setminus E_1$

This is a complete case distinction because $v \in V$, $V = V_1 \cup V_2 = (V_1 \cap V_2) \cup (V_1 \setminus V_2) \cup (V_2 \setminus V_1)$, $V_1 \setminus V_2 = V_1 \setminus E_2$ (follows from $V \cap E_1 = V_1$ and $V \cap E_2 = V_2$), $V_2 \setminus V_1 = V_2 \setminus E_1$ (follows from $V \cap E_1 = V_1$ and $V \cap E_2 = V_2$), and $V_1 \cap V_2 \subseteq \nabla_1 \cup \nabla_2$.

Case 1: $v \in V_1 \cap V_2 \cap \nabla_1$

- We split t_1 into subsequences before and after v (recall that $t_1|_V = \lambda|_{E_1}$). Choose $r_1, s_1 \in E_1^*$ such that $t_1 = r_1.\langle v\rangle.s_1$ and $r_1|_{V_1} = \langle\rangle$ hold.

- $r_1|_{C_1} = \langle\rangle$ holds because $t_1|_{C_1} = \langle\rangle$. Hence, $r_1 \in N_1^*$ holds (follows from $r_1 \in E_1^*$ and $r_1|_{V_1} = \langle\rangle$). $r_1|_{E_2} \in (N_1 \cap C_2)^*$ holds because $N_1 \cap V_2 = \emptyset$ and $N_1 \cap N_2 = \emptyset$. From $C_2 \cap N_1 \subseteq \Upsilon_2$ we obtain that $r_1|_{E_2} \in (N_1 \cap C_2 \cap \Upsilon_2)^*$ holds.

- We insert $r_1|_{E_2}$ into $\tau|_{E_2}.t_2 \in Tr_2$ after $\tau|_{E_2}$. By an inductive argument, we obtain from $BSIA^{\rho_2}_{\mathcal{V}_2}(Tr_2)$ and $total(ES_2, C_2 \cap N_1)$ that $t'_2 \in E_2^*$ exists with $(\tau.r_1)|_{E_2}.t'_2 \in Tr_2$, $t'_2|_{V_2} = t_2|_{V_2}$, and $t'_2|_{C_2} = \langle\rangle$ (events in $r_1|_{E_2}$ are inserted from left to right into $\tau|_{E_2}.t_2$).

- We split t'_2 into subsequences before and after v (recall that $\lambda|_{E_2} = t_2|_V = t'_2|_V$). Choose $r'_2, s'_2 \in E_2^*$ such that $t'_2 = r'_2.\langle v\rangle.s'_2$ and $r'_2|_{V_2} = \langle\rangle$ hold. $r'_2|_{C_2} = \langle\rangle$ and $s'_2|_{C_2} = \langle\rangle$ hold because $t'_2|_{C_2} = \langle\rangle$.

- Since $r'_2|_{C_2} = \langle\rangle$ we have $r'_2 \in N_2^*$ (recall that $r'_2 \in E_2^*$ and $r'_2|_{V_2} = \langle\rangle$). $r'_2|_{E_1} \in (N_2 \cap C_1)^*$ holds because $N_2 \cap V_1 = \emptyset$ and $N_2 \cap N_1 = \emptyset$. Since $C_1 \cap N_2 \subseteq \Upsilon_1$, we have $r'_2|_{E_1} \in (N_2 \cap C_1 \cap \Upsilon_1)^*$.

- We insert $r'_2|_{E_1}$ into $\tau|_{E_1}.r_1.\langle v\rangle.s_1 \in Tr_1$ after $\tau|_{E_1}.r_1$. By an inductive argument on the length of $r'_2|_{E_1}$, we obtain from $FCIA^{\rho_1,\Gamma_1}_{\mathcal{V}_1}(Tr_1)$ and $total(ES_1, C_1 \cap N_2)$ that there are $s'_1 \in E_1^*$ and $q'_1 \in ((C_1 \cap \Upsilon_1) \cup (N_1 \cap \Delta_1))^*$ with $\tau|_{E_1}.r_1.q'_1.\langle v\rangle.s'_1 \in Tr_1$, $q'_1|_{C_1 \cap \Upsilon_1} = r'_2|_{E_1}$, $s'_1|_{V_1} = s_1|_{V_1}$, and $s'_1|_{C_1} = \langle\rangle$ (events in $r'_2|_{E_1}$ are inserted from left to right). Note that q'_1 need not be identical to $r'_2|_{E_1}$, however, it may differ only in events from $N_1 \cap \Delta_1$.

- From $r_2' \in N_2^*$ and $C \cap E_2 \subseteq C_2$, we obtain $r_2'|_C = \langle\rangle$. $q_1'|_C = \langle\rangle$ follows from $q_1' \in ((C_1 \cap \Upsilon_1) \cup (N_1 \cap \Delta_1))^*$, $q_1'|_{C_1 \cap \Upsilon_1} = r_2'|_{E_1}$, $r_2' \in N_2^*$, and $C \cap E_2 \subseteq C_2$.

- $q_1'|_{E_2} = r_2'|_{E_1}$ holds because $q_1'|_{E_2} = q_1'|_{((C_1 \cap \Upsilon_1) \cup (N_1 \cap \Delta_1)) \cap E_2} = q_1'|_{(C_1 \cap \Upsilon_1) \cap E_2} = r_2'|_{E_1 \cap E_2} = r_2'|_{E_1}$ (follows from $q_1' \in ((C_1 \cap \Upsilon_1) \cup (N_1 \cap \Delta_1))^*$, $N_1 \cap \Delta_1 \cap E_2 = \emptyset$, $r_2' \in E_2^*$). Since $q_1'|_{E_2} = r_2'|_{E_1}$, $q_1'|_V = \langle\rangle$ (follows from $V \cap E_1 = V_1$ and $q_1' \in ((C_1 \cap \Upsilon_1) \cup (N_1 \cap \Delta_1))^*$), $r_2'|_V = \langle\rangle$ (follows from $V \cap E_2 = V_2$ and $r_2'|_{V_2} = \langle\rangle$), $q_1'|_C = \langle\rangle$, and $r_2'|_C = \langle\rangle$ there is a trace $q' \in E^*$ with $q'|_{E_1} = q_1'$, $q'|_{E_2} = r_2'$, $q'|_V = \langle\rangle$, and $q'|_C = \langle\rangle$.

- The preconditions of the induction hypothesis are satisfied for $\tau.r_1.q'.\langle v\rangle$, λ', s_1', s_2':

  - $(\tau.r_1.q'.\langle v\rangle)|_{E_1}.s_1' \in Tr_1$ and $(\tau.r_1.q'.\langle v\rangle)|_{E_2}.s_2' \in Tr_2$ hold because $r_1 \in E_1^*$, $q'|_{E_1} = q_1'$, $\tau|_{E_1}.r_1.q_1'.\langle v\rangle.s_1' \in Tr_1$, $q'|_{E_2} = r_2'$, $(\tau.r_1)|_{E_2}.r_2'.\langle v\rangle.s_2' \in Tr_2$, and $v \in E_1 \cap E_2$.
  - $\lambda'|_{E_1} = s_1'|_V$ and $\lambda'|_{E_2} = s_2'|_V$ hold because $(\langle v\rangle.\lambda')|_{E_1} = t_1|_V = (r_1.\langle v\rangle.s_1)|_V$, $r_1|_V = \langle\rangle$, $s_1'|_V = s_1|_V$, $(\langle v\rangle.\lambda')|_{E_2} = t_2|_V = t_2'|_V = (r_2'.\langle v\rangle.s_2')|_V$, and $r_2'|_V = \langle\rangle$.
  - $s_1'|_{C_1} = \langle\rangle$ and $s_2'|_{C_2} = \langle\rangle$ hold.

- From the induction hypothesis, we obtain that there is a trace $t' \in E^*$ for which $\tau.r_1.q'.\langle v\rangle.t' \in Tr$, $t'|_V = \lambda'$, and $t'|_C = \langle\rangle$.

- We now show that the conclusion of the lemma holds for $t = r_1.q'.\langle v\rangle.t'$:

  - $\tau.t \in Tr$ holds because $\tau.r_1.q'.\langle v\rangle.t' \in Tr$.
  - $t|_V = \lambda$ because $r_1|_V = \langle\rangle$, $q'|_V = \langle\rangle$, $t'|_V = \lambda'$, and $\lambda = \langle v\rangle.\lambda'$.
  - $t|_C = \langle\rangle$ because $r_1|_C = \langle\rangle$, $q'|_C = \langle\rangle$, $\langle v\rangle|_C = \langle\rangle$, and $t'|_C = \langle\rangle$.

Case 2 ($v \in V_1 \cap V_2 \cap \nabla_2$) can be shown like case 1 (with indices 1 and 2 exchanged).

Case 3: $v \in V_1 \setminus E_2$

- We split t_1 into subsequences before and after v (recall that $t_1|_V = \lambda|_{E_1}$). Choose $r_1, s_1 \in E_1^*$ such that $t_1 = r_1.\langle v\rangle.s_1$ and $r_1|_{V_1} = \langle\rangle$ hold.

- $r_1|_{C_1} = \langle\rangle$ and $s_1|_{C_1} = \langle\rangle$ hold because $t_1|_{C_1} = \langle\rangle$. Hence, $r_1 \in N_1^*$ holds (follows from $r_1 \in E_1^*$ and $r_1|_{V_1} = \langle\rangle$). $r_1|_{E_2} \in (N_1 \cap C_2)^*$ holds because $N_1 \cap V_2 = \emptyset$ and $N_1 \cap N_2 = \emptyset$. Since $C_2 \cap N_1 \subseteq \Upsilon_2$, we have $r_1|_{E_2} \in (N_1 \cap C_2 \cap \Upsilon_2)^*$.

- We insert $r_1|_{E_2}$ into $\tau|_{E_2}.t_2 \in Tr_2$ after $\tau|_{E_2}$. By an inductive argument on the length of $r_1|_{E_2}$, we obtain from $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$ and $total(ES_2, C_2 \cap N_1)$ that $t_2' \in E_2^*$ exists with $(\tau.r_1)|_{E_2}.t_2' \in Tr_2$, $t_2'|_{V_2} = t_2|_{V_2}$, and $t_2'|_{C_2} = \langle\rangle$ (events in $r_1|_{E_2}$ are inserted from left to right into $\tau|_{E_2}.t_2$).

- The preconditions of the induction hypothesis are satisfied for $\tau.r_1.\langle v\rangle$, λ', s_1, t_2':

  - $(\tau.r_1.\langle v\rangle)|_{E_1}.s_1 \in Tr_1$ and $(\tau.r_1.\langle v\rangle)|_{E_2}.t_2' \in Tr_2$ hold (recall $\langle v\rangle|_{E_2} = \langle\rangle$).
  - $\lambda'|_{E_1} = s_1|_V$ and $\lambda'|_{E_2} = t_2'|_V$ because $(\langle v\rangle.\lambda')|_{E_1} = t_1|_V = (r_1.\langle v\rangle.s_1)|_V$, $r_1|_V = \langle\rangle$, $(\langle v\rangle.\lambda')|_{E_2} = t_2|_V = t_2'|_V$, and $v \notin E_2$.
  - $s_1|_{C_1} = \langle\rangle$ and $t_2'|_{C_2} = \langle\rangle$ hold.

- From the induction hypothesis, we obtain that there is a trace $t' \in E^*$ for which $\tau.r_1.\langle v\rangle.t' \in Tr$, $t'|_V = \lambda'$, and $t'|_C = \langle\rangle$ hold.
- We now show that the conclusion of the lemma holds for $t = r_1.\langle v\rangle.t'$:
  - $\tau.t \in Tr$ holds because $\tau.r_1.\langle v\rangle.t' \in Tr$.
  - $t|_V = \lambda$ because $r_1|_V = \langle\rangle$, $t'|_V = \lambda'$, and $\lambda = \langle v\rangle.\lambda'$.
  - $t|_C = \langle\rangle$ because $r_1|_C = \langle\rangle$, $\langle v\rangle|_C = \langle\rangle$, and $t'|_C = \langle\rangle$.

Case 4 ($v \in V_2 \setminus E_1$) can be shown like case 3 (with indices 1 and 2 exchanged). □

D.2 Detailed Proof of Compositionality Theorems for BSPs

This section contains the detailed proofs of Theorems 6.4.1 and 6.4.2 (i.e. the compositionality results for BSPs) that have been sketched in Section 6.4.3 already.

Proof (of Theorem 6.4.1). We prove each of the propositions with the help of Lemma 6.4.4.

Proposition 1: Let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary with $\beta.\langle c\rangle.\alpha \in Tr$ and $\alpha|_C = \langle\rangle$. We have $(\beta.\langle c\rangle.\alpha)|_{E_1} \in Tr_1$ and $(\beta.\langle c\rangle.\alpha)|_{E_2} \in Tr_2$. By an inductive argument on the length of $(\langle c\rangle.\alpha)|_{C_1}$, we obtain from $BSD_{\mathcal{V}_1}(Tr_1)$ that there is a trace $\alpha'_1 \in E_1^*$ with $(\beta)|_{E_1}.\alpha'_1 \in Tr_1$, $\alpha'_1|_{V_1} = \alpha|_{V_1}$, and $\alpha'_1|_{C_1} = \langle\rangle$. By an inductive argument on the length of $(\langle c\rangle.\alpha)|_{C_2}$, we obtain from $BSD_{\mathcal{V}_2}(Tr_2)$ that there is a trace $\alpha'_2 \in E_2^*$ with $(\beta)|_{E_2}.\alpha'_2 \in Tr_2$, $\alpha'_2|_{V_2} = \alpha|_{V_2}$, and $\alpha'_2|_{C_2} = \langle\rangle$. Lemma 6.4.4 yields for $\tau = \beta$, $\lambda = \alpha|_V$, $t_1 = \alpha'_1$, and $t_2 = \alpha'_2$ that there is a trace $t \in E^*$ with $\beta.t \in Tr$, $t|_V = \alpha|_V$, and $t|_C = \langle\rangle$. Thus, $BSD_{\mathcal{V}}(Tr)$ holds.

Proposition 2: Let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary with $\beta.\alpha \in Tr$ and $\alpha|_C = \langle\rangle$. We have $(\beta.\alpha)|_{E_1} \in Tr_1$ and $(\beta.\alpha)|_{E_2} \in Tr_2$. By an inductive argument on the length of $\alpha|_{C_1}$, we obtain from $BSD_{\mathcal{V}_1}(Tr_1)$ that there is a trace $\alpha'_1 \in E_1^*$ with $\beta|_{E_1}.\alpha'_1 \in Tr_1$, $\alpha'_1|_{V_1} = \alpha|_{V_1}$, and $\alpha'_1|_{C_1} = \langle\rangle$. By an inductive argument on the length of $\alpha|_{C_2}$, we obtain from $BSD_{\mathcal{V}_2}(Tr_2)$ that there is a trace $\alpha'_2 \in E_2^*$ with $\beta|_{E_2}.\alpha'_2 \in Tr_2$, $\alpha'_2|_{V_2} = \alpha|_{V_2}$, and $\alpha'_2|_{C_2} = \langle\rangle$. If $\langle c\rangle|_{E_1} \neq \langle\rangle$ (or $\langle c\rangle|_{E_2} \neq \langle\rangle$) then an application of $BSI_{\mathcal{V}_1}(Tr_1)$ (or $BSI_{\mathcal{V}_2}(Tr_2)$, respectively) yields that there are traces $\alpha''_1 \in E_1^*$, $\alpha''_2 \in E_2^*$ with $(\beta.\langle c\rangle)|_{E_1}.\alpha''_1 \in Tr_1$, $\alpha''_1|_{V_1} = \alpha|_{V_1}$, $\alpha''_1|_{C_1} = \langle\rangle$, $(\beta.\langle c\rangle)|_{E_2}.\alpha''_2 \in Tr_2$, $\alpha''_2|_{V_2} = \alpha|_{V_2}$, and $\alpha''_2|_{C_2} = \langle\rangle$ (if $\langle c\rangle|_{E_1} = \langle\rangle$, or $\langle c\rangle|_{E_2} = \langle\rangle$ then choose $\alpha''_1 = \alpha'_1$, or $\alpha''_2 = \alpha'_2$, respectively). Lemma 6.4.4 yields for $\tau = \beta.\langle c\rangle$, $\lambda = \alpha|_V$, $t_1 = \alpha''_1$, and $t_2 = \alpha''_2$ that there is a trace $t \in E^*$ with $\beta.\langle c\rangle.t \in Tr$, $t|_V = \alpha|_V$, and $t|_C = \langle\rangle$. Thus, $BSI_{\mathcal{V}}(Tr)$ holds.

Proposition 3: Let $\alpha, \beta \in E^*$ and $c \in C$ be arbitrary with $\beta.\alpha \in Tr$, $\alpha|_C = \langle\rangle$, and $Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)$. Like in the proof of proposition 2, we obtain from $BSD_{\mathcal{V}_1}(Tr_1)$ and $BSD_{\mathcal{V}_2}(Tr_2)$ that there are traces $\alpha'_1 \in E_1^*$, $\alpha'_2 \in E_2^*$ with $\beta|_{E_1}.\alpha'_1 \in Tr_1$, $\alpha'_1|_{V_1} = \alpha|_{V_1}$, $\alpha'_1|_{C_1} = \langle\rangle$, $\beta|_{E_2}.\alpha'_2 \in Tr_2$, $\alpha'_2|_{V_2} = \alpha|_{V_2}$, and $\alpha'_2|_{C_2} = \langle\rangle$. $Adm^{\rho}_{\mathcal{V}}(Tr, \beta, c)$ implies that there is a trace $\gamma \in E^*$ with $\gamma|_{\rho(\mathcal{V})} = \beta|_{\rho(\mathcal{V})}$ and $\gamma.\langle c\rangle \in Tr$. We have $(\gamma|_{E_1})|_{\rho(\mathcal{V})} = (\beta|_{E_1})|_{\rho(\mathcal{V})}$ and $(\gamma|_{E_2})|_{\rho(\mathcal{V})} = (\beta|_{E_2})|_{\rho(\mathcal{V})}$. Since $\rho_1(\mathcal{V}_1) \subseteq \rho(\mathcal{V}) \cap E_1$ and $\rho_2(\mathcal{V}_2) \subseteq \rho(\mathcal{V}) \cap E_2$, also $(\gamma|_{E_1})|_{\rho_1(\mathcal{V}_1)} = (\beta|_{E_1})|_{\rho_1(\mathcal{V}_1)}$ and $(\gamma|_{E_2})|_{\rho_2(\mathcal{V}_2)} = (\beta|_{E_2})|_{\rho_2(\mathcal{V}_2)}$ hold. Consequently, if $c \in E_1$ (or $c \in E_2$) then, $Adm^{\rho_1}_{\mathcal{V}_1}(Tr_1, \beta|_{E_1}, c)$ (or $Adm^{\rho_2}_{\mathcal{V}_2}(Tr_2, \beta|_{E_2}, c)$, respectively) holds. The remainder of this proof is along the same lines like the proof of proposition 2.

Proposition 4: Let $\alpha, \beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary with $\beta.\langle c.v\rangle.\alpha \in Tr$ and $\alpha|_C = \langle\rangle$. We have $(\beta.\langle c.v\rangle.\alpha)|_{E_1} \in Tr_1$ and $(\beta.\langle c.v\rangle.\alpha)|_{E_2} \in Tr_2$. Like in the proof of proposition 2, we obtain from $BSD_{\mathcal{V}_1}(Tr_1)$ and $BSD_{\mathcal{V}_2}(Tr_2)$ that there are traces $\alpha'_1 \in E_1^*$, $\alpha'_2 \in E_2^*$ such that $(\beta.\langle c.v\rangle)|_{E_1}.\alpha'_1 \in Tr_1$, $\alpha'_1|_{V_1} = \alpha|_{V_1}$, $\alpha'_1|_{C_1} = \langle\rangle$, $(\beta.\langle c.v\rangle)|_{E_2}.\alpha'_2 \in Tr_2$, $\alpha'_2|_{V_2} = \alpha|_{V_2}$, and

$\alpha'_2|_{C_2} = \langle\rangle$ hold. Note that $c \in E_1$ (or $c \in E_2$) implies $c \in C_1 \cap \Upsilon_1$ (or $c \in C_2 \cap \Upsilon_2$, respectively) because $c \in C \cap \Upsilon$, $C \cap E_j \subseteq C_j$, and $\Upsilon \cap E_j \subseteq \Upsilon_j$ hold. Similarly, $v \in E_1$ (or $v \in E_2$) implies $v \in V_1 \cap \nabla_1$ (or $v \in V_2 \cap \nabla_2$, respectively) because $v \in V \cap \nabla$, $V \cap E_j = V_j$, and $\nabla \cap E_j \subseteq \nabla_j$ hold (for $j \in \{1,2\}$).

In order to show that, for $j \in \{1,2\}$, there are traces $\alpha''_j \in E_j^*$ and $\delta''_j \in (N_j \cap \Delta_j)^*$ such that $\beta|_{E_j}.\delta''_j.\langle v\rangle|_{E_j}.\alpha''_j \in Tr_j$, $\alpha''_j|_{V_j} = \alpha'_j|_{V_j}$, and $\alpha''_j|_{C_j} = \langle\rangle$ hold, we make a case distinction on $c \in E_j$ and $v \in E_j$: Firstly, assume that $c \notin E_j$ holds. In this case, $\beta|_{E_j}.\delta''_j.\langle v\rangle|_{E_j}.\alpha''_j \in Tr_j$, $\alpha''_j|_{V_j} = \alpha'_j|_{V_j}$, and $\alpha''_j|_{C_j} = \langle\rangle$ hold for $\alpha''_j = \alpha'_j$ and $\delta''_j = \langle\rangle$. Secondly, assume that $c \in E_j$ and $v \notin E_j$ hold. Since $c \in E_j$, we have $c \in C_j \cap \Upsilon_j$. An application of $BSD_{\mathcal{V}_j}(Tr_j)$ yields that there is a trace $\alpha''_j \in E_j^*$ with $\beta|_{E_j}.\alpha''_j \in Tr_j$, $\alpha''_j|_{V_j} = \alpha'_j|_{V_j}$, and $\alpha''_j|_{C_j} = \langle\rangle$. Hence, $\beta|_{E_j}.\delta''_j.\langle v\rangle|_{E_j}.\alpha''_j \in Tr_j$ holds for $\delta''_j = \langle\rangle$. Thirdly, assume that $c \in E_j$ and $v \in E_j$ hold. Since $c \in E_j$ and $v \in E_j$, we have $c \in C_j \cap \Upsilon_j$ and $v \in V_j \cap \nabla_j$. An application of $FCD_{\mathcal{V}_j}^{\Gamma_j}(Tr_j)$ yields that there are traces $\alpha''_j \in E_j^*$ and $\delta''_j \in (N_j \cap \Delta_j)^*$ such that $\beta|_{E_j}.\delta''_j.\langle v\rangle|_{E_j}.\alpha''_j \in Tr_j$, $\alpha''_j|_{V_j} = \alpha'_j|_{V_j}$, and $\alpha''_j|_{C_j} = \langle\rangle$ hold.

Since $\delta''_1 \in (N_1 \cap \Delta_1)^*$, $N_1 \cap \Delta_1 \cap E_2 = \emptyset$, $\delta''_2 \in (N_2 \cap \Delta_2)^*$, and $N_2 \cap \Delta_2 \cap E_1 = \emptyset$, we have $\delta''_1|_{E_2} = \langle\rangle$ and $\delta''_2|_{E_1} = \langle\rangle$. Consequently, $(\beta.\delta''_1.\delta''_2.\langle v\rangle)|_{E_1}.\alpha''_1 \in Tr_1$ and $(\beta.\delta''_1.\delta''_2.\langle v\rangle)|_{E_2}.\alpha''_2 \in Tr_2$ hold. Lemma 6.4.4 yields for $\tau = \beta.\delta''_1.\delta''_2.\langle v\rangle$, $\lambda = \alpha|_V$, $t_1 = \alpha''_1$, and $t_2 = \alpha''_2$ that there is a trace $t \in E^*$ with $\beta.\delta''_1.\delta''_2.\langle v\rangle.t \in Tr$, $t|_V = \alpha|_V$, and $t|_C = \langle\rangle$. Since $\delta''_1.\delta''_2 \in (N \cap \Delta)^*$ holds (follows from $\Delta \supseteq (\Delta_1 \cap N_1) \cup (\Delta_2 \cap N_2)$ and $N \supseteq N_1 \cup N_2$), we have $FCD_{\mathcal{V}}^{\Gamma}(Tr)$.

Proposition 5: Let $\alpha, \beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary with $\beta.\langle v\rangle.\alpha \in Tr$ and $\alpha|_C = \langle\rangle$. We have $(\beta.\langle v\rangle.\alpha)|_{E_1} \in Tr_1$ and $(\beta.\langle v\rangle.\alpha)|_{E_2} \in Tr_2$. Like in the proof of proposition 2, we obtain from $BSD_{\mathcal{V}_1}(Tr_1)$ and $BSD_{\mathcal{V}_2}(Tr_2)$ that there are traces $\alpha'_1 \in E_1^*$, $\alpha'_2 \in E_2^*$ such that $(\beta.\langle v\rangle)|_{E_1}.\alpha'_1 \in Tr_1$, $\alpha'_1|_{V_1} = \alpha|_{V_1}$, $\alpha'_1|_{C_1} = \langle\rangle$, $(\beta.\langle v\rangle)|_{E_2}.\alpha'_2 \in Tr_2$, $\alpha'_2|_{V_2} = \alpha|_{V_2}$, and $\alpha'_2|_{C_2} = \langle\rangle$ hold. Note that $c \in E_1$ (or $c \in E_2$) implies $c \in C_1 \cap \Upsilon_1$ (or $c \in C_2 \cap \Upsilon_2$, respectively) because $c \in C \cap \Upsilon$, $C \cap E_j \subseteq C_j$, and $\Upsilon \cap E_j \subseteq \Upsilon_j$ hold. Similarly, $v \in E_1$ (or $v \in E_2$) implies $v \in V_1 \cap \nabla_1$ (or $v \in V_2 \cap \nabla_2$, respectively) because $v \in V \cap \nabla$, $V \cap E_j = V_j$, and $\nabla \cap E_j \subseteq \nabla_j$ hold (for $j \in \{1,2\}$).

Without loss of generality, we assume $N_1 \cap \Delta_1 \cap E_2 = \emptyset$ and $N_2 \cap \Delta_2 \cap E_1 \subseteq \Upsilon_1$. The case $N_2 \cap \Delta_2 \cap E_1 = \emptyset$ and $N_1 \cap \Delta_1 \cap E_2 \subseteq \Upsilon_2$ can be handled similarly because it is symmetric.

In order to show that there are $\alpha''_2 \in E_2^*$, $\delta''_2 \in (N_2 \cap \Delta_2)^*$ such that $\beta|_{E_2}.\langle c\rangle|_{E_2}.\delta''_2.\langle v\rangle|_{E_2}.\alpha''_2 \in Tr_2$, $\alpha''_2|_{V_2} = \alpha'_2|_{V_2}$, and $\alpha''_2|_{C_2} = \langle\rangle$ hold, we make a case distinction on $c \in E_2$ and $v \in E_2$: Firstly, assume that $c \notin E_2$ holds. In this case, $\beta|_{E_2}.\langle c\rangle|_{E_2}.\delta''_2.\langle v\rangle|_{E_2}.\alpha''_2 \in Tr_2$, $\alpha''_2|_{V_2} = \alpha'_2|_{V_2}$, and $\alpha''_2|_{C_2} = \langle\rangle$ hold for $\alpha''_2 = \alpha'_2$ and $\delta''_2 = \langle\rangle$. Secondly, assume that $c \in E_2$ and $v \notin E_2$ hold. Since $c \in E_2$, we have $c \in C_2 \cap \Upsilon_2$. An application of $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$ (recall that ES_2 is total in $C_2 \cap \Upsilon_2$) yields that there is a trace $\alpha''_2 \in E_2^*$ with $\beta|_{E_2}.\langle c\rangle.\alpha''_2 \in Tr_2$, $\alpha''_2|_{V_2} = \alpha'_2|_{V_2}$, and $\alpha''_2|_{C_2} = \langle\rangle$. Hence, $\beta|_{E_2}.\langle c\rangle|_{E_2}.\delta''_2.\langle v\rangle|_{E_2}.\alpha''_2 \in Tr_2$ holds for $\delta''_2 = \langle\rangle$. Thirdly, assume that $c \in E_2$ and $v \in E_2$ hold. Since $c \in E_2$ and $v \in E_2$, we have $c \in C_2 \cap \Upsilon_2$ and $v \in V_2 \cap \nabla_2$. An application of $FCI_{\mathcal{V}_2}^{\Gamma_2}(Tr_2)$ yields that there are traces $\alpha''_2 \in E_2^*$ and $\delta''_2 \in (N_2 \cap \Delta_2)^*$ such that $\beta|_{E_2}.\langle c\rangle|_{E_2}.\delta''_2.\langle v\rangle|_{E_2}.\alpha''_2 \in Tr_2$, $\alpha''_2|_{V_2} = \alpha'_2|_{V_2}$, and $\alpha''_2|_{C_2} = \langle\rangle$ hold.

Note that $\langle c\rangle|_{E_1} \in (C_1 \cap \Upsilon_1)^*$ holds because $c \in E_1$ implies $c \in C_1 \cap \Upsilon_1$. Moreover, $\delta''_2|_{E_1} \in (C_1 \cap \Upsilon_1)^*$ holds because $\delta''_2 \in (N_2 \cap \Delta_2)^*$, $N_2 \cap \Delta_2 \cap E_1 \subseteq \Upsilon_1$, $N_2 \cap N_1 = \emptyset$, and $N_2 \cap V_1 = \emptyset$.

In order to show that there are traces $\alpha''_1 \in E_1^*$, $\delta''_1 \in ((N_1 \cap \Delta_1) \cup (C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2))^*$ such that $\beta|_{E_1}.\langle c\rangle|_{E_1}.\delta''_1.\langle v\rangle|_{E_1}.\alpha''_1 \in Tr_1$, $\alpha''_1|_{V_1} = \alpha'_1|_{V_1}$, $\alpha''_1|_{C_1} = \langle\rangle$, and $\delta''_1|_{E_2} = \delta''_2|_{E_1}$, we make a case distinction on $v \in E_1$: Firstly, assume that $v \notin E_1$ holds. By an inductive argument on the length of $(\langle c\rangle.\delta''_2)|_{E_1}$, we obtain from $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ (insert $(\langle c\rangle.\delta''_2)|_{E_1}$ into $\beta|_{E_1}.\alpha'_1$ after $\beta|_{E_1}$ from left to right; recall that ES_1 is total in $C_1 \cap \Upsilon_1$) that there is a trace $\alpha''_1 \in E_1^*$ such that $\beta|_{E_1}.\langle c\rangle|_{E_1}.\delta''_2|_{E_1}.\alpha''_1 \in Tr_1$, $\alpha''_1|_{V_1} = \alpha'_1|_{V_1}$, and $\alpha''_1|_{C_1} = \langle\rangle$. Hence, $\beta|_{E_1}.\langle c\rangle|_{E_1}.\delta''_1.\langle v\rangle|_{E_1}.\alpha''_1 \in Tr_1$,

$\delta_1'' \in ((N_1 \cap \Delta_1) \cup (C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2))^*$, $\delta_1''|_{E_2} = \delta_2''|_{E_1}$ hold for $\delta_1'' = \delta_2''|_{E_1}$. Secondly, assume that $v \in E_1$ holds. Since $v \in E_1$, we have $v \in V_1 \cap \nabla_1$. By an inductive argument on the length of $(\langle c \rangle . \delta_2'')|_{E_1}$, we obtain from $FCI_{\mathcal{V}_1}^{\Gamma_1}(Tr_1)$ (insert $(\langle c \rangle . \delta_2'')|_{E_1}$ into $\beta|_{E_1} . \langle v \rangle|_{E_1} . \alpha_1'$ after $\beta|_{E_1}$ and before v from left to right) that there are traces $\alpha_1'' \in E_1^*$ and $\delta_1'' \in ((N_1 \cap \Delta_1) \cup (C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2))^*$ such that $\beta|_{E_1} . \langle c \rangle|_{E_1} . \delta_1'' . \langle v \rangle|_{E_1} . \alpha_1'' \in Tr_1$, $\alpha_1''|_{V_1} = \alpha_1'|_{V_1}$, $\alpha_1''|_{C_1} = \langle\rangle$, and $\delta_1''|_{C_1 \cap \Upsilon_1} = \delta_2''|_{E_1}$ hold. $\delta_1''|_{E_2} = \delta_2''|_{E_1}$ follows from $\delta_1''|_{C_1 \cap \Upsilon_1} = \delta_2''|_{E_1}$, $\delta_1'' \in ((N_1 \cap \Delta_1) \cup (C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2))^*$, and $N_1 \cap \Delta_1 \cap E_2 = \emptyset$.

Summarizing, there are $\alpha_1'' \in E_1^*$, $\alpha_2'' \in E_2^*$, $\delta_1'' \in ((N_1 \cap \Delta_1) \cup (C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2))^*$, $\delta_2'' \in (N_2 \cap \Delta_2)^*$ such that $\beta|_{E_1} . \langle c \rangle|_{E_1} . \delta_1'' . \langle v \rangle|_{E_1} . \alpha_1'' \in Tr_1$, $\beta|_{E_2} . \langle c \rangle|_{E_2} . \delta_2'' . \langle v \rangle|_{E_2} . \alpha_2'' \in Tr_2$, $\alpha_1''|_{V_1} = \alpha|_{V_1}$, $\alpha_1''|_{C_1} = \langle\rangle$, $\alpha_2''|_{V_2} = \alpha|_{V_2}$, $\alpha_2''|_{C_2} = \langle\rangle$, and $\delta_1''|_{E_2} = \delta_2''|_{E_1}$ hold. Since $\delta_1''|_{E_2} = \delta_2''|_{E_1}$ and $((N_1 \cap \Delta_1) \cup (N_2 \cap \Delta_2)) \subseteq (N \cap \Delta)$, there is a trace $\delta' \in (N \cap \Delta)^*$ such that $\delta'|_{E_1} = \delta_1''$ and $\delta'|_{E_2} = \delta_2''$. $\beta . \langle c \rangle . \delta' . \langle v \rangle \in Tr$ follows from $\beta|_{E_1} . \langle c \rangle|_{E_1} . \delta_1'' . \langle v \rangle|_{E_1} . \alpha_1'' \in Tr_1$, $\beta|_{E_2} . \langle c \rangle|_{E_2} . \delta_2'' . \langle v \rangle|_{E_2} . \alpha_2'' \in Tr_2$, and the definition of Tr (constructed by composition). From Lemma 6.4.4, we obtain for $\tau = \beta . \langle c \rangle . \delta' . \langle v \rangle$, $\lambda = \alpha|_V$, $t_1 = \alpha_1''$, and $t_2 = \alpha_2''$ that there is a trace $t \in E^*$ such that $\beta . \langle c \rangle . \delta' . \langle v \rangle . t \in Tr$, $t|_V = \alpha|_V$, and $t|_C = \langle\rangle$ hold. Thus, $FCI_{\mathcal{V}}^{\Gamma}(Tr)$ holds.

Proposition 6: Let $\alpha, \beta \in E^*$, $c \in C \cap \Upsilon$, and $v \in V \cap \nabla$ be arbitrary with $\beta . \langle v \rangle . \alpha \in Tr$, $\alpha|_C = \langle\rangle$, and $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, c)$. We have $(\beta . \langle v \rangle . \alpha)|_{E_1} \in Tr_1$ and $(\beta . \langle v \rangle . \alpha)|_{E_2} \in Tr_2$. Like in the proof of proposition 2, we obtain from $BSD_{\mathcal{V}_1}(Tr_1)$ and $BSD_{\mathcal{V}_2}(Tr_2)$ that there are traces $\alpha_1' \in E_1^*$, $\alpha_2' \in E_2^*$ such that $(\beta . \langle v \rangle)|_{E_1} . \alpha_1' \in Tr_1$, $\alpha_1'|_{V_1} = \alpha|_{V_1}$, $\alpha_1'|_{C_1} = \langle\rangle$, $(\beta . \langle v \rangle)|_{E_2} . \alpha_2' \in Tr_2$, $\alpha_2'|_{V_2} = \alpha|_{V_2}$, and $\alpha_2'|_{C_2} = \langle\rangle$ hold. That $Adm_{\mathcal{V}}^{\rho}(Tr, \beta, c)$ implies $Adm_{\mathcal{V}_1}^{\rho_1}(Tr_1, \beta|_{E_1}, c)$ and $Adm_{\mathcal{V}_2}^{\rho_2}(Tr_2, \beta|_{E_2}, c)$ can be shown like in the proof of proposition 3.

The remainder of this proof is along the same lines like the proof of proposition 5. The only differences are that $BSIA_{\mathcal{V}_2}^{\rho_2}(Tr_2)$ is applicable (for inserting c into $(\beta . \langle v \rangle)|_{E_2} . \alpha_2'$) because $Adm_{\mathcal{V}_2}^{\rho_2}(Tr_2, \beta|_{E_2}, c)$ holds rather than because $total(ES_2, C_2 \cap \Upsilon_2)$, that $FCIA_{\mathcal{V}_2}^{\rho_2, \Gamma_2}(Tr_2)$ is applied to insert c into $(\beta . \langle v \rangle)|_{E_2} . \alpha_2'$ (applicable because $Adm_{\mathcal{V}_2}^{\rho_2}(Tr_2, \beta|_{E_2}, c)$ holds) rather than $FCI_{\mathcal{V}_2}^{\Gamma_2}(Tr_2)$, that $BSIA_{\mathcal{V}_1}^{\rho_1}(Tr_1)$ is applicable (for inserting $(\langle c \rangle . \delta_2'')|_{E_1}$ into $\beta|_{E_1} . \alpha_1'$) because $Adm_{\mathcal{V}_1}^{\rho_1}(Tr_1, \beta|_{E_1}, c)$ and $total(ES_1, C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2)$ (rather than because ES_1 is total in $C_1 \cap \Upsilon_1$), and that $FCIA_{\mathcal{V}_1}^{\rho_1, \Gamma_1}(Tr_1)$ is applied to insert $(\langle c \rangle . \delta_2'')|_{E_1}$ into $\beta|_{E_1} . \langle v \rangle|_{E_1} . \alpha_1'$ (applicable because $Adm_{\mathcal{V}_1}^{\rho_1}(Tr_1, \beta|_{E_1}, c)$ and $total(ES_1, C_1 \cap \Upsilon_1 \cap N_2 \cap \Delta_2)$ hold) rather than $FCI_{\mathcal{V}_1}^{\Gamma_1}(Tr_1)$. □

Proof (of Theorem 6.4.2). Let $\tau' \in Tr$ be arbitrary. We have $\tau'|_{E_1} \in Tr_1$ and $\tau'|_{E_2} \in Tr_2$. From $R_{\mathcal{V}_1}(Tr_1)$ and $R_{\mathcal{V}_2}(Tr_2)$, we obtain that there are traces $\tau_1' \in Tr_1$ and $\tau_2' \in Tr_2$ with $\tau_1'|_V = \tau'|_{E_1 \cap V}$, $\tau_1'|_{C_1} = \langle\rangle$, $\tau_2'|_V = \tau'|_{E_2 \cap V}$, $\tau_2'|_{C_2} = \langle\rangle$. Lemma 6.4.4 yields for $\tau = \langle\rangle$, $\lambda = \tau'|_V$, $t_1 = \tau_1'$, and $t_2 = \tau_2'$ that there is a trace $t \in E^*$ with $t \in Tr$, $t|_V = \tau'|_V$, and $t|_C = \langle\rangle$. Thus, $R_{\mathcal{V}}(Tr)$ holds. □

Appendix E

Further Details of the Case Study

E.1 PP-Statements

Pre/postcondition-statements (abbreviated by *PP-statements*) can be used to specify the transition relation of a state-event system. PP-statements presume a specific notion of state, i.e. that *a state is a mapping from state variables to values.* Given a set E of events and a set $VAR = \{var_1, \ldots, var_n\}$ of n distinct variables, a PP-statement `Stat` restricts the possible transitions for some event $e_i \in E$ based on the notation

$$\begin{array}{lcl} e_i \texttt{ affects } var_{j_1}, \ldots, var_{j_k} & & \\ \texttt{Pre} & : & P(var_1, \ldots, var_n) \\ \texttt{Post} & : & Q(var_1, \ldots, var_n, var'_1, \ldots, var'_n) \end{array} \tag{E.1}$$

where $1 \leq j_l \leq n$ $(1 \leq l \leq k)$. The `affects` slot specifies that an occurrence of e_i may only affect the value of state variables in $var_{j_1}, \ldots, var_{j_k}$. The values of all other state variables remain unchanged when e_i occurs. The precondition slot of the PP-statement specifies that e_i is only *enabled* in states for which the condition P is satisfied. P must be a first-order logic formula that contains no primed state variables and the postcondition Q must be a first-order logic formula that may contain primed as well as unprimed state variables. Using primed variables other than $var_{j_1}, \ldots, var_{j_k}$ in Q is not ruled out but should be avoided for reasons to be explained later.

The semantics of PP-statements is given by a translation into higher-order formulas. The above PP-statement `Stat` translates into the following formula where the free variables s, e, s' are implicitly universally quantified:

$$\begin{array}{rcl} (s, e, s') \in T_{\texttt{Stat}} & \Leftrightarrow & [e{=}e_i \Rightarrow (\forall var \notin \{var_{j_1}, \ldots, var_{j_k}\}.\, s(var) = s'(var) \\ & & \quad \wedge P(s(var_1), \ldots, s(var_n)) \\ & & \quad \wedge Q(s(var_1), \ldots, s(var_n), s'(var_1), \ldots, s'(var_n))] \end{array}$$

Hence, each PP-statement `Stat` in a specification specifies a transition relation $T_{\texttt{Stat}} \subseteq S{\times}E{\times}S$. According to the above formula, a transition (s, e_i, s') complies with $T_{\texttt{Stat}}$ if and only if all frame axioms hold for s, s' (values of variables not in $var_{j_1}, \ldots, var_{j_k}$ remain unchanged), the precondition P holds for s, and the postcondition Q holds for s, s'. For all events besides e_i, $T_{\texttt{Stat}}$ permits arbitrary transitions, i.e. $\forall e \in E. \forall s, s' \in S. (e \neq e_i \Rightarrow (s, e, s') \in T_{\texttt{Stat}})$. Note that using primed state variables other than $var_{j_1}, \ldots, var_{j_k}$ in the postcondition Q may

lead to a contradiction with the frame axioms. This is the reason why all primed variables that occur in Q should be listed in the affects slot.

The *pre/postcondition-specification* (abbreviated by *PP-specification* in the following) of a transition relation consists of a set `Spec` of PP-statements. In practice, a PP-specification usually contains exactly one PP-statement for every event in E.[1] The semantics of a PP-specification `Spec` is given by the following translation into a higher-order formula:

$$(s, e, s') \in T_{\texttt{Spec}} \quad \Leftrightarrow \quad \bigwedge_{\texttt{Stat} \in \texttt{Spec}} (s, e, s') \in T_{\texttt{Stat}}$$

This means, a transition (s, e, s') complies with the transition relation $T_{\texttt{Spec}}$ specified by a PP-specification `Spec` if and only if it complies with each transition relation $T_{\texttt{Stat}}$ specified by some PP-statement `Stat` in `Spec`.

Remark E.1.1. For notational convenience, we permit the use of place holders in PP-statements. A parametric PP-statement for a list $x_1, \ldots, x_m$ of (typed) place holders has the following form:

$$\begin{array}{l} e_i(x_1, \ldots, x_m) \texttt{ affects } var_{j_1}, \ldots, var_{j_k} \\ \texttt{Pre} \quad : \quad P_{x_1,\ldots,x_m}(var_1, \ldots, var_n) \\ \texttt{Post} \quad : \quad Q_{x_1,\ldots,x_m}(var_1, \ldots, var_n, var'_1, \ldots, var'_n) \ . \end{array}$$

As usual, parametric PP-statements denote the set of all grounded PP-statements that are (type correct) instantiations of these parametric statements. ◇

Example E.1.2. In Fig. E.1, it is illustrated how to specify a transition relation with parametric PP-statements. The state-event system denoted by this specification models a random

S	$=$	$\{(\texttt{state}) \mapsto (s) \mid s \in \{r, t\}\}$, *running/terminated*
$s_0(r)$	$=$	$(\texttt{state}) \mapsto (r)$
E	$=$	$\{out(n) \mid n \in \mathbb{N}\} \cup \{term\}$
I	$=$	$\emptyset$
O	$=$	$\{out(n) \mid n \in \mathbb{N}\}$
T	$\subseteq$	$S \times E \times S$
term affects state		
Pre	:	state $= r$
Post	:	state $= t$
$out(n)$ affects —		
Pre	:	state $= r$
Post	:	tt

Figure E.1: Specification of the random generator by preconditions and postconditions

generator that can output any sequence of natural numbers (events $out(n)$) before it terminates (event *term*). Possible traces of this system have the form $\langle out(n_1) \ldots out(n_m) \rangle . \langle term \rangle$

[1] If there are multiple PP-statement for some event $e \in E$ then these PP-statements can be merged by (1) conjoining the preconditions by conjunction, (2) conjoining the postconditions by conjunction, and (3) intersecting the lists in the affects-slots. In order to avoid chaotic behavior, a PP-specification should contain at least one PP-statement for each event $e \in E$ because, otherwise, this event would be always enabled and could lead to arbitrary successor states.

or $\langle out(n_1) \ldots out(n_m) \rangle$ (respectively models that the sequence $n_1 \ldots . n_m$ has been output and that the system has or has not terminated). The dash in the `affects` slot of the PP-statement for *out* indicates the empty list of state variables, i.e. occurrences of *out* do not affect any state variables. ◇

E.2 Examples for the Strong Security Condition

The following four examples illustrate the strong security condition (cf. Definition 7.2.9)

Example E.2.1 (Implicit flow). The program if $h = 1$ then $l := 1$ else $l := 0$ is *not* strongly secure. Choose $mem_1 = (0,0)$ and $mem_2 = (1,0)$. If the computation starts with mem_1 then, after two computation steps, one arrives at the memory $(0,0)$ because $\langle$if $h = 1$ then $l := 1$ else $l := 0, (0,0), \sigma\rangle \rightarrow \langle l := 0, (0,0), \sigma\rangle$ and $\langle l := 0, (0,0), \sigma\rangle \rightarrow \langle\langle\rangle, (0,0), \sigma\rangle$ hold. If computation starts with mem_2 then, after two computation steps, one arrives at the memory $(1,1)$ because $\langle$if $h = 1$ then $l := 1$ else $l := 0, (1,0), \sigma\rangle \rightarrow \langle l := 1, (1,0), \sigma\rangle$ and $\langle l := 1, (1,0), \sigma\rangle \rightarrow \langle\langle\rangle, (1,1), \sigma\rangle$ hold. This shows that executing the given program with two low-equal memories may lead to memories that are not low equal. ◇

Example E.2.2 (Termination leak). The program $l := 0;$(while $h \neq 1$ do skip)$; l := 1$ is *not* strongly secure. Choose $mem_1 = (0,0)$ and $mem_2 = (1,0)$. If the computation starts with mem_1 then, after three computation steps (assignment of 0 to l, test of condition $h \neq 1$, and execution of skip), one arrives at the memory $(0,0)$. If the computation starts with mem_2 then, after three computation steps (assignment of 0 to l, test of condition $h \neq 1$, and assignment of 1 to l), one arrives at the memory $(1,1)$, which is not low equal to $(0,0)$. ◇

Example E.2.3 (Timing leak). The program fork$((l := 1)$ ((if $h = 1$ then (while $h <$ 10000 do $h := h + 1)); l := 0)$) is *not* strongly secure. Choose $mem_1 = (0,0)$ and $mem_2 = (1,0)$. Assume a scheduler that re-schedules immediately after executing a fork-command and that re-schedules after two computation steps if no fork-command has been executed. If the computation starts with mem_1 then, after three computation steps (execution of fork-command, test of condition $h = 1$, assignment of 0 to l), one arrives at the configuration $\langle l := 1, (0,0), \sigma\rangle$. If the computation starts with mem_2 then, after three computation steps (execution of fork-command, test of condition $h = 1$, test of condition $h < 10000$), one arrives at the configuration $\langle (l := 1), \quad (h := h + 1;$(while $h < 10000$ do $h := h + 1); l := 0), (1,0), \sigma\rangle$. After another computation step (assignment of 1 to l in both cases), the memories in both computations are still low equal. However, the first computation has already terminated (with memory $(0,1)$) while the second has not terminated. After the second computation has terminated, the resulting memory is $(1,0)$. This shows that executing the given program with two low-equal memories may lead to differences in the timing behavior and also to memories that are not low equal. ◇

Example E.2.4 (Communication leak). The program if $h = 1$ then send(cid_l, val) else skip is *not* strongly secure (assuming $level(cid_l) = low$). If the computation starts with the memory $(0,0)$ then the contents of the low channel cid_l does not change after two computation steps but if the computation starts with the memory $(1,0)$ then a value is sent on cid_l. This means, executing the program with two low-equal memories (and identical channel-status functions) may result in channel-status functions that are not low equal. ◇

E.3 Adequateness of the System Specification

In this section, we verify that our specification of DMWL process pools (cf. Definition 7.3.8) provides an adequate model of DMWL programs and their behavior. In the following, we show that every behavior that is possible for the specified state-event system also complies with the operational semantics of DMWL and that every behavior that complies with these semantics is possible for the specified state-event system.

Notational Conventions. Let $DMWLProcPool((p_1, initthread_1), \ldots, (p_n, initthread_n))$ be a DMWL process pool that we abbreviate by $SES = (S, s_0, E, I, O, T)$. Let $P = \{p_1, \ldots, p_n\}$. For all $p \in P$, let $initthread^p \in THREAD$ and define $SES^p = (S^p, s_0^p, E^p, I^p, O^p, T^p)$ by $SES^p = DMWLProcess(p, initthread^p)$. Note that $SES = \|_{p \in P} SES^p$ holds. We use $outbuf(cid)$ to denote the function that returns $\langle\rangle$ in case $outbuf$ is empty or contains an entry with an identifier different from cid and that returns the second element of $outbuf$ otherwise, i.e., $outbuf(cid) = \langle val \rangle$ if $outbuf = (cid, val)$. Recall also from Section 7.3.2 that no process can send to itself, i.e., $sender(cid) \neq receiver(cid)$ holds for all $cid \in CID$ and that we expect all processes in a process pool to comply with *sender* and *receiver*.

E.3.1 Basic Notions

In order to relate the transition relation of DMWL processes to the semantics of DMWL programs, it is necessary to construct a translation from one syntax to the other. For this purpose, we define two functions *channel* and *config* that extract the channel status function and the configuration of a single process, respectively, from a state of the DMWL process pool.

Definition E.3.1. Let $p_i, p_r \in PID$ with $p_i = sender(cid)$ and $p_r = receiver(cid)$.
The function $channel : S \to (CID \to VAL^*)$ is defined by:

$$channel(s) : cid \mapsto \begin{cases} outbuf_s^{p_i}(cid).\, inbuf_s^{p_r}(cid).\, pending_s^{p_r}(cid) & , \text{if } p_i, p_r \in P \\ outbuf_s^{p_i}(cid) & , \text{if } p_i \in P,\ p_r \notin P \\ inbuf_s^{p_r}(cid).\, pending_s^{p_r}(cid) & , \text{if } p_i \notin P,\ p_r \in P \\ \langle\rangle & , \text{if } p_i \notin P,\ p_r \notin P \end{cases}$$

◇

That is, $channel(s)(cid)$ denotes the sequence of messages that are on channel cid in state $s \in S$.

Definition E.3.2. $config : S \to PID \to [(\vec{CMD} \times (VAR \to VAL)) \cup \{\bot\}]$ is defined by:

$$config(s)(p) = \begin{cases} \bot & , \text{if } p \notin P \\ (cseq(thread_s), mem_s), & \text{if } p \in P \text{ and } s|_p = (mem_s, thread_s, \ldots) \end{cases}$$

where $s|_p$ extracts the local state of process p from the global state s. ◇

That is, $config(s)(p)$ denotes the pair consisting of the command vector and the memory in the local configuration of process p. The auxiliary function *cseq* translates a function $thread : TID \to (CMD \cup \{\bot, \top, \langle\rangle\})$ into a corresponding vector of DMWL commands.

Definition E.3.3. The function $cseq : (TID \to (CMD \cup \{\bot, \top, \langle\rangle\})) \to \vec{CMD}$ returns a vector of DMWL commands for each function $thread : TID \to (CMD \cup \{\bot, \top, \langle\rangle\})$. It is defined by

$$cseq(thread) = cseq_{aux}(\langle 0\rangle, thread) \qquad \diamond$$

Definition E.3.4. $cseq_{aux} : TID \to (TID \to (CMD \cup \{\bot, \top, \langle\rangle\})) \to \vec{CMD}$ is defined by

$$cseq_{aux}(tid, thread) = \langle\rangle \text{ , if } thread(tid) \in \{\bot, \langle\rangle\}$$

$$cseq_{aux}(tid, thread) = \langle thread(tid)\rangle \text{ , if } thread(tid) \in CMD$$

$$cseq_{aux}(tid, thread) = cseq_{aux}(tid.\langle 0\rangle, thread) \ldots cseq_{aux}(tid.\langle n\rangle, thread),$$
$$\text{if } thread(tid) = \top,\ n \in \mathbb{N} \text{ is chosen maximal such that } thread(tid.\langle n\rangle) \neq \bot \qquad \diamond$$

For a thread with identifier *tid* that has already terminated (or has never existed), $cseq_{aux}$ returns the empty command vector. However, if the thread is still running and has not spawned any child threads so far then $cseq_{aux}$ returns the command of that thread. Finally, if the thread has spawned child threads during its execution then the result of $cseq_{aux}$ is determined by a recursive application of $cseq_{aux}$ to the continuing parent thread (identifier $tid.\langle 0\rangle$) and to all child threads (identifier $tid.\langle i\rangle$ with $i > 0$). In the latter case, it is exploited that thread identifiers are chosen incrementally by *fork*-events and that $thread(tid) = \langle\rangle$ holds after termination of a thread with identifier *tid* (rather than $thread(tid) = \bot$). Summarizing, $cseq_{aux}(tid, thread)$ denotes the vector consisting of the command of the thread with identifier *tid* and the commands of all threads spawned by this thread and its children. Finally, $cseq(thread)$ denotes the command vector for all threads that resulted from the initial thread.

E.3.2 Adequateness Theorems

We are now ready to argue for the adequateness of our specification by the following four theorems. The somewhat involved proofs of these theorems are contained in Section E.3.3.

Theorem E.3.5. Let s, s' be reachable states for the composed *SES*, $p \in P$, and $e \in E^p_{local}$ such that $s \xrightarrow{e}_T s'$ holds. Assume that if $e = ite\text{-}rcv^{ff}_p(cid, var, val, C_1, C_2)$ then $channel(s)(cid) = \langle\rangle$.

1. If $e \neq fork_p(C, D_1 \ldots D_n)$ then
   $\langle\!| thread^p_s(atid^p_s), mem^p_s, channel(s) |\!\rangle \rightarrowtriangle \langle\!| thread^p_{s'}(atid^p_{s'}), mem^p_{s'}, channel(s') |\!\rangle$.
2. If $e = fork_p(C, D_1 \ldots D_n)$ then
   $\langle\!| thread^p_s(atid^p_s), mem^p_s, channel(s) |\!\rangle$
   $\rightarrowtriangle \langle\!| thread^p_{s'}(atid^p_s.\langle 0\rangle) \ldots thread^p_{s'}(atid^p_s.\langle n\rangle), mem^p_{s'}, channel(s') |\!\rangle$. $\diamond$

Theorem E.3.6. Let s, s' be reachable states for the composed *SES* and $\gamma \in E^*$. If $s \stackrel{\gamma}{\Longrightarrow} s'$ and γ contains no *setvar*-events or *start*-events and for all *trans*-events on a channel *cid* both, $receiver(cid)$ as well as $sender(cid)$ are in P then

$$\lhd config(s, p_1), \ldots, config(s, p_n); channel(s) \rhd$$
$$\twoheadrightarrow^* \lhd config(s', p_1), \ldots, config(s', p_n); channel(s') \rhd \ . \qquad \diamond$$

Theorem E.3.5 shows that the occurrence of a local event in E^p_{local} (modeling the execution of a DMWL command), indeed, corresponds to the behavior of the respective command in the sense that it results in a state with the same command vector/memory and the same channel

status like the configuration reached by the command. Theorem E.3.6 extends this result to traces. It shows that, for every behavior that is possible for the state-event system, the semantics of DMWL allow for an equivalent behavior of the corresponding DMWL program. Note that this theorem holds for arbitrary reachable starting states (rather than only for s_0). In the theorem, *setvar*-events and *start*-events are excluded because they do not correspond to any DMWL-commands. For DMWL programs, the setting of initial values is implicitly assumed to occur before program execution. In the event-based specification, this is made explicit by the occurrence of *setvar*- and *start*-events. Also excluded are *trans*-events on open channels, i.e. channels for which at most one of $sender(cid)$ or $receiver(cid)$ is in P. The reason for this restriction is that, firstly , if $sender(cid) \notin P$ then every event $trans(cid, val)$ would be always enabled (messages can be supplied by the environment at any point of time on such channels) and, secondly, if $receiver(cid) \notin P$ then an event $trans(cid, val)$ deletes a message from a channel without moving it into the input buffer of some process (messages from such channels have to be consumed by the environment). In contrast to this, the semantics of DMWL do not permit changes to the channel status function by the environment and message transmission on channels is instantaneously in the semantics. Consequently, adequateness can only be established for closed systems, i.e. systems without open channels, or for open systems where the open channels are not used.

Theorem E.3.7. Let $C, C' \in CMD$, $D_1 \ldots D_k \in \vec{CMD}$, $mem, mem' : VAR \to VAL$, and $\sigma, \sigma' : CID \to VAL^*$. Let $p \in P$ and $s \in S$ be a reachable state for the composed *SES* with $initialized_s^p = tt$, $executed_s^p = ff$, $atid_s^p \neq \bot$, $thread_s^p(atid_s^p) = C$, $mem_s^p = mem$, and $channel(s) = \sigma$. If $first(thread_s^p(atid_s^p)) \in \{\mathsf{receive}_p(cid', \ldots), \mathsf{if\text{-}receive}_p(cid', \ldots) \mid cid' \in CID\}$ and $pending_s^p(cid') = \langle\rangle$ then $channel(s)(cid') = \langle\rangle$ shall hold.

1. If $\langle\!| C, mem, \sigma |\!\rangle \twoheadrightarrow \langle\!| C', mem', \sigma' |\!\rangle$ then there are $e \in E^p_{local}$ and $s' \in S$ such that $s \xrightarrow{e}_T s'$, $thread_{s'}^p(atid_{s'}^p) = C'$, $mem_{s'}^p = mem'$, and $channel(s') = \sigma'$ hold.

2. If $\langle\!| C, mem, \sigma |\!\rangle \twoheadrightarrow \langle\!| C' D_1 \ldots D_k, mem', \sigma' |\!\rangle$ (with $k \geq 1$) then there are $e \in E^p_{local}$ and $s' \in S$ such that $s \xrightarrow{e}_T s'$, $mem_{s'}^p = mem'$, and $channel(s') = \sigma'$ hold. Moreover, $thread_{s'}^p(atid_s^p) = \top$, $thread_{s'}^p(atid_s^p.\langle 0 \rangle) = C'$, and $thread_{s'}^p(atid_s^p.\langle i \rangle) = D_i$ hold for all $i \in \{1, \ldots, k\}$. ◇

Theorem E.3.8. Let $(\vec{C_1}, mem_1) \ldots (\vec{C_n}, mem_n)$ and $(\vec{C'_1}, mem'_1) \ldots (\vec{C'_n}, mem'_n)$ be sequences of pairs, each consisting of a command vector and a memory such that none of the commands involves sending on an open channel. Let σ, σ' be channel status functions. Moreover, let $s \in S$ be a reachable state for the composed *SES* such that $channel(s) = \sigma$ and, for all $p_i \in P$, $initialized_s^{p_i} = tt$, $executed_s^{p_i} = ff$, $atid_s^{p_i} = \bot$, and $config(s)(p_i) = (\vec{C_i}, mem_i)$ hold.

If $\lhd(\vec{C_1}, mem_1), \ldots, (\vec{C_n}, mem_n); \sigma \rhd \twoheadrightarrow^* \lhd(\vec{C'_1}, mem'_1), \ldots, (\vec{C'_n}, mem'_n); \sigma' \rhd$ then there is a reachable state $s' \in S$ and a sequence $\gamma \in E^*$ that contains no *setvar*-events, *start*-events, or *trans*-events on open channels such that $s \xRightarrow{\gamma} s'$, $channel(s') = \sigma'$ and, for all $p_i \in P$, $config(s', p_i) = (\vec{C'_i}, mem'_i)$ hold. ◇

Theorem E.3.7 shows that executing a single DMWL command corresponds to the occurrence of a local event (modeling that command). That is, if a local configuration $\langle\!| C', mem', \sigma' |\!\rangle$ (or $\langle\!| C' D_1 \ldots D_k, mem', \sigma' |\!\rangle$) can be reached from a configuration $\langle\!| C, mem, \sigma |\!\rangle$ by a single $\twoheadrightarrow$-transition (execution of a single DMWL command) then for all states s corresponding to the configuration $\langle\!| C, mem, \sigma |\!\rangle$ (i.e. C is the active thread of process p in s, mem is the local memory of process p in s, and σ equals $channel(s)$) there is a local event e (modeling

the respective command) and a state s' (corresponding to the configuration $\langle C', mem', \sigma' \rangle$ or $\langle C'D_1 \ldots D_k, mem', \sigma' \rangle$) such that e is enabled in s and its occurrence results in the state s'. Theorem E.3.8 extends this result to the execution of distributed programs (i.e. $\twoheadrightarrow^*$-transitions). It shows that if the configuration $\lhd(\vec{C'}_{p_1}, mem'_{p_1}), \ldots, (\vec{C'}_{p_n}, mem'_{p_n}); \sigma' \rhd$ can be reached from $\lhd(\vec{C}_{p_1}, mem_{p_1}), \ldots, (\vec{C}_{p_n}, mem_{p_n}); \sigma \rhd$ then for every state $s \in S$ that corresponds to the second configuration there is a state $s' \in S$ that corresponds to the first configuration and there is a trace $\gamma \in E^*$ (without *setvar*-events, *start*-events, and *trans*-events on open channels) such that γ is enabled in state s and results in state s'.

E.3.3 Proofs of the Adequateness Theorems

In this section, we present the proofs of Theorems E.3.5, E.3.6, E.3.7, and E.3.8.

Theorems E.3.5 and E.3.6 ensure that every trace of a DMWL process pool models a behavior that complies with the semantics of DMWL.

Proof (of Theorem E.3.5). In the proof, we are only concerned with values of state objects and events for the single process p. For better readability, we omit the index p from state objects and events.

We make a case distinction on the event e.

skip The PP-statement for *skip* implies $first(thread_s(atid_s)) = \mathsf{skip}$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = rest(thread_s(atid_s))$, and $channel(s') = channel(s)$. Rules [Skip] and [Seq$_1$] of the semantics (cf. Figure 7.2) ensure that $\langle thread_s(atid_s), mem_s, channel(s) \rangle \twoheadrightarrow \langle thread_{s'}(atid_{s'}), mem_{s'}, channel(s') \rangle$ holds.

assign(var,val) We have $first(thread_s(atid_s)) = var := Exp$ for Exp with $Exp \downarrow^{mem_s} val$, $mem_{s'} = mem_s[var \mapsto val]$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = rest(thread_s(atid_s))$, and $channel(s') = channel(s)$. [Assign] and [Seq$_1$] ensure $\langle thread_s(atid_s), mem_s, channel(s) \rangle \twoheadrightarrow \langle thread_{s'}(atid_{s'}), mem_{s'}, channel(s') \rangle$.

ite^{tt}(B, C_1, C_2) We have $first(thread_s(atid_s)) = \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, $B \downarrow^{mem_s} tt$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = C_1; rest(thread_s(atid_s))$, $channel(s') = channel(s)$. From the rules [If$_{tt}$] and [Seq$_1$] we obtain that $\langle thread_s(atid_s), mem_s, channel(s) \rangle \twoheadrightarrow \langle thread_{s'}(atid_{s'}), mem_{s'}, channel(s') \rangle$ holds.

ite^{ff}(B, C_1, C_2) We have $first(thread_s(atid_s)) = \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, $B \downarrow^{mem_s} ff$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = C_2; rest(thread_s(atid_s))$, $channel(s') = channel(s)$. From the rules [If$_{ff}$] and [Seq$_1$] we obtain that $\langle thread_s(atid_s), mem_s, channel(s) \rangle \twoheadrightarrow \langle thread_{s'}(atid_{s'}), mem_{s'}, channel(s') \rangle$ holds.

while^{tt}(B, C_1) We have $first(thread_s(atid_s)) = \mathsf{while}\ B\ \mathsf{do}\ C_1$, $B \downarrow^{mem_s} tt$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = C_1; \mathsf{while}\ B\ \mathsf{do}\ C_1; rest(thread_s(atid_s))$, and $channel(s') = channel(s)$. The rules [While$_{tt}$] and [Seq$_1$] ensure $\langle thread_s(atid_s), mem_s, channel(s) \rangle \twoheadrightarrow \langle thread_{s'}(atid_{s'}), mem_{s'}, channel(s') \rangle$.

while^{ff}(B, C_1) We have $first(thread_s(atid_s)) = \mathsf{while}\ B\ \mathsf{do}\ C_1$, $B \downarrow^{mem_s} ff$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = rest(thread_s(atid_s))$, $channel(s') = channel(s)$. [While$_{ff}$], [Seq$_1$] ensure $\langle thread_s(atid_s), mem_s, channel(s) \rangle \twoheadrightarrow \langle thread_{s'}(atid_{s'}), mem_{s'}, channel(s') \rangle$.

fork$(C, \vec{D})$ We have $first(thread_s(atid_s)) = \mathsf{fork}(C D_1 \ldots D_n)$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_s) = \top$, $thread_{s'}(atid_s.\langle 0\rangle) = C$; $rest(thread(atid_s))$, and $channel(s') = channel(s)$. For all $i \in \{1, \ldots, n\}$ holds $thread_{s'}(atid_s.\langle i\rangle) = D_i$. The rules [Fork] and [$\mathrm{Seq}_2$] ensure $\langle\!| thread_s(atid_s), mem_s, channel(s) |\!\rangle \twoheadrightarrow \langle\!| thread_{s'}(atid_{s'}.\langle 0\rangle) \ldots thread_{s'}(atid_{s'}.\langle n\rangle), mem_{s'}, channel(s') |\!\rangle$.

send(cid,val) We have $first(thread_s(atid_s)) = \mathsf{send}(cid, Exp)$ for some expression Exp with $Exp \downarrow^{mem_s} val$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = rest(thread_s(atid_s))$, and $outbuf_{s'} = (cid, val)$. Since the occurrence of e only affects the values of $thread_s(atid_s)$, $executed_s$, $ainfo_s$, $outbuf_s$ we have $channel(s') = channel(s)[cid \mapsto \langle val\rangle.channel(s)(cid)]$. From the rules [Send] and [Seq_1] we obtain that $\langle\!| thread_s(atid_s), mem_s, channel(s) |\!\rangle \twoheadrightarrow \langle\!| thread_{s'}(atid_{s'}), mem_{s'}, channel(s') |\!\rangle$ holds.

receive(cid,var,val) We have $first(thread_s(atid_s)) = \mathsf{receive}(cid, var)$, $pending_s(cid) = vals.\langle val\rangle$, $mem_{s'} = mem_s[var \mapsto val]$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = rest(thread_s(atid_s))$, and $pending_{s'}(cid) = vals$. Since the occurrence of e only affects the values of $thread_s(atid_s)$, $executed_s$, $ainfo_s$, $mem_s(var)$, and $pending_s(cid)$, we have $channel(s') = channel(s)[cid \mapsto butlast(channel(s)(cid))]$. [Receive] and [$\mathrm{Seq}_1$] ensure $\langle\!| thread_s(atid_s), mem_s, channel(s) |\!\rangle \twoheadrightarrow \langle\!| thread_{s'}(atid_{s'}), mem_{s'}, channel(s') |\!\rangle$.

ite-rcv^{tt}(cid, var, val, C_1, C_2) $first(thread_s(atid_s)) = \mathsf{if\text{-}receive}(cid, var, C_1, C_2)$, $pending_s(cid) = vals.\langle val\rangle$, $mem_{s'} = mem_s[var \mapsto val]$, $atid_{s'} = atid_s$, $thread_{s'}(atid_{s'}) = C_1; rest(thread_s(atid_s))$, and $pending_{s'}(cid) = vals$. The occurrence of e only affects the values of $thread_s(atid_s)$, $executed_s$, $ainfo_s$, $mem_s(var)$, $pending_s(cid)$, i.e. we have $channel(s') = channel(s)[cid \mapsto butlast(channel(s)(cid))]$. [$\mathrm{IfRcv}_{tt}$] and [$\mathrm{Seq}_1$] ensure $\langle\!| thread_s(atid_s), mem_s, channel(s) |\!\rangle \twoheadrightarrow \langle\!| thread_{s'}(atid_{s'}), mem_{s'}, channel(s') |\!\rangle$.

ite-rcv^{ff}(cid, var, val, C_1, C_2) $first(thread_s(atid_s)) = \mathsf{if\text{-}receive}(cid, var, C_1, C_2)$, $pending_s(cid) = \langle\rangle$, $mem_{s'} = mem_s$, $atid_{s'} = atid_s$, and $thread_{s'}(atid_{s'}) = C_2; rest(thread_s(atid_s))$. By assumption, we have $channel(s)(cid) = \langle\rangle$. Since e only affects $thread_s(atid_s)$, $executed_s$, and $ainfo_s$, we have $channel(s') = channel(s)$. The rules [IfRcv_{ff}] and [Seq_1] ensure $\langle\!| thread_s(atid_s), mem_s, channel(s) |\!\rangle \twoheadrightarrow \langle\!| thread_{s'}(atid_{s'}), mem_{s'}, channel(s') |\!\rangle$. □

Before proving Theorem E.3.6, we present a lemma that is helpful in this proof.

Lemma E.3.9. If s is a reachable state for the composed *SES* and $p \in P$ then

- $initialized_s^p = tt \wedge executed_s^p = ff \wedge atid_s^p \neq \bot \wedge thread_s^p(atid_s^p) \notin \{\bot, \top, \langle\rangle\}$ $\wedge outbuf_s^p = \langle\rangle$,
- $initialized_s^p = tt \wedge executed_s^p = tt \wedge atid_s^p \neq \bot \wedge outbuf_s^p \neq \langle\rangle$,
- $initialized_s^p = tt \wedge executed_s^p = tt \wedge atid_s^p \neq \bot \wedge outbuf_s^p = \langle\rangle$,
- $initialized_s^p = tt \wedge executed_s^p = ff \wedge atid_s^p = \bot \wedge outbuf_s^p = \langle\rangle$, or
- $initialized_s^p = ff \wedge executed_s^p = ff \wedge atid_s^p = \bot \wedge outbuf_s^p = \langle\rangle$ hold. ◇

Proof. In the proof, we are only concerned with values of state objects and events for the single process p. For better readability, we omit the index p from state objects and events.

Choose $\gamma \in E^*$ with $s_0 \stackrel{\gamma}{\Longrightarrow} s$. The proof is by induction on γ.

Base case ($\gamma = \langle\rangle$): $initialized_s = ff$, $executed_{s_0} = ff$, $atid_{s_0} = \bot$, and $outbuf_{s_0} = \langle\rangle$ follow from Figure 7.11.

Step case ($\gamma = \delta.\langle e \rangle$): There is a state $s_i \in S$ with $s_0 \stackrel{\delta}{\Longrightarrow} s_i$ and $s_i \stackrel{e}{\longrightarrow} s$. The induction hypothesis ensures that the proposition holds in s_i. If e is a *setvar*-, *outvar*, input *trans*-event, or $e \notin E^p$ then the proposition follows directly because these events do not affect the values of *initialized*, *executed*, *atid*, *thread*, and *outbuf* in process p. In the rest of the proof, we assume $e \in E^p$ and that e is no *setvar*-, *outvar*, or input *trans*-event. According to the induction hypothesis, we distinguish five cases.

- Assume $initialized_{s_i} = tt$, $executed_{s_i} = ff$, $atid_{s_i} \neq \bot$, $thread_{s_i}(atid_{s_i}) \notin \{\bot, \top, \langle\rangle\}$, and $outbuf_{s_i} = \langle\rangle$. This implies $e \in E^p_{local}$. The PP-statement for e ensures $initialized_s = tt$, $executed_s = tt$, and $atid_s \neq \bot$.

- Assume $initialized_{s_i} = tt$, $executed_{s_i} = tt$, $atid_{s_i} \neq \bot$, and $outbuf_{s_i} \neq \langle\rangle$. This implies $e = trans(cid, val)$ for some $cid \in CID$ and $val \in VAL$ with $sender(cid) = p$. The PP-statement ensures $initialized_s = tt$, $executed_s = tt$, $atid_s \neq \bot$, and $outbuf_s = \langle\rangle$.

- Assume $initialized_{s_i} = tt$, $executed_{s_i} = tt$, $atid_{s_i} \neq \bot$, and $outbuf_{s_i} = \langle\rangle$. This implies $e = yield(info)$ for some $info \in INFO$. The PP-statement ensures $initialized_s = tt$, $executed_s = ff$, $atid_s = \bot$, and $outbuf_s = \langle\rangle$.

- Assume $initialized_{s_i} = tt$, $executed_{s_i} = ff$, $atid_{s_i} = \bot$, and $outbuf_{s_i} = \langle\rangle$. This implies $e = schedule(tid)$ for some $tid \in TID$. The PP-statement ensures $initialized_s = tt$, $executed_s = ff$, $atid_s \neq \bot$, $thread_s(atid_s) \notin \{\bot, \top, \langle\rangle\}$, and $outbuf_s = \langle\rangle$.

- Assume $initialized_{s_i} = ff$, $executed_{s_i} = ff$, $atid_{s_i} = \bot$, and $outbuf_{s_i} = \langle\rangle$. This implies $e = start$. The PP-statement ensures $initialized_s = tt$, $executed_s = ff$, $atid_s = \bot$, and $outbuf_s = \langle\rangle$. □

According to Lemma E.3.9, it suffices to consider the five given cases when analyzing the possible behaviors of DMWL process pools.

We are now ready to prove Theorem E.3.6.

Proof (of Theorem E.3.6). First, we establish a few restrictions on γ. Observe that the following properties are satisfied for all $cid \in CID$, all $val, val' \in VAL$, all $p_r, p_i \in P$ with $p_r \neq p_i$, and all $schedule_{p_r}(\ldots)$, $ite\text{-}rcv^{ff}_{p_r}(cid, \ldots)$, $send_{p_i}(cid, val')$-events:

- If $\gamma = \gamma_1.\langle schedule_{p_r}(\ldots)\rangle.\gamma_2.\langle ite\text{-}rcv^{ff}_{p_r}(cid, \ldots)\rangle.\gamma_3$
  and $\gamma_2|_{E^{p_r}} \in \{trans(cid', val') \mid cid' \in CID, receiver(cid') = p_r, sender(cid') \neq p_r\}^*$
  then $s \stackrel{\gamma'}{\Longrightarrow} s'$ where $\gamma' = \gamma_1.\langle schedule_{p_r}(\ldots).ite\text{-}rcv^{ff}_{p_r}(cid, \ldots)\rangle.\gamma_2.\gamma_3$.

- If $\gamma = \gamma_1.\langle send_{p_i}(cid, val)\rangle.\gamma_2.\langle schedule_{p_r}(\ldots).ite\text{-}rcv^{ff}_{p_r}(cid, \ldots)\rangle.\gamma_3$
  and $\gamma_2|_{E^{p_i}} \in \{trans(cid', val') \mid cid \in CID, cid' \neq cid\}^*$
  then $s \stackrel{\gamma'}{\Longrightarrow} s'$ where
  $\gamma' = \gamma_1.\gamma_2.\langle schedule_{p_r}(\ldots).ite\text{-}rcv^{ff}_{p_r}(cid, \ldots).send_{p_i}(cid, val)\rangle.\gamma_3$.

Both of these properties can be proven by a simple induction over γ_2. According to the above properties we may make the following assumptions about γ. These assumptions will be helpful for applying Theorem E.3.5.

1. If $\gamma = \gamma_1.\langle schedule_{p_r}(\ldots)\rangle.\gamma_2.\langle ite\text{-}rcv^{ff}{}_{p_r}(cid,\ldots)\rangle.\gamma_3$
   and $\gamma_2|_{E^{p_r}} \in \{trans(cid', val') \mid cid' \in CID, receiver(cid') = p_r \wedge sender(cid') \neq p_r\}^*$
   then $\gamma_2 = \langle\rangle$.

2. If $\gamma = \gamma_1.\langle send_{p_i}(cid, val)\rangle.\gamma_2.\langle schedule_{p_r}(\ldots).ite\text{-}rcv^{ff}{}_{p_r}(cid,\ldots)\rangle.\gamma_3$
   then an event $trans(cid, val)$ occurs in γ_2. Together with the precondition of $ite\text{-}rcv^{ff}$, this implies that there must be a *receive-* or $ite\text{-}rcv^{tt}$-event in γ_2 that consumes the message *val* on *cid*.

Moreover, we assume that γ contains no *setvar-*, *start-*, or *outvar*-events. That γ contains neither *setvar-* nor *start*-events is an assumption of the theorem. Since *outvar*-events have no effect on the state they can safely be removed.

The proof of the theorem proceeds by induction on the length of γ.

Base case ($\gamma = \langle\rangle$): The proposition holds because $s' = s$ and $\twoheadrightarrow^*$ is reflexive.

Step case ($\gamma = \langle e\rangle.\delta$): There exists $s_i \in S$ with $s \xrightarrow{e} s_i$ and $s_i \stackrel{\delta}{\Longrightarrow} s'$. We make a case distinction depending on if e is associated with one or more DMWL processes (at most two).

Firstly, assume that there are $p_i, p_r \in P$ with $p_i \neq p_r$ and $e \in E^{p_i} \cap E^{p_r}$. Hence, $e = trans(cid, val)$ holds. Without loss of generality, assume $sender(cid) = p_i$ and $receiver(cid) = p_r$. We have $channel(s') = channel(s)$. The occurrence of e affects only *inbuf* and *outbuf* but no other state variables. The proposition follows from the induction hypothesis.

Secondly, assume that there is exactly one $p \in P$ with $e \in E^p$. For better readability, we omit the index p from state objects and events in the following. We make a case distinction according to Lemma E.3.9.

- Assume $initialized_s = tt$, $executed_s = ff$, $atid_s \neq \bot$, $thread_s(atid_s) \notin \{\bot, \top, \langle\rangle\}$, and $outbuf_s = \langle\rangle$. The proposition follows from $e \in E^p_{local}$, Theorem E.3.5, the frame axioms for events in E^p_{local}, Definition E.3.2, rule [Step], rule [Pick], and the induction hypothesis. Note that if $e = ite\text{-}rcv^{ff}(cid', \ldots)$ then $channel(s)(cid') = \langle\rangle$ holds because of our initial assumptions (1,2) on γ. Hence, the requirements of Theorem E.3.5 are, indeed, satisfied.

- The case $initialized_s = tt$, $executed_s = tt$, $atid_s \neq \bot$, and $outbuf_s \neq \langle\rangle$ leads to a contradiction. Since $e = trans(cid, val)$, $sender(cid), receiver(cid) \in P$, and a process cannot send to itself, this case contradicts our assumption that there is exactly one $p \in P$ with $e \in E^p$.

- Assume $initialized_s = tt$, $executed_s = tt$, $atid_s \neq \bot$, and $outbuf_s = \langle\rangle$. Then $e = yield(info)$ holds for $info = (ainfo_s, mem_s(l), blocked\text{-}set_s)$. The PP-statement for *yield* implies $config(s')(p) = config(s)(p)$ and $channel(s') = channel(s)$. The proposition follows from the induction hypothesis.

- Assume $initialized_s = tt$, $executed_s = ff$, $atid_s = \bot$, and $outbuf_s = \langle\rangle$. Then $e = schedule(tid)$ holds for some $tid \in TID$. The PP-statement for *schedule* implies that $config(s', p) = config(s)(p)$ and $channel(s') = channel(s)$ hold. The proposition follows from the induction hypothesis.

- Assume $initialized_s = ff$, $executed_s = ff$, $atid_s = \bot$, and $outbuf_s = \langle\rangle$. Since s is a reachable state, $initialized_s = ff$, and e is enabled in s, e must be a *outvar-*, *setvar-*, *start-*, or *trans*-event. According to our assumptions, e is neither a *outvar-* nor a *setvar-* nor a *start*-event. Since e belongs to a single process only, e cannot be a *trans*-event

(cf. our previous argumentation on *trans*-events). Hence, this case is impossible under the assumptions made. □

Theorems E.3.7 and E.3.8 ensure that for every behavior of a distributed DMWL program there is a trace of the corresponding DMWL process pool.

Proof (of Theorem E.3.7). Assume that $\mathcal{D}$ is a derivation of $\langle\!| C, mem, \sigma |\!\rangle \twoheadrightarrow \langle\!| C', mem', \sigma' |\!\rangle$. There must be exactly one application of one of the rules [Skip], [Assign], [If$_{tt}$], [If$_{ff}$], [While$_{tt}$], [While$_{ff}$], [Send], [Receive], [IfRcv$_{tt}$], or [IfRcv$_{ff}$] in $\mathcal{D}$.

We make a case distinction depending on which of these rules occurs in $\mathcal{D}$. In the proof, we are only concerned with values of state objects and events for the single process p. For better readability, we omit the index p from state objects and events.

[Skip] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{skip}$, $C' = \mathit{rest}(C)$, $mem' = mem$, and $\sigma' = \sigma$. For $e = \mathit{skip}$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = \mathit{rest}(C)$, $mem_{s'} = mem$, and $\mathit{channel}(s') = \sigma$.

[Assign] and [Seq$_1$] ensure $\mathit{first}(C) = \mathit{var} := \mathit{Exp}$ with $\mathit{Exp} \downarrow^{mem} \mathit{val}$ (for some $\mathit{var} \in \mathit{VAR}$, $\mathit{val} \in \mathit{VAL}$, $\mathit{Exp} \in \mathit{EXP}$), $C' = \mathit{rest}(C)$, $mem' = mem[\mathit{var} \mapsto \mathit{val}]$, and $\sigma' = \sigma$. For $e = \mathit{assign}(\mathit{var}, \mathit{val})$ there is $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = \mathit{rest}(C)$, $mem_{s'} = mem[\mathit{var} \mapsto \mathit{val}]$, and $\mathit{channel}(s') = \sigma$.

[If$_{tt}$] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ with $B \downarrow^{mem} tt$ (for $C_1, C_2 \in \mathit{CMD}$, $B \in \mathit{BOOL}$), $C' = C_1; \mathit{rest}(C)$, $mem' = mem$, and $\sigma' = \sigma$. For $e = \mathit{ite}^{tt}(B, C_1, C_2)$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = C_1; \mathit{rest}(C)$, $mem_{s'} = mem$, and $\mathit{channel}(s') = \sigma$.

[If$_{ff}$] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ with $B \downarrow^{mem} ff$ (for $C_1, C_2 \in \mathit{CMD}$, $B \in \mathit{BOOL}$), $C' = C_2; \mathit{rest}(C)$, $mem' = mem$, and $\sigma' = \sigma$. For $e = \mathit{ite}^{ff}(B, C_1, C_2)$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = C_2; \mathit{rest}(C)$, $mem_{s'} = mem$, and $\mathit{channel}(s') = \sigma$.

[While$_{tt}$] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{while}\ B\ \mathsf{do}\ C_1$ with $B \downarrow^{mem} tt$ (for $C_1 \in \mathit{CMD}$, $B \in \mathit{BOOL}$), $C' = C_1; \mathsf{while}\ B\ \mathsf{do}\ C_1; \mathit{rest}(C)$, $mem' = mem$, and $\sigma' = \sigma$. For $e = \mathit{while}^{tt}(B, C_1)$ there is $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = C_1; \mathsf{while}\ B\ \mathsf{do}\ C_1; \mathit{rest}(C)$, $mem_{s'} = mem$, and $\mathit{channel}(s') = \sigma$.

[While$_{ff}$] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{while}\ B\ \mathsf{do}\ C_1$ with $B \downarrow^{mem} ff$ (for $C_1 \in \mathit{CMD}$, $B \in \mathit{BOOL}$), $C' = \mathit{rest}(C)$, $mem' = mem$, and $\sigma' = \sigma$. For $e = \mathit{while}^{ff}(B, C_1)$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = \mathit{rest}(C)$, $mem_{s'} = mem$, and $\mathit{channel}(s') = \sigma$.

[Send] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{send}(\mathit{cid}, \mathit{Exp})$ with $\mathit{Exp} \downarrow^{mem} \mathit{val}$, $C' = \mathit{rest}(C)$, $mem' = mem$, $\sigma'(\mathit{cid}) = \langle \mathit{val} \rangle.\sigma(\mathit{cid})$, and $\sigma'(\mathit{cid}') = \sigma(\mathit{cid}')$ for all $\mathit{cid}' \neq \mathit{cid}$. For $e = \mathit{send}(\mathit{cid}, \mathit{val})$ there is $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = \mathit{rest}(C)$, $mem_{s'} = mem$, $\mathit{channel}(s')(\mathit{cid}) = \langle \mathit{val} \rangle.\sigma(\mathit{cid})$, and $\mathit{channel}(s')(\mathit{cid}') = \sigma(\mathit{cid}')$ for all $\mathit{cid}' \neq \mathit{cid}$.

[Receive] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{receive}(\mathit{cid}, \mathit{var})$, $\sigma(\mathit{cid}) = \mathit{vals}.\langle \mathit{val} \rangle$, $C' = \mathit{rest}(C)$, $mem' = mem[\mathit{var} \mapsto \mathit{val}]$, $\sigma'(\mathit{cid}) = \mathit{vals}$, and $\sigma'(\mathit{cid}') = \sigma(\mathit{cid}')$ hold for all $\mathit{cid}' \neq \mathit{cid}$. By assumption $\mathit{pending}_s(\mathit{cid}) = \mathit{vals}'.\langle \mathit{val} \rangle$ holds (where vals' is a suffix of vals). For $e = \mathit{receive}(\mathit{cid}, \mathit{var}, \mathit{val})$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = \mathit{rest}(C)$, $mem_{s'} = mem[\mathit{var} \mapsto \mathit{val}]$, $\mathit{channel}(s')(\mathit{cid}) = \mathit{vals}$, and $\mathit{channel}(s')(\mathit{cid}') = \sigma(\mathit{cid}')$ for all $\mathit{cid}' \neq \mathit{cid}$.

[IfRcv$_{tt}$] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{if\text{-}receive}(\mathit{cid}, \mathit{var}, C_1, C_2)$, $\sigma(\mathit{cid}) = \mathit{vals}.\langle \mathit{val}\rangle$, $C' = C_1; \mathit{rest}(C)$, $\mathit{mem}' = \mathit{mem}[\mathit{var} \mapsto \mathit{val}]$, $\sigma'(\mathit{cid}) = \mathit{vals}$, $\sigma'(\mathit{cid}') = \sigma(\mathit{cid}')$ for all $\mathit{cid}' \neq \mathit{cid}$. By assumption $\mathit{pending}_s(\mathit{cid}) = \mathit{vals}'.\langle \mathit{val}\rangle$(where vals' is a suffix of vals). For $e = \mathit{ite\text{-}rcv}^{tt}(\mathit{cid}, \mathit{var}, \mathit{val}, C_1, C_2)$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = C_1; \mathit{rest}(C)$, $\mathit{mem}_{s'} = \mathit{mem}[\mathit{var} \mapsto \mathit{val}]$, $\mathit{channel}(s')(\mathit{cid}) = \mathit{vals}$, and $\mathit{channel}(s')(\mathit{cid}') = \sigma(\mathit{cid}')$ for all $\mathit{cid}' \neq \mathit{cid}$.

[IfRcv$_{ff}$] and [Seq$_1$] ensure $\mathit{first}(C) = \mathsf{if\text{-}receive}(\mathit{cid}, \mathit{var}, C_1, C_2)$, $\sigma(\mathit{cid}) = \langle\rangle$, $C' = C_2; \mathit{rest}(C)$, $\mathit{mem}' = \mathit{mem}$, and $\sigma' = \sigma$. There is a state $s' \in S$ for $e = \mathit{ite\text{-}rcv}^{ff}(\mathit{cid}, \mathit{var}, \mathit{val}, C_1, C_2)$ with $s \xrightarrow{e} s'$, $\mathit{thread}_{s'}(\mathit{atid}_{s'}) = C_2; \mathit{rest}(C)$, $\mathit{mem}_{s'} = \mathit{mem}$, and $\mathit{channel}(s') = \sigma$.

Assume that $\mathcal{D}$ is a derivation of $\langle C, \mathit{mem}, \sigma\rangle \twoheadrightarrow \langle C' D_1 \ldots D_k, \mathit{mem}', \sigma'\rangle$. There must be exactly one application of the rule [Fork] in $\mathcal{D}$.

[Fork] and [Seq$_2$] ensure $\mathit{first}(C) = \mathsf{fork}(C'' D_1 \ldots D_k)$ (for some $C'' \in \mathit{CMD}$, $D_1 \ldots D_k \in \vec{\mathit{CMD}}$), $C' = C''; \mathit{rest}(C)$, $\mathit{mem}' = \mathit{mem}$, and $\sigma' = \sigma$. For $e = \mathit{fork}(C'', D_1 \ldots D_k)$ there is a state $s' \in S$ with $s \xrightarrow{e} s'$, $\mathit{mem}_{s'} = \mathit{mem}$, $\mathit{channel}(s') = \sigma$, $\mathit{thread}_{s'}(\mathit{atid}_s) = \top$, $\mathit{thread}_{s'}(\mathit{atid}_s.\langle 0\rangle) = C'$, and for all $i \in \{1, \ldots, k\}$ holds $\mathit{thread}_{s'}(\mathit{atid}_s.\langle i\rangle) = D_i$. □

Theorem E.3.7 ensures that a small step in the operational semantics corresponds to the occurrence of some local event in E^p_{local}.

Proof (of Theorem E.3.8). Let $\mathcal{D}$ be a derivation of

$$\triangleleft(\vec{C}_1, \mathit{mem}_1), \ldots, (\vec{C}_n, \mathit{mem}_n); \sigma \triangleright \twoheadrightarrow^* \triangleleft(\vec{C}'_1, \mathit{mem}'_1), \ldots, (\vec{C}'_n, \mathit{mem}'_n); \sigma' \triangleright \ .$$

The proof proceeds by induction on m, the number of applications of the rule [Step] in $\mathcal{D}$.

Base case: $\mathcal{D}$ contains no rule applications. Hence, $\sigma' = \sigma$ and $(\vec{C}'_i, \mathit{mem}'_i) = (\vec{C}_i, \mathit{mem}_i)$ hold for all $i \in \{1, \ldots n\}$. Consequently, the proposition holds for $\gamma = \langle\rangle$ and $s' = s$.

Step case ($m = m' + 1$): There is a global configuration

$$\triangleleft(\vec{C}''_1, \mathit{mem}''_1), \ldots, (\vec{C}''_n, \mathit{mem}''_n); \sigma'' \triangleright$$

such that

$$\triangleleft(\vec{C}_1, \mathit{mem}_1), \ldots, (\vec{C}_n, \mathit{mem}_n); \sigma \triangleright \twoheadrightarrow \triangleleft(\vec{C}''_1, \mathit{mem}''_1), \ldots, (\vec{C}''_n, \mathit{mem}''_n); \sigma'' \triangleright$$

and

$$\triangleleft(\vec{C}''_1, \mathit{mem}''_1), \ldots, (\vec{C}''_n, \mathit{mem}''_n); \sigma'' \triangleright \twoheadrightarrow^* \triangleleft(\vec{C}'_1, \mathit{mem}'_1), \ldots, (\vec{C}'_n, \mathit{mem}'_n); \sigma' \triangleright$$

hold. Let $\mathcal{D}'$ be a derivation of the former judgment and $\mathcal{D}''$ be a derivation of the latter judgment. Consequently, there are m' application of rule [Step] in $\mathcal{D}''$ and there is one application of this rule in $\mathcal{D}'$. If there are a state $s'' \in S$ and a sequence $\gamma'' \in E^*$ that contains no *setvar*-events, *start*-events, or *trans*-events on open channels such that $s \xrightarrow{\gamma''} s''$, $\mathit{channel}(s'') = \sigma''$, $\mathit{initialized}^{p_i}_{s''} = tt$, $\mathit{executed}^{p_i}_{s''} = ff$, $\mathit{atid}^{p_i}_{s''} = \bot$, and $\mathit{config}(s'')(p_i) = (\vec{C}''_i, \mathit{mem}''_i)$ hold for all $p_i \in P$ then the proposition follows from the induction hypothesis.

We now show that γ'' and s'' with these properties indeed exist.

[Step] is applied once in $\mathcal{D}''$. Let p_j be the identifier of the process that is selected in this application. [Pick] is applied once in $\mathcal{D}''$. Let tid be the identifier of the thread that is selected in this application. Event $e_1 = \mathit{schedule}_{p_j}(\mathit{tid})$ is enabled in s. For $s_1 \in S$ with $s \xrightarrow{e_1} s_1$, we

have $initialized_{s_1}^{p_j} = tt$, $executed_{s_1}^{p_j} = ff$, $atid_{s_1}^{p_j} \neq \bot$, $thread_{s_1}^{p_j} = thread_s^{p_j}$, $mem_{s_1}^{p_j} = mem_s^{p_j}$, and $channel(s_1) = channel(s)$. Moreover, $config(s_1)(p') = config(s)(p')$ holds for all $p' \in P$ with $p' \neq p_j$. From the definitions of the rules [Step], [Pick] and Theorem E.3.7, we obtain that there are an event $e_2 \in E_{local}^{p_j}$ and a state $s_2 \in S$ with $s_1 \xrightarrow{e_2} s_2$, $initialized_{s_2}^{p_j} = tt$, $executed_{s_2}^{p_j} = tt$, $atid_{s_2}^{p_j} \neq \bot$, $mem_{s_2}^{p_j} = mem_j''$, $cseq(thread_{s_2}^{p_j}) = \vec{C}_j''$, $channel(s_2) = \sigma''$, and $config(s_2)(p') = config(s_1)(p')$ holds for all $p' \in P$ with $p' \neq p_j$. The output buffer is non-empty if and only if e_2 is a *send*-event. If $outbuf_{s_2}^{p_j} = (cid, val)$ then let $\delta = \langle trans(cid, val) \rangle$ (recall that cid cannot be an open channel according to our assumption that none of the commands involves sending on open channels). If $outbuf_{s_2}^{p_j} = \langle\rangle$ then let $\delta = \langle\rangle$. In both cases, there is a state s_3 with $s_2 \stackrel{\delta}{\Longrightarrow} s_3$, $outbuf_{s_3}^{p_j} = \langle\rangle$, $initialized_{s_3}^{p_j} = tt$, $executed_{s_3}^{p_j} = tt$, $atid_{s_3}^{p_j} \neq \bot$, $mem_{s_3}^{p_j} = mem_j''$, $cseq(thread_{s_3}^{p_j}) = \vec{C}_j''$, $channel(s_3) = \sigma''$, and $config(s_3)(p') = config(s_1)(p')$ holds for all $p' \in P$ with $p' \neq p_j$. Event $e_3 = yield_{p_j}(ainfo_{s_3})$ is enabled in s_3. Hence, there is a state $s'' \in S$ with $s_2 \xrightarrow{e_3} s''$. Thus, for $\gamma'' = \langle e_1.e_2 \rangle.\delta.\langle e_3 \rangle$, we have $s \stackrel{\gamma''}{\Longrightarrow} s''$ and $channel(s'') = \sigma''$. Moreover, $initialized_{s''}^{p_i} = tt$, $executed_{s''}^{p_i} = ff$, $atid_{s''}^{p_i} = \bot$, and $config(s'', p_i) = (\vec{C}''_i, mem_i'')$ hold for all $p_i \in P$. □

E.4 Lemmas used in the Proof by Unwinding

The following lemma is used in the proof of Theorem 7.5.1 in Section 7.5.2.

Lemma E.4.1 (Satisfaction of unwinding conditions for single processes). Let $C \in CMD$, and $p \in PID$. Moreover, let $SES^p = (S^p, s_0^p, E^p, I^p, O^p, T^p)$ be defined by $SES^p = DMWLProcess(p, C)$. Moreover, let $\bowtie \subseteq S \times S$ be defined like in the proof of Theorem 7.5.1.

If C is strongly secure then $lrbe_{\mathcal{HI}^p}^{\rho^p}(T^p, \bowtie)$ and $osc_{\mathcal{HI}^p}(T^p, \bowtie)$ hold. ◇

Proof. We prove the two unwinding conditions:

lrbe$_{\mathcal{HI}^p}^{\rho^p}(T^p, \bowtie)$ Let $s \in S^p$ and $c \in C^p$. Assume $reachable(SES^p, s)$ and $En_{\mathcal{HI}^p}^{\rho^p}(T^p, s, c)$. According to the definition of C^p, either $c = setvar(h, val)$ or $c = trans(cid_h, val)$ holds (where $val \in VAL$, $cid_h \in CID$ with $receiver(cid_h) = p$ and $level(cid_h) = high$). We make a case distinction on these two possibilities.

Firstly, assume $c = setvar(h, val)$. Since *setvar*-events are only enabled if the initialization phase is finished, we conclude from $En_{\mathcal{HI}^p}^{\rho^p}(T^p, s, c)$ that $initialized_s = ff$ holds. According to Figure 7.12, there is a state $s' \in S^p$ such that $s \xrightarrow{c}_{T^p} s'$. Since s' differs from s only in the value of the high variable, we have $mem_{s'} =_L mem_s$. Conditions 1–9 are true for s and s' (Condition 8 is ensured by Lemma E.4.2). Consequently, $s \bowtie s'$ holds.

Secondly, assume $c = trans(cid_h, val)$. According to Figure 7.12, there is a state $s' \in S^p$ such that $s \xrightarrow{c}_{T^p} s'$. Since s' differs from s only in the value of $inbuf(cid_h)$. Since $level(cid_h) = high$, we have $inbuf_{s'} =_L inbuf_s$. Conditions 1–9 are true for s and s' (Condition 8 is ensured by Lemma E.4.2). Consequently, $s \bowtie s'$ holds.

osc$_{\mathcal{HI}^p}(T^p, \bowtie)$ Let $s_1, s_1', s_2' \in S^p$ and $e \in V^p \cup N^p$ be arbitrary. Assume $reachable(SES^p, s_1)$, $reachable(SES^p, s_2)$, $s_1' \xrightarrow{e}_{T^p} s_2'$, and $s_1' \bowtie s_1$. Based on Definition 7.4.2 we distinguish the following cases for e (omitting the index p from events and state objects for better readability):

$e = \textbf{\textit{schedule}}(\textbf{\textit{tid}})$ Since e is enabled in s'_1, we have $atid_{s'_1} = \bot$, $thread_{s'_1}(tid) \notin \{\bot, \top, \langle\rangle\}$, $tid \notin blocked\text{-}set_{s'_1}$, and $initialized_{s'_1} = tt$. Since $s_1 \bowtie s'_1$, we also have $atid_{s_1} = \bot$ (from Condition 4), $thread_{s_1}(tid) \notin \{\bot, \top, \langle\rangle\}$ (from Condition 8), $tid \notin blocked\text{-}set_{s_1}$ (from Condition 9), and $initialized_{s_1} = tt$ (from Condition 1). Hence, e is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e \rangle$. Obviously, $\delta|_{C^p} = \langle\rangle$, $\delta|_{V^p} = \langle e \rangle|_{V^p}$, and $s_1 \stackrel{\delta}{\Longrightarrow}_{T^p} s_2$ are all true for these choices. It remains to show $s'_2 \bowtie s_2$.

An occurrence of e affects only the values of *atid*, *inbuf*, and *pending*. According to Figure 7.12, the value of *atid* and *inbuf* is identical in s_2 and s'_2. Moreover, $pending_{s_2} =_L pending_{s'_2}$ follows from $inbuf_{s_1} =_L inbuf_{s'_1}$ $pending_{s_1} =_L pending_{s'_1}$ (from Condition 5), and the way in which *pending* is updated by *schedule*-events (cf. Figure 7.12). We conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{yield}}(\textbf{\textit{info}})$ Since e is enabled in s'_1, we have that $executed_{s'_1} = tt$, $info = (ainfo_{s'_1}, mem_{s'_1}(l), blocked\text{-}set_{s'_1})$, and $outbuf_{s'_1} = \langle\rangle$. Since $s'_1 \bowtie s_1$, we also have $executed_{s_1} = tt$ (from 2) and $info = (ainfo_{s_1}, mem_{s_1}(l), blocked\text{-}set_{s_1})$ (from 3, 7, and 9). Moreover, we have either $outbuf_{s_1} = \langle\rangle$ or $outbuf_{s_1} = (cid_h, val)$ for some $cid_h \in CID$ with $level(cid_h) = high$ and $sender(cid_h) = p$ (from 6). We make a case distinction.

If $outbuf_{s_1} = \langle\rangle$ then e is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e \rangle$. With this choice, it only remains to show $s'_2 \bowtie s_2$. An occurrence of e affects only the value s of *executed*, *atid*, and *ainfo*. According to Figure 7.12, the value of these state variables is identical in s'_2 and s_2. Conditions 1–9 are all satisfied for s'_2 and s_2. Consequently, $s'_2 \bowtie s_2$ holds.

If $outbuf_{s_1} = (cid_h, val)$ then e *is not* enabled in s_1 but the event $e' = trans(cid_h, val)$ *is* enabled, i.e. there is a state s with $s_1 \xrightarrow{e'}_{T^p} s$. An occurrence of e' affects only the value of $outbuf$ ($sender(cid_h) = p$) and $outbuf_s = \langle\rangle$ holds. Hence, e is enabled in s, i.e. there is a state s' with $s \xrightarrow{e}_{T^p} s'$. We choose $s_2 = s'$ and $\delta = \langle e'.e \rangle$. The rest of the argument is along the same lines as in the previous case.

$e = \textbf{\textit{outvar}}(h, \textbf{\textit{val}})$ We choose $s_2 = s_1$ and $\delta = \langle\rangle$. Since *outvar*-events have no effect on the state and $\langle e \rangle|_{V^p} = \langle\rangle$ holds, we conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{outvar}}(l, \textbf{\textit{val}})$ Since e is enabled in s'_1, we have $mem_{s'_1}(l) = val$. Since $s'_1 \bowtie s_1$, we have $mem_{s_1}(l) = val$ (from 7). Hence, e is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e \rangle$. Since *outvar*-events have no effect on the state, we have $s_1 = s_2$ and $s'_1 = s'_2$. We conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{setvar}}(l, \textbf{\textit{val}})$ Since e is enabled in s'_1, we have $initialized_{s'_1} = ff$. Since $s'_1 \bowtie s_1$, we have $initialized_{s_1} = ff$ (from 1). Hence, e is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e \rangle$. An occurrence of e affects only the value of $mem(l)$. According to Figure 7.12, the value of l is identical in s'_2 and s_2. We conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{start}}$ Since e is enabled in s'_1, we have $initialized_{s'_1} = ff$. Since $s'_1 \bowtie s_1$, we have $initialized_{s_1} = ff$ (from 1). Hence, e is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e \rangle$. An occurrence of e affects only the value of *initialized*. According to Figure 7.12, the value of this state variable is identical in s'_2 and s_2. We conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{trans}}(\textbf{\textit{cid}}_h, \textbf{\textit{val}})$ (where $sender(cid_h) = p$ and $level(cid_h) = high$) Since e is enabled in s'_1, $outbuf_{s'_1} = (cid_h, val)$ holds. The occurrence of e only affects *outbuf* and $outbuf_{s'_2} = \langle\rangle$ holds. Since $s'_1 \bowtie s_1$, we have either $outbuf_{s_1} = \langle\rangle$ or $outbuf_{s_1} = (cid'_h, val')$ for some $cid'_h \in CID$ with $level(cid'_h) = high$ and $sender(cid'_h) = p$ (from 6). We make a case distinction.

If $outbuf_{s_1} = \langle\rangle$ then choose $s_2 = s_1$ and $\delta = \langle\rangle$. With this choice, Conditions 1–9 are all satisfied for s'_2 and s_2 because $s'_1 \bowtie s_1$ holds. Consequently, $s'_2 \bowtie s_2$ holds.

If $outbuf_{s_1} = (cid'_h, val')$ then the event $e' = trans(cid'_h, val')$ is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e'}_{T^p} s$. An occurrence of e' affects only the value of *outbuf* ($sender(cid'_h) = p$ holds) and $outbuf_s = \langle\rangle$ holds. Hence, we have $outbuf_s = \langle\rangle = outbuf_{s'_2}$ (implies $outbuf_s =_L outbuf_{s'_2}$). We choose $s_2 = s$ and $\delta = \langle e'\rangle$. With this choice, Conditions 1–9 are all satisfied for s'_2 and s_2 because $s'_1 \bowtie s_1$ holds. Consequently, $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{trans}}(\textbf{\textit{cid}}_l, \textbf{\textit{val}})$ (where $sender(cid_l) = p$ and $level(cid_l) = low$) Since e is enabled in s'_1, $outbuf_{s'_1} = (cid_l, val)$ holds. Since $s'_1 \bowtie s_1$, we have $outbuf_{s_1} = (cid_l, val)$ (from 6). Hence, e is enabled in s_1, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e\rangle$. The occurrence of e only affects *outbuf* and $outbuf_{s_2} = \langle\rangle = outbuf_{s'_2}$ holds. We conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e = \textbf{\textit{trans}}(\textbf{\textit{cid}}_l, \textbf{\textit{val}})$ (where $receiver(cid_l) = p$ and $level(cid_l) = low$) Event e is enabled in s_1 because it has no preconditions, i.e. there is a state s with $s_1 \xrightarrow{e}_{T^p} s$. We choose $s_2 = s$ and $\delta = \langle e\rangle$. An occurrence of e affects only the value of $inbuf(cid_l)$. According to Figure 7.12, the value of this state variable is identical in s_2 and s'_2 because $inbuf_{s_1}(cid_l) = inbuf_{s'_1}(cid_l)$ holds (follows from $inbuf_{s_1} =_L inbuf_{s'_1}$). We conclude from $s'_1 \bowtie s_1$ that $s'_2 \bowtie s_2$ holds.

$e \in E^p_{\textbf{\textit{local}}}$ Since $s'_1 \xrightarrow{e}_{T^p} s'_2$, we have $initialized_{s'_1} = tt$, $initialized_{s'_2} = tt$, $executed_{s'_2} = tt$, and $atid_{s'_2} = atid_{s'_1}$. In order to make Theorems E.3.5 and E.3.7 applicable, we define some auxiliary states.

Let $t_1, t'_1, t'_2 \in S^p$ be states for which $inbuf_{t_1}(cid) = \langle\rangle$, $inbuf_{t'_1}(cid) = \langle\rangle$, and $inbuf_{t'_2}(cid) = \langle\rangle$ hold for all $cid \in CID$. Moreover, except for the value of *inbuf*, the states t_1, t'_1 and t'_2 shall be identical to s_1, s'_1 and s'_2, respectively. In particular, we have $t'_1 \xrightarrow{e}_{T^p} t'_2$ (value of *inbuf* neither affects the possibility of local events nor is it affected by their occurrence), $initialized_{t'_1} = tt$, $executed_{t'_1} = ff$, $atid_{t'_1} \neq \perp$, and $t'_1 \bowtie t_1$. Note that if $pending_{t'_1}(cid) = \langle\rangle$ ($pending_{t_1}(cid) = \langle\rangle$) then $channel(t'_1)(cid) = \langle\rangle$ ($channel(t_1)(cid) = \langle\rangle$) holds. In particular, if $e = ite\text{-}rcv^{ff}_p(cid, \ldots)$ then $channel(t'_1)(cid) = \langle\rangle$ holds ($pending_{t'_1}(cid) = \langle\rangle$ is implied by the precondition of e). According to our construction, Theorem E.3.5 is applicable for t'_1 and t'_2.[2] We obtain $\langle\!| thread_{t'_1}(atid_{t'_1}), mem_{t'_1}, channel(t'_1) |\!\rangle \twoheadrightarrow \langle\!| cseq_{\mathsf{aux}}(atid_{t'_2}, thread_{t'_2}), mem_{t'_2}, channel(t'_2) |\!\rangle$. Since $t'_1 \bowtie t_1$, we have $mem_{t'_1} =_L mem_{t_1}$ (from 7), $channel(t'_1) =_L channel(t_1)$ (from 5 and 6), and $thread_{t'_1}(atid_{t'_1}) \cong_L thread_{t_1}(atid_{t_1})$ (from 8). Unwinding the definition of a low bisimulation (cf. Def-

[2] Theorem E.3.5 presumes that the state-event system under consideration corresponds to a DMWL process pool while the state-event system considered here, is defined as a single DMWL process. Nevertheless, the theorem is applicable in the current context because the state-event system for a DMWL process pool consisting of a single process only is identical to the state-event system for that process (cf. Definition 7.3.8). For the same reason, Theorem E.3.7 can be applied in the following.

inition 7.2.7) for $\approx_L$ (a strong low-bisimulation is also a low bisimulation) yields that there are $\vec{D}'$, mem', and σ' such that $\langle\!| thread_{t_1}(atid_{t_1}), mem_{t_1}, channel(t_1) |\!\rangle \twoheadrightarrow \langle\!| \vec{D}', mem', \sigma' |\!\rangle$, $mem_{t_2'} =_L mem'$, $channel(t_2') =_L \sigma'$, and $cseq_{\mathtt{aux}}(atid_{t_2'}, thread_{t_2'}) \approx_L \vec{D}'$ hold. Thus, we can apply Theorem E.3.7. This yields that there are an event $e' \in E^p_{local}$ and a state t with $t_1 \xrightarrow{e'} t$, $mem_t = mem'$, $channel(t) = \sigma'$, and $cseq_{\mathtt{aux}}(atid_t, thread_t) = \vec{D}'$.

Let $s_2 \in S^p$ be a state for which $inbuf_{s_2}(cid) = inbuf_{s_1}(cid)$ holds for all $cid \in CID$. Moreover, except for the value of *inbuf*, the state s_2 shall be identical to t. This means, we have $s_1 \xrightarrow{e'} s_2$, $initialized_{s_2} = tt$, $executed_{s_2} = tt$, $atid_{s_2} = atid_{s_1}$, $mem_{s_2} = mem'$, and $cseq_{\mathtt{aux}}(atid_{s_2}, thread_{s_2}) = \vec{D}'$. For all $cid \in CID$ with $receiver(cid) = p$, we have $channel(s_2)(cid) = inbuf_{s_1}(cid).\sigma'(cid)$.

It remains to show that $s_2' \bowtie s_2$ holds. Let us investigate conditions 1–9.

1. We have already shown that $initialized_{s_2'} = tt$ and $initialized_{s_2} = tt$ hold.
2. We have already shown that $executed_{s_2'} = tt$ and $executed_{s_2} = tt$ hold.
3. Firstly, assume that e is no *fork*-event. We have $thread_{s_2'}(atid_{s_1'}) = \langle\rangle = thread_{s_2}(atid_{s_1'})$ or $thread_{s_2'}(atid_{s_1'}) \approx_L thread_{s_2}(atid_{s_1'})$ (cf. proof of condition 8 below). Hence, $terminates(thread_{s_2'}(atid_{s_1'}) = terminates(thread_{s_2}(atid_{s_1'})$ holds.
   Secondly, assume that e is a *fork*-event. If k child threads are spawned after computation starts in s_1' then k threads are spawned after computation starts in s_1 because $cseq_{\mathtt{aux}}(atid_{s_2'}, thread_{s_2'}) \approx_L cseq_{\mathtt{aux}}(atid_{s_2}, thread_{s_2})$ (cf. proof of condition 8 below).
   Consequently, in both cases, $atid_{s_2'} = atid_{s_1'}$ holds.
4. We have that $atid_{s_2'} = atid_{s_1'}$ and $atid_{s_2} = atid_{s_1}$ hold. Together with $atid_{s_1'} = atid_{s_1}$ (from $s_1' \bowtie s_1$), this implies $atid_{s_2'} = atid_{s_2}$.
5. From $inbuf_{s_2'} = inbuf_{s_1'}$ (local events do not affect the value of *inbuf*), $inbuf_{s_1'} =_L inbuf_{s_1}$ $(s_1' \bowtie s_1)$, and $inbuf_{s_1} = inbuf_{s_2}$ we conclude $inbuf_{s_2'} =_L inbuf_{s_2}$.
6. We have that $outbuf_{s_2'} = \langle\rangle$ or $outbuf_{s_2'} = (cid', val')$ holds for some $cid' \in CID$ with $sender(cid') = p$. Moreover, $outbuf_{s_2} = \langle\rangle$ or $outbuf_{s_2} = (cid'', val'')$ for some $cid'' \in CID$ with $sender(cid'') = p$. Finally, we have $outbuf_{s_2'}(cid) = channel(s_2')(cid)$, $channel(s_2')(cid) = channel(t_2')(cid)$, $channel(t_2')(cid) =_L \sigma'$, $\sigma' = channel(t)$, $channel(t) = channel(s_2)(cid)$, $channel(s_2)(cid) = outbuf_{s_2'}(cid)$ for all $cid \in CID$ with $sender(cid) = p$. Hence, $outbuf_{s_2'} =_L outbuf_{s_2}$ holds.
7. We have that $mem_{s_2'} = mem_{t_2'}$, $mem_{t_2'} =_L mem'$, and $mem' = mem_{s_2}$ hold. This implies that $mem_{s_2'} =_L mem_{s_2}$ also holds.
8. For all $tid \in TID$, $thread_{s_1'}(tid) = thread_{t_1'}(tid)$, $thread_{s_2'}(tid) = thread_{t_2'}(tid)$, $thread_{s_1}(tid) = thread_{t_1}(tid)$, and $thread_{s_2}(tid) = thread_t(tid)$ hold.
   Firstly, assume that e is no *fork*-event. Then $thread_{t_2'}(tid) = thread_{t_1'}(tid)$ and $thread_{t_2}(tid) = thread_{t_1}(tid)$ hold if $tid \neq atid_{s_1'}$ (recall that $atid_{t_1'} = atid_{s_1'}$ and $atid_{t_1} = atid_{s_1'}$). We obtain from $s_1' \bowtie s_1$ and the above that

$$\begin{aligned} &thread_{s_2}(tid) = \bot = thread_{s_2'}(tid) \vee thread_{s_2}(tid) = \top = thread_{s_2'}(tid) \\ &\vee thread_{s_2}(tid) = \langle\rangle = thread_{s_2'}(tid) \vee thread_{s_2}(tid) \approx_L thread_{s_2'}(tid) \end{aligned}$$

   holds if $tid \neq atid_{s_1'}$. For $tid = atid_{s_1'}$, we have $cseq_{\mathtt{aux}}(tid, thread_{s_2'}) \approx_L \vec{D}'$ and

$\vec{D'} = cseq_{\mathsf{aux}}(tid, thread_{s_2})$. Consequently,

$$thread_{s_2}(tid) = \langle\rangle = thread_{s'_2}(tid) \vee thread_{s_2}(tid) \approx_L thread_{s'_2}(tid)$$

holds. Summarizing, Condition 8 is fulfilled for s_2 and s'_2.
Secondly, assume that e is a *fork*-event. In this case, one has to consider the child threads in addition to the active thread but, otherwise, this case can be handled along the same lines as the first one ($thread_{s'_2}(atid_{s'_1}) = \top = thread_{s_2}(atid_{s'_1})$ holds for the active threads and the respective child threads are pairwise bisimilar because $cseq_{\mathsf{aux}}(atid_{s'_2}, thread_{s'_2}) \approx_L cseq_{\mathsf{aux}}(atid_{s_2}, thread_{s_2})$).

9. In general, $first(C') = \mathsf{receive}(cid, var)$ and $C' \approx_L C'$ imply $dom_{ch}(cid) = low$ because a blocking receive on high channels is not strongly secure. The consumption of a value from a low channel and the blocking of a thread because of an empty incoming channel is both observable. Hence, the commands that are strongly bisimilar must behave identically in this respect. That is, if $first(C') = \mathsf{receive}(cid, var)$ and $C' \approx_L C$ then $first(C) = \mathsf{receive}(cid, var)$ must hold. Hence, two commands that are strongly bisimilar either both start with a receive-command (on the same low channel) or both don't start with a receive-command.
Since $atid_{s'_2} \neq \bot$ (precondition of events in E^p_{local}), $atid_{s'_2} = atid_{s_2}$ (from 4 above), and

$$\begin{aligned} &thread_{s_2}(tid) = \bot = thread_{s'_2}(tid) \vee thread_{s_2}(tid) = \top = thread_{s'_2}(tid) \\ &\vee thread_{s_2}(tid) = \langle\rangle = thread_{s'_2}(tid) \vee thread_{s_2}(tid) \approx_L thread_{s'_2}(tid) \end{aligned}$$

(from 8 above) we have $blocked\text{-}set_{s'_2} = blocked\text{-}set_{s_2}$.

We conclude that $s'_2 \bowtie s_2$ holds. □

Lemma E.4.2. Let $C \in CMD$ be a DMWL command, $p \in PID$ be a process identifier, and Tr be the set of possible traces induced by the state-event system $DMWLProcess(p, C)$ (abbreviated by $SES^p = (S^p, s_0^p, E^p, I^p, O^p, T^p)$). Moreover, let $s \in S^p$ be a state and $\tau \in Tr$ be a trace for which $s_0^p \stackrel{\tau}{\Longrightarrow} s$ holds.

If C is strongly secure then

$$\forall tid \in TID.\, (thread_s(tid) \notin \{\bot, \top, \langle\rangle\} \Rightarrow thread_s(tid) \approx_L thread_s(tid)) \qquad \diamond$$

Proof. The proof is by induction on the length of τ.

In the *base case* ($\tau = \langle\rangle$), we have $thread_s(\langle 0\rangle) = C$ and $thread_s(tid) = \bot$ for all $tid \in TID$ with $tid \neq \langle 0\rangle$. Since C is strongly secure, we have $thread_s(0) \approx_L thread_s(0)$. Hence, the proposition holds in the base case.

In the *step case* ($\tau = \beta.\langle e\rangle$), we have $s_0 \stackrel{\beta}{\Longrightarrow} s' \stackrel{e}{\longrightarrow} s$ (for some state $s' \in S^p$. In order to apply Theorem E.3.5 and E.3.7, we define two auxiliary states.

Let $t, t' \in S^p$ be states for which $inbuf_t(cid) = \langle\rangle$ and $inbuf_{t'}(cid) = \langle\rangle$ hold for all $cid \in CID$. Moreover, except for the value of *inbuf*, the states t and t' shall be identical to s and s', respectively. In particular, we have $t' \stackrel{e}{\longrightarrow}_{T^p} t$ (value of *inbuf* neither affects the possibility of local events nor is it affected by their occurrence), $initialized_{t'} = tt$, $executed_{t'} = ff$, $atid_{t'} \neq \bot$, $thread_{t'}(atid_{t'}) \notin \{\bot, \top, \langle\rangle\}$, $initialized_t = tt$, $executed_t = tt$, and $atid_t = atid_t$. Note that if $pending_{t'}(cid) = \langle\rangle$ (or $pending_t(cid) = \langle\rangle$) then $channel(t')(cid) = \langle\rangle$ (or $channel(t)(cid) = \langle\rangle$,

respectively) holds. In particular, if $e = \mathit{ite\text{-}rcv}_p^{ff}(cid, \ldots)$ then $\mathit{channel}(t')(cid) = \langle\rangle$ holds ($\mathit{pending}_{t'}(cid) = \langle\rangle$ is implied by the precondition of e).

From the induction assumption, we obtain that $\mathit{thread}_{s'}(tid) \approx_L \mathit{thread}_{s'}(tid)$ holds for all $tid \in \mathit{TID}$ with $\mathit{thread}_{s'}(tid) \notin \{\bot, \top, \langle\rangle\}$. If $e \notin E^p_{local}$ then the value of *thread* remains unchanged and the proposition follows directly. In the following, let us assume that $e \in E^p_{local}$ holds. We distinguish two cases for e:

$e \neq \textbf{\textit{fork}}(\ldots)$ Theorem E.3.5 and $t' \xrightarrow{e} t$ imply that $\langle\!|\mathit{thread}_{t'}(\mathit{atid}_{t'}), \mathit{mem}_{t'}, \mathit{channel}(t')|\!\rangle \twoheadrightarrow \langle\!|\mathit{thread}_t(\mathit{atid}_t), \mathit{mem}_t, \mathit{channel}(t)|\!\rangle$ holds. For all $tid \in \mathit{TID}$ with $\mathit{thread}_{s'}(tid) \notin \{\bot, \top, \langle\rangle\}$, we have $\mathit{thread}_{t'}(tid) \approx_L \mathit{thread}_{t'}(tid)$ because $\mathit{thread}_{s'}(tid) \approx_L \mathit{thread}_{s'}(tid)$ (by induction assumption) and $\mathit{thread}_{t'} = \mathit{thread}_{s'}$. Unwinding the definition of low bisimulation (cf. Definition 7.2.7) for $\approx_L$ (a strong low-bisimulation is also a low bisimulation) yields that there are C', mem', and σ' such that $\langle\!|\mathit{thread}_{t'}(\mathit{atid}_{t'}), \mathit{mem}_{t'}, \mathit{channel}(t')|\!\rangle \twoheadrightarrow \langle\!|C', \mathit{mem}', \sigma'|\!\rangle$ and $\mathit{thread}_t(\mathit{atid}_t) \approx_L C'$ hold. Since $\twoheadrightarrow$-transitions are deterministic, we also have $C' = \mathit{thread}_t(\mathit{atid}_t)$. Consequently, $\mathit{thread}_t(\mathit{atid}_t) \approx_L \mathit{thread}_t(\mathit{atid}_t)$ holds. Together with $\mathit{thread}_t = \mathit{thread}_s$, this implies that $\mathit{thread}_s(\mathit{atid}_s) \approx_L \mathit{thread}_s(\mathit{atid}_s)$ holds. For all $tid \in \mathit{TID}$ with $tid \neq \mathit{atid}_s$, we have $\mathit{thread}_s(tid) = \mathit{thread}_{s'}(tid)$ and, hence, if $\mathit{thread}_s(tid) \notin \{\bot, \top, \langle\rangle\}$ then $\mathit{thread}_s(tid) \approx_L \mathit{thread}_s(tid)$ holds (follows from $\mathit{thread}_{s'}(tid) \approx_L \mathit{thread}_{s'}(tid)$).

$e = \textbf{\textit{fork}}(C, D_1 \ldots D_n)$ From Theorem E.3.5 and $t' \xrightarrow{e} t$, we obtain

$$\begin{aligned}&\langle\!|\mathit{thread}_{t'}(\mathit{atid}_{t'}), \mathit{mem}_{t'}, \mathit{channel}(t')|\!\rangle\\ &\twoheadrightarrow \langle\!|\mathit{thread}_t(\mathit{atid}_t.\langle 0\rangle) \ldots \mathit{thread}_t(\mathit{atid}_t.\langle n\rangle), \mathit{mem}_t, \mathit{channel}(t)|\!\rangle\end{aligned}$$

For all $tid \in \mathit{TID}$ with $\mathit{thread}_{s'}(tid) \notin \{\bot, \top, \langle\rangle\}$, we have $\mathit{thread}_{t'}(tid) \approx_L \mathit{thread}_{t'}(tid)$ because $\mathit{thread}_{s'}(tid) \approx_L \mathit{thread}_{s'}(tid)$ (by induction assumption) and $\mathit{thread}_{t'} = \mathit{thread}_{s'}$. Unwinding the definition of low bisimulation (cf. Definition 7.2.7) for $\approx_L$ yields that there are $D'_0 \ldots D'_n$, mem', and σ' such that $\langle\!|\mathit{thread}_{t'}(\mathit{atid}_{t'}), \mathit{mem}_{t'}, \mathit{channel}(t')|\!\rangle \twoheadrightarrow \langle\!|D'_0 \ldots D'_n, \mathit{mem}', \sigma'|\!\rangle$ and $\mathit{thread}_t(\mathit{atid}_t.\langle i\rangle) \approx_L D'_i$ (for all $i \in \{0, \ldots, n\}$). Moreover, since $\twoheadrightarrow$-transitions are deterministic, we have $\mathit{thread}_t(\mathit{atid}_t.\langle i\rangle) = D'_i$. Consequently, $\mathit{thread}_t(\mathit{atid}_t.\langle i\rangle) \approx_L \mathit{thread}_t(\mathit{atid}_t.\langle i\rangle)$ holds. Together with $\mathit{thread}_t = \mathit{thread}_s$, this implies that $\mathit{thread}_s(\mathit{atid}_s.\langle i\rangle) \approx_L \mathit{thread}_s(\mathit{atid}_s.\langle i\rangle)$ holds for all $i \in \{0, \ldots, n\}$. For all $tid \in \mathit{TID}$ with $tid \notin \{\mathit{atid}_s, \mathit{atid}_s.\langle 0\rangle, \ldots, \mathit{atid}_s.\langle n\rangle\}$, we have $\mathit{thread}_s(tid) = \mathit{thread}_{s'}(tid)$ and, hence, if $\mathit{thread}_s(tid) \notin \{\bot, \top, \langle\rangle\}$ then $\mathit{thread}_s(tid) \approx_L \mathit{thread}_s(tid)$ holds (follows from $\mathit{thread}_{s'}(tid) \approx_L \mathit{thread}_{s'}(tid)$). We also have $\mathit{thread}_s(\mathit{atid}_s) = \top$. □

E.5 Differences to Prior Versions of the Case Study

In Section 7.7, we have explained the *main* differences between the second version of our case study and the version in Chapter 7. The remaining differences are due to a slightly different focus in the two versions.

In [MS03a], the main goal was to find an information flow property that *precisely* corresponds to strong security (i.e. having a completeness result in addition to the soundness result). In Chapter 7, the main goal was to illustrate the identification of security requirements, their specification by an information flow property, and the verification of this property (soundness). Completeness was not so much an issue. This resulted in a minor difference of the security predicates used in the definitions of IFP^p (or IFP^P) and *SecProp*. While BSIA^{p^p}

is used in the definition of IFP^p, *BSI* is used in the definition of *SecProp*, instead. Thus, the property *SecProp* is slightly stronger than IFP^p (recall the ordering of these BSPs from Theorem 3.5.12). More precisely, *SecProp* requires that *setvar*-events on high variables can be inserted at every point of a trace (*BSI*) rather than only at points before a *start*-event has occurred ($BSIA^{\rho^p}$). This difference in the security predicate was essential to arrive at a completeness result in [MS03a]. However, demanding that *setvar*-events can be inserted at every point of a trace does not comply with our specification of DMWL processes (or DMWL process pools) according to Definition 7.3.3 (or Definition 7.3.8, respectively). This is because, according to our system specification, *setvar*-events are only enabled before the initialization phase has been completed (modeled by the occurrence of a *start*-event), which models that the variables of a DMWL process can only be set by the environment before program execution begins. After program execution begins, the value of variables can only be changed by the program. The system specification in [MS03a] necessarily differs in this respect from the one in Chapter 7 because that specification satisfies *SecProp* (and, hence, also *BSI*). That is, in [MS03a], there is no notion of an initialization phase (no *start*-events) and *setvar*-events are always enabled.

In general, changes to the system specification (even if they appear minor at first sight) can have severe consequences. In particular, this is true when the set of possible traces differ. Since the system specifications in [MS03a] and in Chapter 7 yield different sets of possible traces, it might appear somewhat strange that both specifications are adequate models of DMWL programs. Nevertheless, we have presented adequateness theorems for our specification in Chapter 7 (in Appendix E.3.2) and similar adequateness theorems have been proved for the specification of DMWL processes in [MS03a]. The reason why this is possible is that the initialization phase (*setvar*-events and *start*-events) is not considered in the adequateness theorems and this is the only aspect in which the two system specifications differ. What is an adequate model of the initialization for DMWL (or MWL) is hard to say because the initialization of variables is not part of the language definition. In Chapter 7, we have tried to stick more closely to the intuitive meaning of "initialization" in our specification by permitting changes to the value of variables by the environment only *before* program execution begins. In [MS03a], we have taken more freedom in our interpretation and have permitted changes to variables by the environment at any point of time. Both system specifications are sensible in our opinion. They constitute two ends of a spectrum of possible interpretations of the initialization of variables in DMWL (MWL). Moreover, the resulting versions of the case study are complementary in the sense that they illuminate different aspects of the strong security condition. The version in [MS03a] focuses on what the information flow property is that corresponds precisely to strong security (providing a basis for comparisons with other security definitions in *MAKS*). The version in Chapter 7, shows that strong security also results in a sensible degree of security if one assumes a rigorous interpretation of the initialization phase.