Expressivity and Decidability of First-order Languages over Feature Trees

Dissertation
zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Technischen Fakultät
der Universität des Saarlandes

von

Rolf Backofen

Saarbrücken
1994

Tag des Kolloquiums:	22. Dezember 1994
Vorsitzender:	Prof. Dr. Harald Ganzinger
Berichterstatter:	Prof. Dr. Gert Smolka
	Prof. Dr. Hans Uszkoreit

Statement

A completeness proof for a variant of FT′ has been worked out with my advisor Gert Smolka and has been published in [BS93a]. A completeness proof of a variant of CFT′ has been published before in [Bac95a]. Chapter 5 of this thesis is an adaption and slight extension of [Bac94] to an infinite signature.

Acknowledgement

This research was carried out at the German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, DFKI), and I want to thank this institute for its support.

I am very grateful to Gert Smolka, who as my main advisor provided the inspiration, stimulating enthusiasm and technical guidance which helped me in writing this thesis. He encouraged me to carry out this research by raising challenging questions, and I appreciate many interesting discussions with him. Furthermore, I want to thank Hans Uszkoreit, my second advisor, for providing an excellent research environment and giving valuable comments on preliminary drafts of this thesis.

Additionally, I am grateful to all my colleagues at the Computational Linguistics Department of DFKI, and to all the people who helped me with discussions and comments, and without whom this thesis would not be what it is: Jochen Dörre, Andreas Podelski, Christian Schulte, Stephen Spackman. Particularly, I am indebted to Hans-Ulrich Krieger, Joachim Niehren and Ralf Treinen, who I always found willing to listen to my problems.

The following people have read draft versions of this thesis: Stephan Busemann, Elizabeth Hinkelman, Hans-Ulrich Krieger, Günter Neumann, Joachim Niehren, Stephen Spackman, Ralf Treinen.

Above all, I am grateful to my wife Doris, who always gave me support, encouragement and new energy, although she had a hard time herself.

Zusammenfassung

Diese Arbeit untersucht die formalen Grundlagen von Merkmalsbeschreibungen, die partielle Beschreibungen von abstrakten, record-ähnlichen Objekten darstellen. Die Beschreibungen verwenden funktionale Attribute genannt Merkmale. Merkmalsbeschreibungen tauchten erstmals in den späten siebziger Jahren im Zusammenhang mit sogenannten constraint-basierten Grammatikformalismen auf [Kay79, KB82, Shi86]. In neuerer Zeit wurde die Verwendung von Merkmalsbeschreibungen in Constraint-Programmiersprachen vorgeschlagen und untersucht [AKN86, AKN89, AKP93, AKPS94, ST94]. Eine der wesentlichen Operationen auf Merkmalsbeschreibungen ist die Unifikation, deren Eingabe zwei Merkmalsbeschreibungen sind und die entweder fehlschlägt, falls die Merkmalsbeschreibungen inkompatibel sind, oder als Ergebnis eine Merkmalsbeschreibung liefert, die die Information der Eingaben kombiniert.

Eine der zentralen Fragestellungen, mit der sich die Literatur über Merkmalsbeschreibungen beschäftigt, ist die Definition einer geeigneten Semantik für Merkmalsbeschreibungen als auch für die Unifikationsoperation. Es gab viele und auch sehr divergierende Formalisierungen von Merkmalsbeschreibungen. Diese Arbeit vertritt hierzu einen klaren Standpunkt. Aufbauend auf den Arbeiten von Johnson [Joh88] und Smolka [Smo88] betrachten wir Merkmalsbeschreibungen (und deren Erweiterungen) als Formeln in geeigneten Sprachen erster Ordnung. Die Semantik von Merkmalsbeschreibungen wird über eine Merkmalstheorie (also eine Menge von geschlossenen Formeln (Sätzen)) definiert. Die Unifikationsoperation wird als ein Erfüllbarkeitstest für konjunktiv verknüpfte Merkmalsbeschreibungen interpretiert.

Wir untersuchen in dieser Arbeit verschiedene in der Literatur eingeführte Merkmalsbeschreibungssprachen. Für die Definition der entsprechenden Merkmalstheorien verwenden wir eine Standardmethode der Logik erster Ordnung, die jedoch erst in jüngster Zeit auch auf Merkmalsbeschreibungen angewendet wurde [BS93a, AKPS94, ST94]. Wir definieren für eine Merkmalsbeschreibungsprache L eine Standardinterpretation $\mathfrak{T}_L$, deren Wertebereich aus sogenannten Merkmalsbäumen besteht. Merkmalsbäume sind Bäume, deren Kanten mit Merkmalen, und deren Blätter mit

Atomsymbolen beschriftet sind. Für die Merkmals- und Atomsymbole verwenden wir ein gemeinsames, unendliches Alphabet $\mathcal{L}$. Die Element von $\mathcal{L}^*$ werden im folgenden Pfade genannt. Die Merkmalstheorie für L definieren wir nun als die Menge aller Sätze, die in $\mathfrak{T}_L$ gültig sind. Diese Vorgehensweise hat den Vorteil, das sie intuitiv ist und eine vollständige Merkmalstheorie liefert (das heißt, für jeden L-Satz ϕ enthält die Theorie von $\mathfrak{T}_L$ entweder ϕ oder $\neg\phi$.)

Unsere Merkmalsbeschreibungssprachen enthalten für jedes Symbol aus $\mathcal{L}$ ein entsprechendes Konstantensymbol. Die Konstantensymbole werden in den Standardinterpretationen als atomare Merkmalsbäume interpretiert (also Merkmalsbäume, die keine Merkmale besitzen und mit einen Atomsymbol markiert sind). Im einzelnen werden wir folgende Sprachen näher untersuchen:

FT: Die Signatur von FT enthält ein unäres Relationssymbol *atom* sowie für jedes Element von $\mathcal{L}$ ein binäres Relationssymbol. Ein Merkmalsconstraint der Form xfy ist in der Standardinterpretation erfüllt, falls y der Unterbaum von x unter dem Merkmal f ist. FT wurde (in einer leicht modifizierten Form) in [BS93a, AKPS94] eingeführt. Sie ist eine Basissprache für Merkmalsbeschreibungen in der Hinsicht, daß alle anderen Merkmalssprachen die Ausdrucksmittel von FT zur Verfügung stellen.

CFT: Neben den Relationssymbolen aus FT enthält CFT für jede endliche Teilmenge $\{f_1, \ldots, f_n\}$ von $\mathcal{L}$ ein unäres Relationssymbol. Ein Aritätsconstraint $x\{f_1, \ldots, f_n\}$ ist in der Standardinterpretation von CFT erfüllt, falls x genau die Merkmale $\{f_1, \ldots, f_n\}$ besitzt. Diese Sprache wurde (wiederum leicht modifiziert) in [ST94] eingeführt.

In der Literatur über Merkmalsbescheibungsprachen wurde für die Sprache FT meistens eine andere als die von uns verwendete Standardinterpretation betrachtet. Der Wertebereich dieser Interpretation besteht aus sogennanten Merkmalsgraphen (das sind Graphen, deren Kanten mit Merkmalen beschriftet sind; siehe [Smo88, Smo92]).

Die nächsten beide Sprachen verallgemeinern die Merkmalsconstraints in FT zu Unterbaumrelationen auf zwei verschiedene Arten:

RFT: : Die Signatur von RFT enthält für jeden regulären Ausdruck L über dem Alphabeth $\mathcal{L}$ ein entsprechendes binäres Relationssymbol. Ein **regulärer Pfadausruck** xLy ist in der Standardinterpretation von RFT erfüllt, falls y ein Unterbaum von x unter einem Pfad $p \in \mathcal{L}^*$ ist und p in der durch L beschriebenen regulären Menge von Pfaden enthalten ist. Reguläre Pfadausdrücke wurden

in [KZ88, KM88] unter der Bezeichnung „functional uncertainty“ im Rahmen des Grammatikformalismus LFG [KB82] eingeführt.

F: Neben den Konstantensymbolen enthält die Signatur von F nur noch ein dreistelliges Relationssymbol $\cdot[\cdot]\cdot$. Ein **generalisiertes Merkmalsconstraint** $x[y]z$ ist in der Standardinterpretation von F erfüllt, falls y einen atomaren Merkmalsbaum denotiert, der mit dem Atomsybol f markiert ist, und z der Unterbaum von x under dem Merkmal f ist. Constraints ähnlicher Art wurden in [Joh88, Tre93] betrachtet.

Wir untersuchen in dieser Arbeit die Theorien dieser Sprachen unter den folgenden Aspekten:

Sind Merkmalsbäume ein adäquater Bereich für Merkmalsbeschreibungen?
Oder um etwas spezifischer zu sein, wie verhält sich unsere Merkmalstheorie von FT (der Basissprache) zu der Theorie, die man erhält wenn man die Merkmalsgrapheninterpretation von FT zugrundelegt?

Was ist die Expressivität dieser Sprachen?
Zum einen wollen wir hier die Sprachen untereinander vergleichen. Zum anderen interessieren wir uns auch für die Frage, inwieweit weitere aus der Literatur bekannte Konzepte kodierbar sind.

Welche Fragment der Merkmalstheorien sind entscheidbar?
Einige Resultate können der Literatur entnommen werden. So ist zum Beispiel bekannt, daß die existentiellen Fragmente von F, FT und CFT entscheidbar sind. Jedoch ist nichts über die volle Theorie von FT und CFT bekannt, und es gibt nur partielle Resultate für das existentielle Fragment von RFT.

Beiträge der Arbeit

Die Hauptresultate der Arbeit sind folgende:

- Wir zeigen, daß die regulären Pfadausdrücke in der Signatur von RFT in F definierbar sind. Damit können alle RFT-Formeln in eine äquivalente F-Formel übersetzt werden (für FT und CFT wurden diese Resultate bereits in [Tre93] gezeigt). Desweitern zeigen wir, daß wichtige in der Literatur eingeführte Relationen in F definierbar sind.

Eine wichtige Klasse von Relationen, die in F kodiert werden können, sind solche, die mit Hilfe von definite Äquivalenzen definierbar sind. Definite Äquivalenzen wurden in [Smo93] eingeführt. Sie dienen dazu, Relationen in der Art und Weise zu definieren, wie es im logischen Programmieren üblich ist [Cla78, Smo93]. Daneben können auch die von den constraint-basierten Grammatikformalismen her bekannten Typsysteme in definite Äquivalenzen übersetzt werden.

- Wir stellen eine Axiomatisierung der Merkmalstheorien für FT und CFT auf. Da unsere Theorien vollständig sind, folgt daraus die Entscheidbarkeit dieser Theorien. Desweiteren zeigen wir, daß auch die Merkmalsgraphinterpretation von FT ein Modell der FT-Axiomatisierung ist. Dies bedeutet, das beide Standardinterpretationen genau die gleiche Semantik für FT liefern.

- Wir zeigen, daß das Erfüllbarkeitsproblem für Konjunktionen von regulären Pfadausdrücken entscheidbar ist. Dabei betrachten wir zunächst eine Formel als erfüllbar, falls sie in irgendeiner Interpetation von RFT erfüllbar ist (die nicht notwendigerweise unsere Standardinterpretation sein muß). Dies wurde in [KM88, BBN+93] als ein offenes Problem beschrieben. Desweiteren zeigen wir, daß die Standardinterpretation von RFT kanonisch für dieses Problem ist (das heißt, eine Konjunktion von regulären Pfadausdrücken ist erfüllbar, falls sie in der Standardinterpretation von RFT erfüllbar ist). Damit ist auch gezeigt, daß das positive existentielle Fragment der Theorie der Standardinterpretation von RFT entscheidbar ist.

Contents

Chapter 1

Introduction

This thesis investigates the formal foundations of feature descriptions, which are partial descriptions, by means of functional attributes called features, of abstract record-like objects. Feature descriptions originated in the late seventies with so-called constraint-based grammars [Kay79, KB82, Shi86], a by now popular family of declarative grammar formalisms for the description and processing of natural language. More recently, the use of feature descriptions in constraint programming languages has been advocated and studied [AKN86, AKN89, AKP93, AKPS94, ST94]. Figure 1.1 gives an example of a feature description. One of the main operations defined on feature descriptions is *unification*, which takes two feature descriptions and yields either a failure, if the feature descriptions contain conflicting information, or a feature description that combines the information from both input descriptions.

One of the main problems addressed in the literature is to provide an appropriate semantics for feature descriptions (and for various extensions of feature descriptions) as well as for the unification operation. There have been many diverging approach-

$$\exists z \left[\begin{array}{lcl} \mathit{sort} & : & \mathit{person} \\ \mathit{name} & : & \left[\begin{array}{lcl} \mathit{firstname} & : & \mathit{Kim} \\ \mathit{lastname} & : & \mathit{Brown} \end{array}\right] \\ \mathit{home\_address} & : & z = \left[\begin{array}{lcl} \mathit{street} & : & \mathit{Drexel\ Ave.} \\ \mathit{city} & : & \mathit{Chicago} \end{array}\right] \\ \mathit{office\_address} & : & z \end{array}\right]$$

Figure 1.1: An example of a feature description. It can be interpreted as a person which works at home.

es to formalising feature descriptions (see sections 1.2 and 1.3). This thesis takes a clear position: following [Joh88, Smo88], we consider feature descriptions and their extensions as formulae in specific first-order languages. The semantics for feature descriptions is provided by a feature theory, which is a set of closed formulae (sentences) having at least one model. Unification is interpreted as an operation testing the satisfiability of conjunctions of feature descriptions.

In this thesis we will investigate several existing feature description languages. For the definition of the corresponding feature theories we apply a standard first-order logic method which was used only recently by [BS93a, AKPS94, ST94] in the context of feature descriptions. Given a feature description language L, we interpret L over the fixed domain of *feature trees* resulting in a first-order structure $\mathfrak{T}_L$ (henceforth called the *standard interpretation of* L). Then we take the feature theory for L to be the set of sentences valid in $\mathfrak{T}_L$ (called the *theory of* $\mathfrak{T}_L$). This approach has the advantage that it is fairly intuitive. Furthermore, it yields a complete theory (i.e., for every L-sentence ϕ, either ϕ or $\neg\phi$ is contained in theory of $\mathfrak{T}_L$).

Feature trees are trees where the edges are labelled by features and the leaves are labelled by atoms. The labelling is functional, that is, the direct subtrees of a feature tree are uniquely determined by the features of the edges leading to them. The names of features and atoms are taken out of a common infinite set of labels $\mathcal{L}$. Formally, a feature tree is a pair consisting of a prefix-closed subset D of $\mathcal{L}^*$ together with a total mapping λ of the leaves of D (i.e., the set of elements p in D with the property that there is no feature f such that pf is in D) into $\mathcal{L}$. Examples of feature trees are listed in Figure 1.2. The elements of $\mathcal{L}^*$ are called paths.

Our feature description languages are first-order languages which do not contain proper function symbols, but contain for every symbol of $\mathcal{L}$ a corresponding constant symbol. As described above, every language is associated with a standard interpretation whose domain is the set of all feature trees. In this interpretation, the constant symbols are interpreted as atomic feature trees (i.e., as feature trees having no subtrees). In particular, we will consider the following languages (and their corresponding standard interpretations) in this thesis:

FT: The signature of FT contains a unary predicate symbol *atom*, and for every symbol in $\mathcal{L}$ a binary relation symbol. A **feature constraint** xfy with $f \in \mathcal{L}$ holds in the standard interpretation of FT iff y is the subtree of x under the feature f, and $atom(x)$ holds if x denotes an atomic feature tree. This language was introduced in [BS93a, AKPS94] with minor differences and contains the descriptional primitives used in every feature description language. Hence, we call this language the *basic* feature description language.

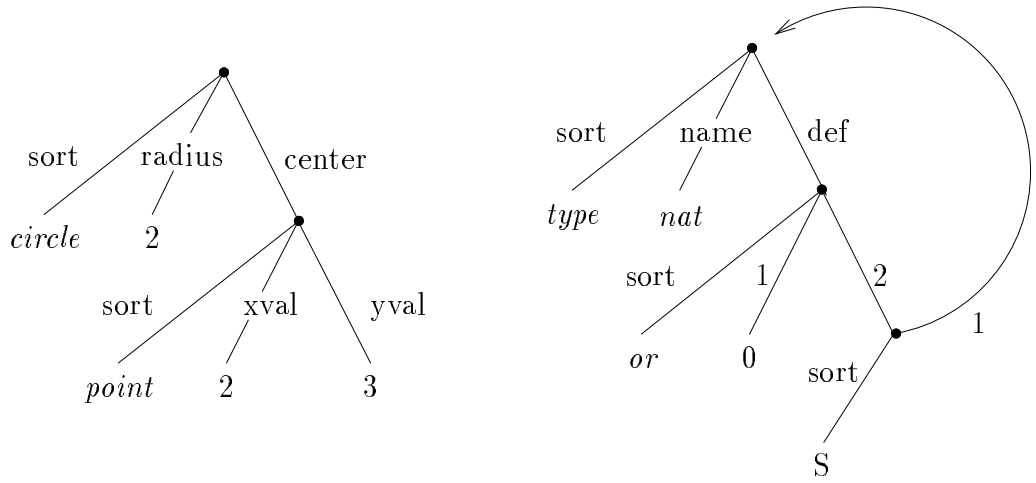


Figure 1.2: Examples of Feature Trees.

CFT: The signature of CFT is the signature of FT extended by a unary predicate symbol for every finite subset of $\mathcal{L}$. An **arity constraint** $x\{f_1, \ldots, f_n\}$ is true in the standard interpretation of CFT if x has exactly $f_1, \ldots, f_n$ as features under its root. This language permits the specification of complete information for some variable x by specifying the arity of x and the subtrees of x under the features listed in the arity of x. As we will show, this is not possible in the language of FT (the feature descriptions in FT are inherently partial). CFT was introduced in a slightly different form by [ST94] and combines the expressive power of FT with Colmerauer's rational tree constraint system RT for Prolog-II.

The next two languages generalise the feature constraints in FT to subtree relations in two different ways, providing additional expressivity:

RFT: The signature of RFT contains, for every regular expression L over the alphabet $\mathcal{L}$, a binary relation symbol. A **regular path expression** xLy holds in the standard interpretation of RFT if y is the subtree of x under some path p, where p is an element of the regular set of paths denoted by L. Using this language, it is now possible to specify properties of arbitrarily deep subtrees, where the paths leading to the subtrees can be restricted by a regular expression. Regular path expressions were introduced as "functional uncertainty" by [KZ88, KM88] for handling long-distance phenomena in the context of the grammar formalism LFG [KB82].

F: Beside the constant symbols, the signature of F contains only a ternary relation symbol $\cdot[\cdot]\cdot$. A **generalised feature constraint** $x[y]z$ is true in the standard interpretation of F if y denotes an atomic feature tree and z is the subtree of x under the feature f, where f is the label of the atomic feature tree denoted by y. In this language, features are now first class objects (i.e., it is now possible to quantify over features). Feature descriptions with first class features have been considered in [Joh88, Tre93].

Note that for the basic feature description language FT, the literature has mainly employed a different standard interpretation (see [Smo88, Smo92]). The domain of this standard interpretation consists of *feature graphs*, which are directed, labelled graphs with edges labelled by features.

To obtain a well-investigated theory for these languages, we address the following questions in our thesis:

Are feature trees an adequate domain for feature descriptions?
More precisely, how does our feature theory for the basic feature description language FT relate to the theory we obtain if we use the feature graph interpretation of this language?

What is the expressivity of the different languages?
Here, it is interesting both to compare the languages each other and to consider whether the languages are expressive enough to encode additional concepts from the literature.

Which fragments of the feature theories are decidable?
A number of results from the literature apply to this concern. For example, it is known that the existential fragments of FT, CFT and F are decidable, and that the full theory of F is undecidable. But there is nothing known about the full theory of FT and CFT, and only partial results have been obtained for the existential fragment of RFT.

Concerning the question of the expressivity of feature description languages, only minor results can be found in the literature, and they are (with a few exceptions) stated only informally. There does not even exist a fixed definition for the expressivity of a feature description language. Using the domain of feature trees for the interpretation of the feature description languages, however, we obtain a simple definition of expressivity, using a standard first-order technique. The expressivity of a language L is defined as the set of relations on feature trees that are definable by L-formulae. An

n-ary relation R over feature trees is said to be definable in L if there is an L-formula $\phi(x_1, \ldots, x_n)$ with $x_1, \ldots, x_n$ as free variables such that the set of tuples of feature trees $(\sigma_1, \ldots, \sigma_n)$ satisfying ϕ in the standard interpretation $\mathfrak{T}_L$ of L is exactly R. Using this definition of expressivity, we follow the work of [Tre93], who showed that the languages FT and CFT are less expressive than F. Note we can use such a simple definition of expressivity since we interpret all the languages over the same domain of feature trees. This is a strong indication that the approach of defining a feature theory via a standard interpretation is fruitful and should be carried over to other languages.

1.1 Contribution of this Thesis

The main results of the thesis are the following:

- We show that the descriptive primitives of RFT are definable in F. Together with the results of [Tre93], this implies that every FT-, CFT- and RFT-formula can be translated into an equivalent F-formula. Furthermore, we show that important additional relations considered in the literature are also definable in F.

  A particularly important class of relations are those that can be described using *definite equivalences* over F. Definite equivalences over arbitrary constraint languages are introduced in [Smo93] and provide a machinery for defining all recursive relations. Given a set $\mathcal{R}$ of relation symbols that are not part of the signature of F, an equivalence

  $$R(x_1, \ldots, x_n) \leftrightarrow D(x_1, \ldots, x_n)$$

  with $R \in \mathcal{R}$ is called *definite* if D has at most $x_1, \ldots, x_n$ as free variables and D is an element of the class of all formulae generated by the production rule

  $$D, D' \quad ::= \quad R(t_1, \ldots, t_n) \mid D \wedge D' \mid D \vee D' \mid \exists x D \mid \phi,$$

  where ϕ denotes a F-formula. The following definition of a feature tree representation of lists and the corresponding append relation is an example of a set of definite equivalences:

  $$\begin{aligned} \texttt{list}(x) &\leftrightarrow x \doteq nil \\ &\vee \exists y(x\{1,2\} \wedge x[2]y \wedge \texttt{list}(y)) \\ \texttt{app}(x,y,z) &\leftrightarrow x \doteq nil \wedge y \doteq z \end{aligned}$$

$$\vee \quad \exists u, v, v'(x[1]u \wedge z[1]u \wedge x[2]v \wedge z[2]v' \wedge \mathtt{app}(v, y, v')),$$

A model of a set of definite equivalences is an interpretation of the extended language which is a conservative extension of the standard interpretation of F and which satisfies the equivalences. Note that a set of definite equivalences does not necessarily define a unique model, but there always exists a least and a greatest model (see [Smo93]). In the above example, the interpretation of `list` in the least model contains all representations of finite lists, whereas in the greatest model the representations of cyclic and infinite lists are also included.

Definite equivalences are important since they allow to define relations in the way they are defined by logic programs. A set of definite clauses P can be translated into an equivalent set of definite equivalences [Cla78, Smo93], whose least model is the model of P defined by the operational semantics of logic programming. Furthermore, type systems as used for processing modern constraint-based grammars can be translated into definite equivalences [Bac95b]. Here, both the least and the greatest model are viewed as the intended semantics.

In general, adding the concept of definite equivalences to a given language L enlarges the expressivity of L. We show in this thesis that F is expressive enough even to encode relations definable by the least model of a set of definite equivalences, and, under certain conditions, also the ones definable by the greatest model. Note that no feature description language having this expressivity was previously known. Since most of the relations used in applications are definable via definite equivalences, this implies that F is a universal feature description language.

- We present an axiomatisation of the theories of FT and CFT. Since our feature theories are complete, this implies that we inherit a decision procedure for valid FT- and CFT-formulae from predicate calculus. This is by no means trivial, since until [BS93a], no complete and decidable feature theory was known.

  Furthermore, we show that the feature graph interpretation of FT is also a model of the FT-axiomatisation. Since all models of a complete theory are elementarily equivalent, this implies that for the basic feature description language FT both the feature tree and the feature graph interpretation yield the same theory. We will show that the theories of the feature tree interpretation and the feature graph interpretation of CFT differ. As [ST94] stated, one can translate every formula ϕ in Colmerauer's rational tree constraint system RT into a CFT-formula ψ such ϕ is valid in the tree interpretation of RT (the stan-

dard interpretation of RT) iff ψ is valid in feature tree interpretation of CFT. This does not hold for the feature graph interpretation of CFT.

Our completeness proofs will exhibit simplification algorithms for the theories of FT and CFT that compute for every feature description an equivalent solved form from which the solutions of the description can be read off easily. For a closed feature description the solved form is either $\top$ (which means that the description is valid) or $\bot$ (which means that the description is invalid). For a feature description with free variables the solved form is $\bot$ if and only if the description is unsatisfiable. As a by-product, we can use the existence of the solved form to investigate the properties of the theories. Thus, we can show that FT is really less expressive than CFT, which is also a new result.

These results are partially published in [BS93a] and [Bac95a].

- We show that the satisfiability problem for conjunctions of regular path expressions is decidable (where we consider a formula satisfiable if it is satisfiable in some interpretation of RFT, not necessarily our standard feature tree interpretation). This problem has remained open for a long time. Previously, there were only a partial positive and a negative result for this problem. Kaplan and Maxwell [KM88] showed that this problem is decidable, provided that a certain acyclicity condition is met. Baader et al. [BBN+93] showed that this problem becomes undecidable if we add unrestricted negation. It has, however, remained an open problem as to whether satisfiability of conjunctions of regular path expressions is decidable in the absence of additional conditions (such as acyclicity). We will show that this is indeed decidable, and furthermore, that the feature tree model of RFT is canonical for satisfiability (i.e., a conjunction of regular path expressions is satisfiable if it is satisfiable in the standard model of RFT). This implies that the positive existential fragment of the theory of the standard interpretation of RFT is decidable.

  Even for the fragment of non-cyclic formulae, our algorithm is an improvement over the algorithm given in [KM88] since it allows for more flexible control in delaying the evaluation of complex regular path expressions. As [KM88] stated, delaying the evaluation of regular path expressions is an important method of gaining efficiency.

  This result is partially published in [Bac94].

Overview

Chapter 2 defines the domain of feature trees and some basic relations and functions on feature trees. We then define the first-order languages F, FT, CFT and RFT, and introduce the standard interpretations of these languages and the corresponding substructures consisting only of the rational feature trees. Chapter 3 investigates the expressivity of our universal feature description language F. According to our definition of expressivity, we present for every n-ary relation R over feature trees, which is encodable in F, a formula $\phi(x_1, \ldots, x_n)$ (called explicit definition for R) whose denotation in the standard interpretation is R. Interestingly, we can use the same definitions if we restrict the relations to the set of rational trees and replace the standard interpretation by its substructure consisting only of the rational feature trees. Chapter 4 presents axiomatisations of the theories of the standard interpretation of FT and CFT and proves their completeness. We show that the feature graph interpretation of FT is also a model of the axiomatisation of the theory of FT. Furthermore, we show that FT is really less expressive than CFT. Chapter 5 shows that the satisfiability problem for conjunction of regular path expressions is decidable. Furthermore, we show that the feature tree interpretation of RFT is canonical for satisfiability (i.e., a conjunction of regular path expressions of satisfiable if and only if its is satisfiable in the feature tree interpretation). Thus, the positive existential fragment of the theory of RFT is decidable.

1.2 Feature Descriptions in Constraint Programming

Feature descriptions are used in several constraint programming languages, where the main representatives are the languages LIFE [AKP93, AK93, MAK90], which is a constraint logic programming language with functions and inheritance, and Oz [HSW93, HSW95, Smo94a, Smo94c, Smo94b, Smo94d], which is a higher-order object-oriented concurrent constraint programming language. Others languages using feature descriptions are Le Fun [AKLN87, AKN89] and Login [AKN86].

The logical sublanguages of both LIFE and Oz are strongly influenced by the constraint logic programming scheme of Jaffar and Lassez [JL87]. In this scheme, Prolog's first-order constructor terms are replaced by a constraint system (i.e., a constraint language together with a class of interpretations), and the unification operation of Prolog is replaced by a satisfiability test of conjunctions of formulae in the constraint language. Höhfeld and Smolka [HS88] generalise the constraint logic programming

scheme to definite specifications over arbitrary constraint languages. If the constraint language is closed under conjunction and renaming of the variables, then both the operational and declarative semantics can be defined in a uniform way.

Oz is also influenced by Saraswat's concurrent constraint programming scheme [SR90, Sar91]. To illustrate the principle of this scheme, we define *concurrent agents* to be expressions of the form

$$\textbf{if } \phi \textbf{ then } \psi \textbf{ else } \psi',$$

where ϕ, ψ and ψ' are formulae in the constraint language. ϕ is called the guard of the agent. The agents live in a context, which itself is just another formula δ of the constraint language. For reducing an agent in some context one has to test whether the context entails or disentails the guard in the underlying constraint system. A formula δ entails a formula ϕ if in every interpretation of the constraint system, every valuation satisfying δ also satisfies ϕ. If the context δ entails the guard ϕ of the above agent, then ψ is conjoined to the context. If δ disentails ϕ (i.e., entails $\neg\phi$), then ψ' is conjoined to the context. Otherwise, the agent is suspended. It will be reawoken if the evaluation of other agents changes the context.

The main motivation for using feature descriptions in these languages is that they provide the notion of a record, familiar from programming languages such as Pascal, as a basic datatype, in a natural and flexible way. LIFE uses open records (i.e., records whose arity is not fixed), whereas Oz uses closed records with fixed arity. This is also reflected by the fact that the basic constraint language of LIFE is similar to FT, whereas Oz uses CFT as basic constraint language. Oz also incorporates extensions of CFT, but the evaluation of these additional constraints is delayed until enough information has been gathered to transform these constraints into corresponding CFT-constraints. Besides the satisfiability test on conjunctions of constraints (which is delayed in the case of Oz for efficiency reasons), the entailment test is another important operation used in these languages.

The flexibility of feature descriptions arises from the fact that they allow for partial descriptions. A good example for this flexibility is the language CFT, which combines the expressivity of FT and RT. The following examples are taken from [ST94]. Given an RT-formula σ

$$x = point(y, z),$$

we can translate σ into the following equivalent CFT-formula:

$$x\ sort\ point \wedge x\{sort, 1, 2\} \wedge x\ 1\ y \wedge x\ 2\ z.$$

But CFT has more expressive power than RT. It is possible to express within CFT

that a record has some feature without specifying others. A description of the form

$$x \ \mathit{colour}\ y$$

just states that x has a colour feature, but it does not disallow other features such as shape, size or position. If the language has a finite signature, the description above can be defined in RT by a disjunction of the form

$$x = \mathit{circle}(\ldots, y, \ldots) \vee x = \mathit{triangle}(\ldots, y, \ldots) \vee \ldots$$

enumerating all constructors for which a *colour* feature is appropriate. But the computational behaviour of this disjunction is much worse than that of the single constraint $x \ \mathit{colour}\ y$. In the case of an infinite signature (which we consider here), such a single feature constraint is not definable in RT (since it would correspond to an infinite disjunction).

Related Work

Historically, the formal foundation for feature descriptions in constraint programming languages started with the work of Hassan Aït-Kaci [AK86], who introduced the ψ-term calculus. ψ-terms are equivalence classes of feature descriptions that are closed under consistent variable renaming. Later, [Smo92] considered feature descriptions as formulae in a first-order language and introduced feature graphs as an interpretation for feature descriptions which is canonical for satisfiability (i.e., a feature description is satisfiable if it is satisfiable in the feature graph interpretation). He also provided a translation of Aït-Kaci ψ-terms into his language.

The languages FT and CFT were introduced in [AKPS94] and [ST94], respectively. Each defined the corresponding feature theory via an axiomatisation which is very similar to the axiomatisation of the standard interpretation of the language as presented in this thesis (in fact, the axiomatisation in [AKPS94] was taken from [BS93a]). But they didn't address the problem of proving completeness of their axiomatisations. And each presented a decision procedure for fragments of these theories, namely an incremental algorithm for testing simultaneously entailment and disentailment of possibly existentially quantified conjunctions of constraints. Since the class of interpretations of these languages are determined by a theory, we can reformulate the notion of entailment. A formula ϕ entails a formula ψ in some theory T (written $\phi \models_T \psi$) iff

$$T \models \tilde{\forall}(\phi \to \psi),$$

and disentails ψ iff

$$T \models \tilde{\forall}(\phi \rightarrow \neg\psi).$$

Furthermore, both prove the independence property. A theory T satisfies the independence property if for all formulae $\psi, \phi_1, \ldots, \phi_n$ in the positive existential fragment of the language of T,

$$\psi \models_T \phi_1 \vee \ldots \vee \phi_n \iff \exists i : \psi \models_T \phi_i.$$

If a theory T satisfies the independence property, then an algorithm for testing entailment and disentailment in T can be used for deciding the existential fragment of T.

Since we can decide the full theories of FT and CFT, it is clear that both the entailment and disentailment test can be performed using our simplification algorithms for these theories. But the ones in [AKPS94] and [ST94] are more efficient since they are optimised for this purpose. On the other hand, these algorithm apply only to a very limited fragment. A simple example that is not covered by these algorithms is to test whether

$$xfc_1 \wedge xgc_2$$

is entailed by

$$\exists y_1, y_2(xfy_1 \wedge xgy_2 \wedge y_1 \neq y_2).$$

This can be decided using our simplifications algorithm.

The languages FT and CFT were introduced in [AKPS94] and [ST94] with a slightly different signature. Their signatures didn't contain the constant symbols and the predicate symbol *atom*, but used sort symbols instead. Sorts are unary predicate symbols which are interpreted as disjoint sets. Both also considered feature trees, which were defined slightly differently because of the different signature. Their feature trees had labels at every node (whereas our feature trees bear label only at the leaves). The denotation of a sort symbol A in the feature tree interpretation is exactly the set of all feature trees having the root labelled with A. Their results can easily be adapted for our signature.

We changed the signature in order to gain a bit more expressivity. Sorts can be expressed in our languages by introducing a new feature *sort* and a new constant symbol for each sort symbol, and by replacing a sort constraint of the form Ax (in prefix notation) with the constraint

$$x\ sort\ A.$$

In our signature, we can express the fact that two variables share the same sort, without knowing the sort:

$$x\ sort\ x_S \wedge atom(x_S) \wedge y\ sort\ y_S \wedge atom(y_S).$$

This implies that we have sorts as first class values, which is not true of the signatures used in [AKPS94, ST94].

A complete axiomatisation for Colmerauer's rational tree system RT over an infinite signature was given in [Mah88]. Our completeness proofs for the languages FT and CFT have the same overall structure used in [Mah88]. However, Maher's proof depends heavily on the structure of first-order terms, since it uses substitutions. This is not appropriate in our case since we are using a relational language. A complete axiomatisation for RT over a finite signature is given in [Mah88, CL89].

A different completeness proof for CFT is presented in [BT94], where Ehrenfeucht-Fraïssé games are used. The method is semantic, in showing that all models of CFT are elementarily equivalent (i.e., make the same sentences valid), which immediately implies that CFT is complete. This yields a trivial decision method for CFT-sentences, by enumerating all consequences of CFT. Given an arbitrary sentence ϕ, the enumeration will produce either ϕ or $\neg\phi$ since CFT is complete. On the other hand, this thesis employs a proof theoretic method in showing explicitly that for every sentence ϕ, either ϕ or $\neg\phi$ is valid in CFT. Both methods have their merits. The proof in [BT94] is shorter (though similar problems arise in handling inequations), while the proof in this thesis presents a decision method for validity.

Another closely related work is the one by Treinen [Tre93], who introduced the languages F (again with sort symbols instead of constants) and EF, which is F extended with arity constraints as used in CFT. Treinen also defined the standard interpretation of feature trees for these languages. Since arity constraints are definable in F, the expressivity of F and EF is the same. But this is only true if we consider the full first-order theory. For the existential fragment, these theories clearly differ. Treinen showed that the existential fragment of the theory of the feature tree interpretation of EF is decidable. Furthermore, he proved that the full theory of F is undecidable. In contrast with our work, Treinen was not concerned with showing that F is a universal feature description language.

1.3 Feature Descriptions and Constraint-Based Grammars

In the last decade, a family of grammar formalisms has become popular which is subsumed under the term *constraint-based grammar formalisms*. These formalisms have in common that they use feature descriptions for modelling linguistic entities such as words, phrases and sentences. Some of the most widely used grammar models in formal theoretical linguistics such as LFG [KB82], FTAG [VSJ88] and HPSG [PS87] employ constraint-based formalisms. One of the main advantages of such formalisms is that they provide a *declarative* representation of linguistic knowledge, i.e. the linguistic knowledge can be stated independently from the way it is processed. This has important impact on grammar engineering in computational linguistics. For example, the time for developing a sizeable grammar in these formalisms is usually much less than in other formalisms. Because of the declarative semantics of the formalism, constraint-based grammars exhibit a higher potential for reusability (see for example [RJ94]). For this reason, European-Union-funded projects involving grammar development have adopted constraint-based grammar formalisms [Eur94].

The motivation for using feature descriptions as representation formalism in constraint-based grammar formalism is stated in [PS87, page 7]:

> "In all these formalisms and theories, linguistic objects are analysed in terms of *partial information structures* which mutually constrain possible collections of phonological structure, syntactic structure, semantic content and contextual factors in actual linguistic situations. Such objects are in essence data structures which specify values for attributes; their capability to bear information of non-trivial complexity arises from their potential for recursive embedding ... and structure-sharing ...."

To summarise, important attributes of feature descriptions are declarativity, partiality and the capability of describing nested structured objects. They allow for structure sharing via coreferences and a uniform representation of different levels of linguistic knowledge.

In the following, we give a detailed example of the use of feature descriptions in constraint-based grammars. We start with an example of annotated context-free rules (phrase structure rules) as used in the PATR-II [SUP$^+$83, Shi89, Shi92] or LFG [KB82] formalisms. These annotations further restrict the set of derivations that are licenced by some grammar rule. Figure 1.3 shows a small grammar for

(R_1) $S \rightarrow NP\ \ VP$
$x_{NP}\ agr\ z \wedge x_{VP}\ agr\ z$

(R_2) $VP \rightarrow V\ \ NP$
$x_{VP}\ agr\ z \wedge x_V\ agr\ z$

(R_3) $V \rightarrow loves$
$x_V\ agr\ z \wedge\ z\ pers\ sg\ \wedge\ z\ num\ 3rd,$

(R_4) $NP \rightarrow Mary$
$x_{NP}\ agr\ z \wedge\ z\ pers\ sg\ \wedge\ z\ num\ 3rd,$

(R_5) $NP \rightarrow John$
$x_{NP}\ agr\ z \wedge\ z\ pers\ sg\ \wedge\ z\ num\ 3rd,$

Figure 1.3: A small grammar

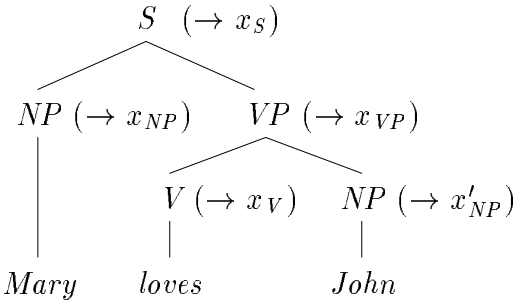
English. A rule of the form

$$S \rightarrow NP\ \ VP$$
$$x_{NP}\ agr\ z \wedge x_{VP}\ agr\ z$$

consists of the context-free rule $S \rightarrow NP\ \ VP$, which is annotated by the feature description $x_{NP}\ agr\ z \wedge x_{VP}\ agr\ z$, where x_{NP}, x_{VP} and z are logical variables.

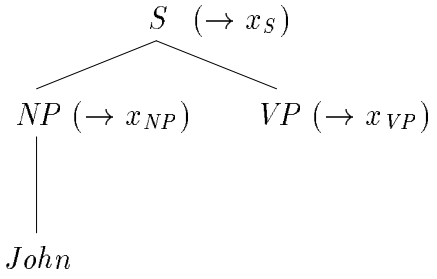
Derivations are described over annotated phrase structures. A phrase structure is a labelled, ordered tree. An annotated phrase structure consists of a phrase structure, a feature description and an association of the non-terminal nodes of the phrase structure with the free variables in the feature descriptions. We indicate that some non-terminal node of a phrase structure is associated with the variable x by writing

$(\rightarrow x)$ to the right of this node. An example of an annotated phrase structure is



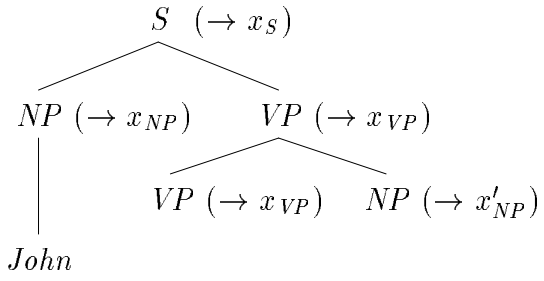
$$x_{VP}\ agr\ z\ \wedge\ x_V\ agr\ z\ \wedge\ x_{NP}\ agr\ z$$
$$\wedge\ z\ pers\ sg\ \wedge\ z\ num\ 3rd$$
$$\wedge\ x'_{NP}\ agr\ z'$$
$$\wedge\ z'\ pers\ sg\ \wedge\ z'\ num\ 3rd.$$

Here, *Mary*, *loves* and *John* are the terminal nodes. When a rule is applied to some leaf of an annotated phrase structure, then the phrase structure is expanded according to the context-free part of the grammar rule. The variable on the left-hand side of the rule is identified with the variable associated with the expanded node, and all other variables of this feature description are consistently renamed to new variables. The resulting feature description is added conjunctively to the feature description of the initial annotated phrase structure. Thus, applying the rule (R_2) to



$$x_{NP}\ agr\ z\ \wedge\ x_{VP}\ agr\ z$$
$$\wedge\ z\ pers\ sg\ \wedge\ z\ num\ 3rd$$

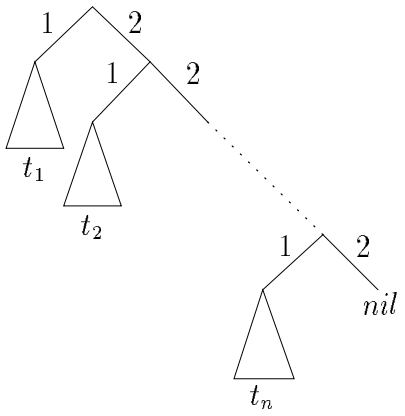
yields



$$x_{NP}\ agr\ z\ \wedge\ x_{VP}\ agr\ z$$
$$\wedge\ z\ pers\ sg\ \wedge\ z\ num\ 3rd$$
$$\wedge\ x_{VP}\ agr\ z'\ \wedge\ x_V\ agr\ z'.$$

Note that the feature description associated with the result can be simplified to $x_{NP}\ agr\ z \wedge x_{VP}\ agr\ z \wedge x_V\ agr\ z \wedge z\ pers\ sg \wedge z\ num\ 3rd$. An annotated phrase structure is licenced by a grammar if it can be generated by applying the grammar rules as described above *and* if the associated feature description is satisfiable.

In some modern grammar formalisms, which are influenced by the work on of the grammar theory HPSG (Pollard and Sag [PS87, PS94]), a more radical approach is taken by uniformly representing all linguistic data (including the phrase structure) within feature descriptions. Since phrase structures are ordered trees, the ordering information must be represented explicity using lists of feature descriptions. A list of feature trees $(t_1, \ldots, t_n)$ can be represented as a feature tree using the features 1 and 2 and the atom *nil*:

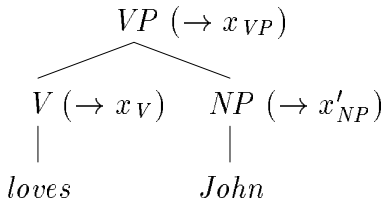


(1.1)

The clumsiness of list description using this encoding can easily be avoided using some syntactic sugar. Thus, we write $y = \langle x_1, \ldots, x_n \rangle$ for

$$\exists y_1 \ldots y_n (y \doteq y_1 \ \wedge \ y_n \ 2 \ \mathit{nil} \ \wedge \ \bigwedge_{i=1}^{n-1} y_i \ 1 \ x_i \ \wedge \ y_i \ 2 \ y_{i+1})$$

Using this representation of list, we can encode the information contained in an annotated phrase structure within feature descriptions. Here, we use the feature *syn* to denote the non-terminal symbol associated with the phrase structure, the feature *phon* to denote the string of terminals covered by the phrase structure, and the feature *dtrs* to list the phrase structures which occur directly under the root. Using this encoding, the annotated phrase structure



$$\begin{aligned} & x_{VP}\ \mathit{agr}\ z \ \wedge \ x_V\ \mathit{agr}\ z \\ & \wedge\ z\ \mathit{pers}\ \mathit{sg} \ \wedge \ z\ \mathit{num}\ \mathit{3rd} \\ & \wedge\ x'_{NP}\ \mathit{agr}\ z' \\ & \wedge\ z'\ \mathit{pers}\ \mathit{sg} \ \wedge \ z'\ \mathit{num}\ \mathit{3rd}. \end{aligned}$$

is translated into the following feature description (where we use matrix notation for

better readability):

$$\left[\begin{array}{lcl} syn & : & VP \\ phon & : & \langle loves\ John\rangle \\ agr & : & z = \left[\begin{array}{lcl} pers & : & sg \\ num & : & 3rd \end{array}\right] \\ dtrs & : & \left\langle \left[\begin{array}{lcl} syn & : & V \\ phon & : & \langle loves\rangle \\ agr & : & z \end{array}\right] , \left[\begin{array}{lcl} syn & : & NP \\ phon & : & \langle John\rangle \\ agr & : & \left[\begin{array}{lcl} pers & : & sg \\ num & : & 3rd \end{array}\right] \end{array}\right] \right\rangle \end{array}\right]$$

After the encoding of annotated phrase structure is established, the question arises as to how to transform the grammar listed in Figure 1.3. First note that we can read a grammar rule such as (R_1) as a recipe for building an annotated phrase structure with top-symbol VP, given phrase structures for a V and an NP. Conversely, we know that every phrase structure labelled with VP must have a V followed by an NP since there is only one rule having VP at the lefthand side. This implies that the set of feature trees satisfying the feature descriptions of a verb phrase R_{VP} can be described by the formula

$$\begin{array}{rcl} R_{VP}(x) & \leftrightarrow & x\ syn\ VP \\ & \wedge & \exists y, z(\,\exists z'(x\ dtrs\ z' \wedge z' = \langle y, z\rangle) \wedge \\ & & \quad R_V(y) \wedge R_{NP}(z) \wedge \exists u(x\ agr\ u \wedge y\ agr\ u) \\ & & \quad \exists u_1, u_2, u_3(x\ phon\ u_1 \wedge y\ phon\ u_2 \wedge z\ phon\ u_3 \wedge \mathtt{app}(u_2, u_3, u_1))), \end{array} \tag{1.2}$$

where R_V and R_{NP} are unary predicates denoting the feature trees representing Vs and NPs, respectively; VP is an atom symbol; and `app` is the append relation on the feature tree representation of lists as defined on page 5. Note that the definition for R_{VP} is a definite equivalence.

Using this formalisation of grammatical rules, parsing (and if we had added semantics in our examples also generation) can ideally be described as a pure deductive process [Per83]. A sentence $word_1 \ldots word_n$ is licenced by the grammar G if

$$\exists x, y(R_S(x) \wedge x\ phon\ y \wedge y = \langle word_1\ \ldots\ word_n\rangle)$$

is valid in all models of G (or valid in some model of G, depending on the intended semantics of grammars).

In representations of grammatical rules (and/or grammatical principles; for a detailed discussion on the difference of grammatical rules and grammatical principles

see [PS87]) like (1.2), the relation symbols representing linguistic entities (such as R_{VP} etc.) are always unary predicates. Now there are quite a number of formalisations that are concerned with feature description languages that contain unary predicate symbols in their signatures. In these formalisations, the unary predicate symbols are called *types*, and the languages are called typed feature description languages. For example, both FT and CFT were introduced in [AKPS94] and [ST94] as typed feature description languages (where the unary predicate symbols were called sorts instead of types). But as the discussion in section 1.2 shows, these types can be simulated using constants, and our versions of FT and CFT are even more expressive.

Based on typed feature descriptions, the concept of type systems played an important in the literature on constraint-based grammar formalisms ([AK86, EZ90b, EZ90a, Pol89, PM90, Smo92, AKPG93, Car92, Zaj92, KS94]) since they allow for a direct encoding of grammatical principles as defined in the grammar theory HPSG [PS87, PS94]. A type system consists of a partial order on the type symbols defining *type inheritance* and a set of type definitions of the form

$$T(x) := \phi(x),$$

where T is a type symbol and ϕ is a typed feature description. The ordering is interpreted as the subset relation on the sets denoted by the types, and a type definition $T(x) := \phi(x)$ restricts the denotation of T to the set of all elements x satisfying $\phi(x)$. But as [Bac95b] shows, type systems can be translated into definite equivalences, and the intended interpretation for type systems is the greatest model of the resulting set of definite equivalences. Retrospectively, one can say that the reason for concentrating on the unary predicate symbols (types) is that the unary predicates model classes of linguistic entities (which are the objects of main interest in constraint-based grammars). This focus might have been the reason that type systems and definite equivalences have often been considered as different concepts. For a detailed discussion of type systems, the interested reader is referred to [Kri95].

Many modern implementation of constraint-based grammar formalisms use type systems. Furthermore, nearly all implementations provide a mechanism whose semantics can be described in terms of definite equivalences. In the following, we consider the systems ALE [Car92, Car94], TFS [Zaj92], CUF [DE91, DD93] and $\mathcal{TDL}/\mathcal{UDiNe}$ [BK93, KS94], which is a representative selection of advanced constraint-based grammar formalisms (descriptions of these and other implemented systems can be found in [BKSU93]). Both $\mathcal{TDL}/\mathcal{UDiNe}$ and TFS have type systems that allow for arbitrary type definitions, and grammatical principles can be defined via type definitions. In TFS, the constraint solver is built in and parsing (or generation) must be performed with the deductive system provided with the type system. Since in

$\mathcal{TDL}/\mathcal{UDiNe}$ the constraint-solver $\mathcal{UDiNe}$ is a separate component, it can also be used for efficiency reason together with a separated parser.

In ALE and CUF, the type systems allow only a restricted form of type definition, which is not sufficient for defining grammatical principles directly. Both systems instead provide a mechanism called definite clauses over feature description languages. The mechanism was introduced in [HS88] and generalises the constraint logic programming scheme of Jaffar and Lassez [JL87]. Definite clauses can be translated into a set of definite equivalences, where the intended semantics is the least model of the resulting set of definite equivalences [Smo93]. In CUF, these definite clauses can be used directly for defining grammatical principles, while this is not possible in ALE. The operational semantics used for the definite clauses in ALE is adapted from PROLOG, which is not the appropriate one for grammatical principles [Man93a]. For this reasons, ALE has a built in parser for annotated context-free rules, in which grammatical principles can be specified.

Roughly speaking, the feature description languages used in the above-mentioned systems are syntactic variants of FT. Clearly, all systems handle the positive existential fragment of FT. Furthermore, negated equations are handled by ALE, $\mathcal{TDL}/\mathcal{UDiNe}$ and CUF. $\mathcal{UDiNe}$ is (to our knowledge) the only implemented feature constraint solver which also handles negation of feature description as defined in [Smo92]. Given a conjunction of constraints ϕ and a distinguished variable x which is free in ϕ, the negation of ϕ with respect to the variable x is defined as

$$\psi = \neg\exists(X - \{x\})\phi,$$

where X is the set of free variable of ϕ. $\mathcal{UDiNe}$ additionally uses a syntactic variant of disjunction which appears in the literature under the name *distributed disjunction* ([Bac89, BEG90, DE89, DE90, MK89]). Distributed disjunction are disjunctions which bear an additional tag, a name. An example of a feature description with distributed disjunctions is

$$\phi = \begin{bmatrix} l1 & : & \{_{d_1}\ +, - \} \\ l2 & : & \{_{d_1}\ 1, 2 \} \end{bmatrix}$$

This feature description is equivalent to the disjunction of

$$\begin{bmatrix} l1 & : & + \\ l2 & : & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} l1 & : & - \\ l2 & : & 2 \end{bmatrix}$$

Note that the combinations

$$\begin{bmatrix} l1 & : & + \\ l2 & : & 2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} l1 & : & - \\ l2 & : & 1 \end{bmatrix}$$

are not in the interpretation of ϕ. The advantages of distributed disjunctions are that they allow for a very compact encoding of linguistic data and that they often avoid expansion to disjunctive normal form during unification.

Related Work

There have been many different and diverging formalisations for feature descriptions. Even for the basic feature description language which contains (roughly speaking) the descriptive primitives of FT, there have been many different approaches besides the predicate logic ones of [Joh88, Smo92]. Examples include the early work by Kasper and Rounds [RK86, KR86, KR90] using a non-standard logic, and the multi-modal logic approaches by [Rea91, BS93b] where every feature corresponds to a modal operator. At least these approaches are comparable to each other. But for the extensions to feature descriptions there are even more divergent approaches. Most of the formalisations were custom-built, and nearly every time a new extension to feature descriptions was proposed, a new formalisation was presented.

There was only one approach to building a more general feature theory (i.e., encoding different extensions in one single feature theory), namely that of Johnson [Joh91, Joh94], who used Schönfinkel-Bernays' formulae for the formalisation of feature descriptions. Schönfinkel-Bernays' formulae are of the form

$$\exists x_1, \ldots, x_n \forall y_1, \ldots, y_m \phi$$

where ϕ is a quantifier-free formula. For this class of formulae, the satisfiability problem is known to be decidable. But this approach is restricted to decidable extensions of feature descriptions, while we also want to encode undecidable extensions such as definite equivalences. Furthermore, regular path expression and subsumption constraints are not expressible within the Schönfinkel-Bernays' fragment (see [Joh94, page 10]), although they are expressible in the feature description language F. Hence, no other feature description language presented in the literature has an expressivity comparable to F. For this reason, we call F a universal feature description language.

The notion of completeness as defined for FT and CFT is different from the notion of completeness considered in related work by Kasper and Rounds [KR90] and Moss [Mos92]. These authors study logical equivalence for rooted and quantifier-free feature descriptions and give complete equational axiomatisations of the respective congruence relations. In contrast, we are concerned with a much larger class of possibly quantified feature descriptions. Moreover, exploiting the power of predicate logic, we are not committed to any particular model or any particular deductive system,

but instead prove a result that implies that any complete proof system for Predicate Logic will be complete for proving equivalence of feature descriptions with respect to any model of our feature theories for FT and CFT.

The complexity of the simplification algorithms is too high for use in a grammar formalism. But one could use the algorithms to decide general properties of a grammar, where the complexity is not relevant. For example, one can check with our simplification algorithms whether a grammatical rule is superfluous by virtue of being subsumed by another rule. For example, consider the two annotated context-free rules

$$(R_1) \qquad \begin{array}{l} NT \rightarrow \; NT_1 \; \ldots \; NT_n \\ \qquad\qquad \phi_1 \end{array}$$

and

$$(R_2) \qquad \begin{array}{l} NT \rightarrow \; NT_1 \; \ldots \; NT_n \\ \qquad\qquad \phi_2 \end{array}$$

where $NT, NT_1, \ \ldots \ , NT_n$ are non-terminal symbols and ϕ_1 and ϕ_2 are the annotated feature descriptions. If ϕ_1 entails ϕ_2 (i.e., every valuation of satisfying ϕ_1 also satisfies ϕ_2), then R_1 is superfluous and can be removed, since in every derivation the application of rule R_1 can be replaced by an application of rule R_2. We have used annotated context-free rules in this example for reasons of simplicity, but a similar test can be performed for formalisms where the phrase structure is encoded in feature descriptions. Now if the feature descriptions make use of negated equations as in the example given in section 1.2, page 11, the known algorithm for testing entailment in FT and CFT (see [AKPS94] and [ST94], respectively) cannot be used. The use of such negated equation for constraint-based grammars is, for example, considered in [Car92].

As mentioned, a partial result for the satisfiability problem of conjunction of regular path constraints was obtained in [KM88]. They showed that the satisfiability problem for conjunctive formulae containing regular path expression is decidable if an acyclicity condition is met. But it cannot be guaranteed that the acyclicity condition is maintained during the application of their algorithm for testing satisfiability.

Solving the satisfiability problem for cyclic descriptions containing regular path expressions requires a non-trivial extension of the algorithm described in [KM88]. Their algorithm uses a set of simplification rules that transforms a feature description into a normal form, from which satisfiability can be read off trivially. If the acyclicity condition is met, the rule system is terminating. But in the case of cyclic descriptions, termination cannot be guaranteed anymore. This is inherent to the problem. We solved the problem in this thesis by introducing a quasi-terminating rule system

(see [Der87]). A rule system is quasi-terminating if it is not terminating, but produces only finitely many different results. To achieve a quasi-terminating rule system we had to translate the problem into a new syntax that enabled us to delay subproblems whose evaluation would cause an infinite number of results.

To see some possible application of regular path expressions, we briefly recall an example that is given in Kaplan and Maxwell [KM88, page 1]. Consider the topicalized sentence

Mary, John telephoned yesterday.

Using s as a variable denoting the whole sentence, the LFG-like clause

$$s\ topic\ x \wedge s\ obj\ x$$

specifies that in s, *Mary* should be interpreted as the object of the relation *telephoned*. The sentence could be extended by introducing additional complement predicates, e.g. in sentences like *Mary, John claimed that Bill telephoned*; *Mary, John claimed that Bill said that ... Henry telephoned yesterday*; .... For this family of sentences the clauses $s\ topic\ x \wedge s\ (comp\ obj)\ x$, $s\ topic\ x \wedge s\ (comp\ comp\ obj)\ x$ and so on would be appropriate; specifying all possibilities would yield an infinite disjunction. This changes if we make use of regular path expressions, allowing the above to be specified as the single clause

$$s\ topic\ x \wedge s\ comp^*\ obj\ x.$$

Chapter 2

First-order Languages over Feature Trees

In this chapter, we define feature trees and some basic functions and relations on feature trees. We then introduce various first-order languages, consider the corresponding feature tree interpretations and their substructures consisting only of the rational trees. We close this chapter by discussing some properties of these languages. Furthermore, we summarise the decidability and undecidability results that either can be taken out of the literature or will be proven in this thesis.

2.1 Basic Definitions for Feature Trees

Throughout the thesis we assume a fixed, countable infinite set $\mathcal{L}$ of **labels**, over which the set of feature trees is defined.[1] The labels are used for labelling both the edges and the leaves of feature trees. A finite string $p \in \mathcal{L}^*$ of labels is called a **path**, where ϵ denotes the empty path. A **tree domain** is a nonempty set $D \subseteq \mathcal{L}^*$ of paths that is **prefix-closed**, i.e., if $pq \in D$, then $p \in D$. Note that every tree domain contains the empty path ϵ. Given a tree domain D, we define $\mathit{leaves}(D)$ to be the set of maximal paths of D, i.e.,

$$\mathit{leaves}(D) \;:=\; \{p \in \mathcal{L}^* \mid \forall f \in \mathcal{L} : pf \notin D\}$$

Definition 2.1 (Feature Tree) *A* feature tree *is a pair $\sigma = (D, \lambda)$, where D is a tree domain and λ is a total function $\lambda\colon \mathit{leaves}(D) \to \mathcal{L}$.*

[1] In the rest of the thesis, we assume that $\mathcal{L}$ contains the following symbols: 0, 1, 2, *nil* and ϵ.

The paths in D represent the nodes of the tree; the empty path represents its root; and λ represents the labelling of the leaves of σ. The letters σ and τ will always denote feature trees. The set of all feature trees is denoted by $\mathcal{T}$. For convenience, we will identify the primitive feature trees of the form $(\{\epsilon\}, \{(\epsilon, a)\})$ with the symbol a itself. These feature trees are called **atoms**.

A feature tree $\sigma = (D, \lambda)$ is called **finite [infinite]** if its domain D is finite [infinite]. A label $f \in \mathcal{L}$ is called a **feature** in some feature tree $\sigma = (D, \lambda)$ if it labels some edge of σ, i.e., there is some path $p \in \mathcal{L}^*$ such that $pf \in D$. The set of all features of a tree σ is denoted by $\mathit{features}(\sigma)$.

Our definition of feature trees differs slightly from the definition given in [AKPS94, BS93a]. In our case, only the leaves of feature trees are labelled, whereas the feature trees in [AKPS94, BS93a] have labels at every node (called sorts). This form of feature trees can easily be simulated in our setting using a special feature *sort*.

The **subtree** $p^{-1}\sigma$ of a feature tree $\sigma = (D, \lambda)$ **at** a path $p \in D$ is the feature tree (D', λ') defined by

$$D' = \{q \mid pq \in D\} \quad \text{and} \quad \lambda' = \{(q, a) \mid (pq, a) \in \lambda\}.$$

If it is convenient, we will also sometimes write $\mathit{subtreeAt}(\sigma, p, \tau)$ if $p^{-1}\sigma = \tau$. A feature tree τ is called a **subtree** of a feature tree σ (written $\mathit{subtree}(\sigma, \tau)$) if σ is a subtree of τ at some path $p \in D$, and a **direct subtree** if in addition $p = f$ for some feature f.

Definition 2.2 (Rational Feature Tree) *A feature tree $\sigma = (D, \lambda)$ is called* rational *if*

1. *σ has only finitely many distinct subtrees, and*
2. *σ is finitely branching (i.e., for every $p \in D$, the set $\{f \in \mathcal{L} \mid pf \in D\}$ is finite).*

Note that for every rational feature tree $\sigma = (D, \lambda)$ there exist finitely many features $f_1, \ldots, f_n$ such that $D \subseteq \{f_1, \ldots, f_n\}^*$.

A very important notion is the one of the **arity** of a feature tree, which is the set of features that occur directly under the root of a feature tree. We write $\mathit{arity}(\sigma) = F$ if $F \subseteq \mathcal{L}$ is the arity of σ. Clearly, the arity of a feature tree σ is the intersection of the tree domain of σ with the set of labels $\mathcal{L}$. All atoms have the arity $\emptyset$, and conversely,

a feature tree with arity $\emptyset$ is an atom. Note that if two feature trees have the same non-empty arity and the same subtrees at the corresponding features, then they are equal. This does not hold for the empty arity, since all atoms have the arity $\emptyset$.

The following functions and partial orders have been considered in the literature on feature descriptions. Although we will not make use of them in the rest of this thesis (except that we are showing that these operations can be defined in our universal feature theory), we define them here since they are natural concepts, and since they have important applications.

The first function is *adjoinAt*. It is used for adjoining a feature tree τ to a given feature tree σ at some feature f. The resulting feature tree $\sigma' = adjoinAt(\sigma, f, \tau)$ has the feature f defined under its root, and the subtree of σ' at f is τ. Except f, σ' has the same features as σ, and the same subtrees at the corresponding features. Thus, $adjoinAt(\sigma, f, \tau)$ replaces the subtree of σ at f by τ if $f \in arity(\sigma)$, and adds the feature f with τ as the corresponding subtree if $f \notin arity(\sigma)$. Note that adjoining a feature to an atom implies that the label of the atom gets lost.

This function was introduced in [HSW95, Smo94b] in the context of the Oz-system (for the description of the Oz-language and the underlying concepts see also [HSW93, Smo94a, Smo94c, Smo94d]), where an important application of *adjoinAt* is inheritance of objects. In Oz, the method table of an object is represented as a record (where the method names are the features, and the values are the actual methods). If an object inherits from another object, then the method table of the inheriting object is derived from the method table of the parent object. If a method name is new, then this name is added to the table. Otherwise the corresponding method is overwritten. This behaviour is exactly modelled by the *adjoinAt* function. Since the method tables cannot be extracted from the object definitions but are computed at runtime, we cannot make any assumptions about the changed record. Again, this holds for the *adjoinAt* function.

From the definition we get immediately that the effect of applying *adjoinAt* several times using the same feature depends only on the last application of *adjoinAt*, i.e.,

$$adjoinAt(adjoinAt(\sigma, f, \tau), f, \tau') \quad = \quad adjoinAt(\sigma, f, \tau').$$

On the other hand, if we adjoin under different features f and g, then the application of *adjoinAt* is order independent[2], i.e.,

$$adjoinAt(adjoinAt(\sigma, f, \tau), g, \tau') \quad = \quad adjoinAt(adjoinAt(\sigma, g, \tau'), f, \tau).$$

[2]This notion was introduced by [NP93], where a ternary relation on so-called *multitrees* similar to *adjoinAt* was introduced. Multitrees differ from feature trees in that a node can have several outgoing edges labelled with the same feature.

Furthermore, adjoining the same feature tree at the same feature to different atoms $a \neq b$ produces the same result:

$$adjoinAt(a, f, \tau) \quad = \quad adjoinAt(b, f, \tau).$$

The next two relations are partial orders on feature tree. These orders have been investigated in detail in [DR89, DR92, Dör93b]. The first order *simulate* is just the subset relation on feature trees:

$$simulate(\sigma, \tau) \quad \text{iff} \quad D_\sigma \subseteq D_\tau \quad \text{and} \; \lambda_\sigma \subseteq \lambda_\tau$$

The second order *subsume* corresponds directly to tree embedding. For this purpose, we have to define the notion of an endomorphism. An **endomorphism** on the set of all feature trees $\mathcal{T}$ is partial map $\gamma : \mathcal{T} \rightsquigarrow \mathcal{T}$ such that the following holds:

- $\gamma(c) = c$ for every $c \in \mathcal{L} \cap \mathrm{dom}(\gamma)$.
- for every feature tree $\sigma \in \mathrm{dom}(\gamma)$ and every feature $f \in \mathcal{L}$, if $f^{-1}\sigma$ is defined, then $\gamma(f^{-1}\sigma)$ is defined and

$$\gamma(f^{-1}\sigma) = f^{-1}\gamma(\sigma).$$

A feature tree σ is said to be **subsumed** by a feature tree τ (written $subsume(\sigma, \tau)$ if there is an endomorphism on $\mathcal{T}$ that maps σ to τ.

There exists an alternative definition of the *subsume* relation, namely

$$\begin{aligned} subsume(\sigma, \tau) \quad \text{iff} \quad & simulate(\sigma, \tau) \; \text{and} \\ & \forall \text{ paths } p, q \in D_\sigma : (p^{-1}\sigma = q^{-1}\sigma \;\Rightarrow\; p^{-1}\tau = q^{-1}\tau). \end{aligned}$$

This definition reflects more the original motivation for introducing this relation, namely to order feature trees due to their "information content". The notion of "information content" is strongly related to the expressive means of the core language for feature descriptions, where the existence of specific paths, the labelling of some paths as atom and the equality of subtrees can be specified. The relation was initially introduced by Shieber [Shi89, Shi92]. There, it was shown that this relation has an application in computational linguistics in solving the coordination problem. Furthermore, Shieber showed that type inference in a programming language with polymorphic types can be described adequately using the subsumption relation. Unfortunately, [DR89, DR92] showed that the satisfiability problem for feature descriptions that use the subsumption relation as a descriptional primitive is undecidable. Therefore, Dörre [Dör93b] argued that extending feature descriptions with a descriptional primitive expressing the *simulate* relation suffices for most applications. This extension has a decidable satisfiability problem [Dör93b].

2.2 The Languages F′, FT′, CFT′, and RFT

In this section, we define various first-order languages with equality whose properties will be considered in the remaining parts of this thesis. All of these languages are purely relational, i.e., the signatures do not contain proper function symbols. We assume an infinite supply of variables ranging over $x, y, z, \ldots$ Since the languages are purely relational, every term is either a constant or a variable. We use the letters $t, t', \ldots$ to denote terms. For all languages we introduce two interpretations, namely the feature tree structure, whose domain is the set of all feature trees (defined over the set $\mathcal{L}$ of labels), and its restriction to rational feature trees. The languages F′, FT′ and CFT′ have been introduced in a slightly modified version in [Tre93], [AKPS94] and [ST94]. These papers used unary predicates called sorts and didn't have constant symbols. To make a clear distinction between our languages and the languages defined by these papers, we do not use the original names F, FT and CFT.

2.2.1 Definition of the Languages

The Language F′

The signature of F′ consists of

- all elements of $\mathcal{L}$ acting as constant symbols, and
- a ternary predicate symbol $\cdot[\cdot]\cdot$

We use mixfix notation $t[t']t''$ for the so-called **generalised feature constraint**. The **feature tree structure** $\mathfrak{T}_{\mathrm{F}'}$ is the F′-structure defined as follows:

- the universe $\mathbf{U}(\mathfrak{T}_{\mathrm{F}'})$ of $\mathfrak{T}_{\mathrm{F}'}$ is the set of all feature trees over $\mathcal{L}$,
- $c^{\mathfrak{T}_{\mathrm{F}'}} = (\{\epsilon\}, \{(\epsilon, c)\})$ for every constant symbol $c \in \mathcal{L}$, and
- $(\sigma_1, \sigma_2, \sigma_3) \in \cdot[\cdot]\cdot^{\mathfrak{T}_{\mathrm{F}'}}$ if and only if there is a constant symbol c such that $\sigma_2 = c^{\mathfrak{T}_{\mathrm{F}'}}$ and $c^{-1}\sigma_1 = \sigma_3$.

The Language FT′

The signature of FT′ consists of

- all elements of $\mathcal{L}$, which act both as constant symbols and binary predicate symbols (called **features**), and
- a unary predicate symbol *atom*.

We use the letters $f, g, h, \ldots$ for the feature symbols, and use infix notation tft' for the so-called **feature constraints**. The **feature tree structure** $\mathfrak{T}_{\mathrm{FT}'}$ is the following FT′-structure:

- the universe $\mathbf{U}(\mathfrak{T}_{\mathrm{FT}'})$ of $\mathfrak{T}_{\mathrm{FT}'}$ is the set of all feature trees over $\mathcal{L}$,
- $c^{\mathfrak{T}_{\mathrm{FT}'}} = (\{\epsilon\}, \{(\epsilon, c)\})$ for every constant symbol $c \in \mathcal{L}$,
- for every feature $f \in \mathcal{L}$: $(\sigma, \tau) \in f^{\mathfrak{T}_{\mathrm{FT}'}}$ iff $f \in \mathit{arity}(\sigma)$ and $\tau = f^{-1}\sigma$,
- $\sigma \in \mathit{atom}^{\mathfrak{T}_{\mathrm{FT}'}}$ iff σ is an atom.

The Language CFT′

The signature of CFT′ is the signature of FT′ extended by

- a unary predicate symbol for every non-empty, finite set $F \subseteq \mathcal{L}$ of features (called **arity**).

We use postfix notation xF for the so-called **arity constraints**. The **feature tree structure** $\mathfrak{T}_{\mathrm{CFT}'}$ is the following CFT′-structure:

- the universe of $\mathfrak{T}_{\mathrm{CFT}'}$ and the interpretations of *atom*, the constant symbols and the feature symbols are defined as in $\mathfrak{T}_{\mathrm{FT}'}$, and
- $\sigma \in F^{\mathfrak{T}_{\mathrm{CFT}'}}$ iff $\mathit{arity}(\sigma) = F$.

Note that the signature of CFT′ contains no arity constraint for the empty arity. The reason is that the interpretation of the empty arity is the same as the interpretation of *atom*. We use in $\mathit{atom}(x)$ instead of $x\emptyset$ since (1) we want FT′ to be a subsignature of CFT′, and (2) the behaviour of the empty arity is different from the non-empty arities; for example, for a non-empty arity $F = \{f_1, \ldots, f_n\}$ we have

$$\mathfrak{T}_{\mathrm{F}'} \models \forall y_1, \ldots, y_n \exists! x (xF \wedge \bigwedge_{i=1}^{n} x f_n y_n),$$

whereas there are infinitely many elements in $\mathbf{U}(\mathfrak{T}_{\mathrm{F}'})$ with the empty arity.

The Language RFT

For the definition of this language we need to extend the notion of regular expressions over a finite alphabet to an infinite alphabet. The formation rule for regular expressions over the alphabet $\mathcal{L}$ is given by

$$L, L_1, L_2 \quad ::= \quad \emptyset \mid \epsilon \mid F \mid \overline{F} \mid L_1 \cup L_1 \mid L_1 \circ L_2 \mid L^*$$

where $F \subseteq \mathcal{L}$ is a finite set of labels. This definition extends the standard definition of regular expressions by allowing expressions of the form $\overline{F}$, which are called **cofinite sets**. The regular set $[\![L]\!] \subseteq \mathcal{L}^*$ denoted by a regular expression L is defined inductively as

$$\begin{aligned}
&[\![\emptyset]\!] = \emptyset, \quad [\![F]\!] = F, \quad [\![\overline{F}]\!] = \mathcal{L} \backslash F, \\
&[\![L_1 \cup L_2]\!] = [\![L_1]\!] \cup [\![L_2]\!], \quad [\![L_1 \circ L_2]\!] = \{pp' \mid p \in [\![L_1]\!] \wedge p' \in [\![L_2]\!]\}, \\
&[\![L^*]\!] = \{p_1 \ldots p_n \in \mathcal{L}^* \mid n \geq 0, p_i \in [\![L]\!] \text{ for } i \in 1 \ldots n\}
\end{aligned}$$

A set $S \subseteq \mathcal{L}^*$ is called **regular** if there is some regular expressions L such that $S = [\![L]\!]$. Since the denotation of $\overline{\emptyset}$ is $\mathcal{L}$, we will just use $\mathcal{L}$ as syntactic sugar for $\overline{\emptyset}$. If $F = \{f\}$ is a set containing only one feature, then we use f as short for $\{f\}$. Similarly, we write $f_1 \ldots f_n$ instead of $f_1 \circ \ldots \circ f_n$. Furthermore, we abbreviate $L \circ L^*$ by L^+.

Proposition 2.1 *The class of regular sets is closed under union, intersection and complement.*

The signature of RFT consists of

- all elements of $\mathcal{L}$ acting as constants, and
- all regular expressions L with $[\![L]\!] \subseteq \mathcal{L}^+$, which are taken as binary predicate symbols.

We use infix notation tLt' for so-called **regular path expressions**. We have excluded the empty path in the regular expressions since $x\{\epsilon\}y$ would be an atomic formula equivalent to $x \doteq y$, which we wanted to avoid. The **feature tree structure** $\mathfrak{T}_{\mathrm{RFT}}$ is the following RFT-structure:

- the universe $\mathbf{U}(\mathfrak{T}_{\mathrm{RFT}})$ of $\mathfrak{T}_{\mathrm{RFT}}$ is the set of all feature trees

- $c^{\mathfrak{T}_{\mathrm{RFT}}} = (\{\epsilon\}, \{(\epsilon, c)\})$ for every constant symbol $c \in \mathcal{L}$.
- for every regular expression L we have $(\sigma, \tau) \in L^{\mathfrak{T}_{\mathrm{RFT}}}$ iff there is a path $p \in [\![L]\!]$ such that $\tau = p^{-1}\sigma$.

For the models $\mathfrak{T}_{\mathrm{F}'}$, $\mathfrak{T}_{\mathrm{FT}'}$, $\mathfrak{T}_{\mathrm{CFT}'}$ and $\mathfrak{T}_{\mathrm{RFT}}$ we denote the substructure consisting only of the rational trees by $\mathfrak{R}_{\mathrm{F}'}$, $\mathfrak{R}_{\mathrm{FT}'}$, $\mathfrak{R}_{\mathrm{CFT}'}$ and $\mathfrak{R}_{\mathrm{RFT}}$, respectively.

2.2.2 Some Properties

In this and the next chapter, we concentrate on the theory of the standard interpretation of F′ (i.e., $\mathrm{Th}(\mathfrak{T}_{\mathrm{F}'})$), and we show that the theories of standard interpretations of the other languages are definitionally equivalent to fragments of $\mathrm{Th}(\mathfrak{T}_{\mathrm{F}'})$. The constraint $\cdot[\cdot]\cdot$ used in F′ generalises over ordinary feature constraints (as for example used in FT′ and CFT′) in that it allows features as first class values. Such predicates were introduced by Johnson [Joh88] and Treinen [Tre93]. But these authors didn't address the problem of showing that F′ is a universal feature description language.

We encode additional predicates in F′ using a standard first-order method by providing explicit definitions for them.

Definition 2.3 (Explicit Definition) *Let L be one of the languages* F′, FT′, CFT′, *or* RFT *and $R \subset \prod_{i=1}^{n} \mathcal{T}$ be some relation over feature trees. An* explicit definition *for R in $\mathfrak{T}_L$ in terms of L consists of a L-formula $\phi(x_1, \ldots, x_n)$ having $x_1, \ldots, x_n$ as free variables such that for all valuations α in $\mathfrak{T}_L$*

$$\mathfrak{T}_L, \alpha \models \phi(x_1, \ldots, x_n) \quad \textit{iff} \quad (\alpha(x_1), \ldots, \alpha(x_n)) \in R$$

Similarly, we say that $\phi(x_1, \ldots x_n)$ is an explicit definition *in $\mathfrak{R}_L$ in terms of L for the restriction R' of R to the set of rational trees iff for all valuations α in $\mathfrak{R}_L$*

$$\mathfrak{R}_L, \alpha \models \phi(x_1, \ldots, x_n) \quad \textit{iff} \quad (\alpha(x_1), \ldots, \alpha(x_n)) \in R'.$$

If L' is another language defined above, we say that $\mathrm{Th}(\mathfrak{T}_{L'})$ *(or* $\mathrm{Th}(\mathfrak{R}_{L'})$*) is* definitionally equivalent to a fragment *of* $\mathrm{Th}(\mathfrak{T}_L)$ *(or* $\mathrm{Th}(\mathfrak{R}_L)$*) if for every relation symbol R in the signature of L', the relation $R^{\mathfrak{T}_{L'}}$ (or $R^{\mathfrak{R}_{L'}}$) has an explicit definition in $\mathfrak{T}_L$ (or $\mathfrak{R}_L$) in terms of L, respectively.*

The definitions we will present are the same regardless whether we consider the feature tree interpretation or the rational feature tree interpretation for the corresponding

languages (although the theories of $\mathfrak{T}_{\mathrm{F}'}$ and $\mathfrak{R}_{\mathrm{F}'}$ for example differ; see Proposition 3.5, page 39). Hence, we could also have chosen the rational feature tree interpretations as standard interpretations. For this reason we will just say that $\phi(x_1, \ldots, x_n)$ is a definition for R within L if ϕ is an explicit definition for R in both $\mathfrak{T}_L$ and $\mathfrak{R}_L$ in terms of L, and that R can be defined within L (or is definable within L) if there is a definition for R within L. Furthermore, we say that L' is definitionally equivalent to a fragment of L if the theories of $\mathfrak{T}_{L'}$ and $\mathfrak{R}_{L'}$ are definitionally equivalent to fragments of $\mathrm{Th}(\mathfrak{T}_L)$ and $\mathrm{Th}(\mathfrak{R}_L)$, respectively.

In the following, we state the relations between the different languages and the known decidability and undecidability results for the theories of the feature tree interpretations of these languages. Henceforth, we will deliberately confuse the languages F′, FT′, CFT′ and RFT with $\mathrm{Th}(\mathfrak{T}_{\mathrm{F}'})$, $\mathrm{Th}(\mathfrak{T}_{\mathrm{FT}'})$, $\mathrm{Th}(\mathfrak{T}_{\mathrm{CFT}'})$ and $\mathrm{Th}(\mathfrak{T}_{\mathrm{RFT}})$, respectively.

For the feature constraints used in FT′ and CFT′, the definition for xfy is just $x[f]y$. For the *atom* predicate, there are two possible definitions, namely

$$\mathtt{atom}(x) \quad := \quad \exists u, v(u[x]v)$$

or

$$\mathtt{atom}(x) \quad := \quad \neg\exists u, v(x[u]v).$$

The existence of this dual characterisation reflects the fact that the label set $\mathcal{L}$ is used for both labelling the edges and the leaves of feature trees. A similarly simple definition is given in [Tre93] for the arity relations of CFT′. Given an (non-empty) arity $F = \{f_1, \ldots, f_n\}$, the corresponding arity relation is defined within F′ as

$$\mathtt{arity}_{\{f_1,\ldots,f_n\}}(x) \quad := \quad \forall u(\exists y\, x[u]y \leftrightarrow \bigvee_{i=1}^{n} u \doteq f_i)$$

Henceforth, we use $x\{f_1, \ldots, f_n\}$ as an abbreviation for $\mathtt{arity}_{\{f_1,\ldots,f_n\}}(x)$.

The definition for the *adjoinAt* function is again very simple. For this purpose, we have just to consider *adjoinAt* as a functional relation. This leads to the following definition:

$$\mathtt{adjoinAt}(x, u, v, y) \quad := \quad y[u]v \wedge \forall z, z'(z \neq u \rightarrow (x[z]z' \leftrightarrow y[z]z')$$

The definitions for the regular path expressions are somewhat more difficult and will be presented in the next sections, where we handle the more complicated examples (see Lemma 3.1, page 41). Overall we get that FT′, CFT′ and RFT are definitionally equivalent to fragments of F′. Clearly, FT′ is a fragment of CFT′ since the signature

of FT′ is a subset of the signature of CFT′, but the converse does not hold (see Corollary 4.1, page 88). Furthermore, we can show that CFT′ is definitionally equivalent to a fragment of RFT by providing the following definitions in RFT:

$$\begin{aligned}
\mathtt{atom}(x) &:= \neg\exists y(x\mathcal{L}^+y)\\
\mathtt{feat}_f(x,y) &:= xfy\\
\mathtt{arity}_{\{f_1,\ldots,f_n\}}(x) &:= \exists y_1\ldots y_n(xf_1y_1\wedge\ldots\wedge xf_ny_n)\\
&\quad\wedge\ \neg\exists y(x\overline{\{f_1,\ldots,f_n\}}y)
\end{aligned}$$

On the other hand, we can show that RFT is undecidable[3]

Theorem 2.1 *The theories of* $\mathfrak{T}_{\mathrm{RFT}}$ *and* $\mathfrak{R}_{\mathrm{RFT}}$ *are undecidable.*

Proof. Venkataraman [Ven87] showed that the first-order theory of constructor trees with the subterm relation is undecidable. Since constructor constraints can be defined within RFT and since the RFT-constraint $x\mathcal{L}^+y$ is the same as the subterm relation, this result follows by an adaption of the proof in [Ven87]. □

Since we will show that both FT′ and CFT′ are decidable, this implies that RFT cannot be definitionally equivalent to a fragment of CFT′. Whether F′ is definitionally equivalent to a fragment of RFT is an open problem.

Concerning decidability and undecidability, some results can be taken from the literature. Beside the full theory of the feature tree interpretations of these languages, we consider the positive existential fragment (Σ_1^+-fragment) and the existential fragment (Σ_1-fragment). Furthermore, we consider the fragment which corresponds to the entailment test of existentially quantified conjunctions of atomic constraints, which is a subset of the Π_2-fragment. A formula ϕ entails a formula ψ in some theory T (written $\phi \models_T \psi$) if $T \models \tilde{\forall}(\phi \to \psi)$.

The decidability of the existential fragment of F′ and FT′ can be proven by an adaption of previous work on feature logic by [Smo88] and [Joh88]. Both presented a system that transforms a quantifier-free formula into a solved form. The decidability of the existential fragment of CFT′ was shown in [ST94].

The decidability of the fragment corresponding to the entailment test of possible existentially conjunction of atomic constraints was shown for FT′ in [AKPS94], and

[3]The undecidability of this language interpreted over arbitrary first-order structures where features are binary relations was shown in [BBN+93].

for CFT′ in [ST94]. For both theories, the so-called independence property was shown. Given existentially quantified conjunctions of atomic formulae $\phi, \psi_1, \ldots, \psi_n$, we say that the theory T has the independence property if

$$\phi \models_T \bigvee_{i=1}^{n} \psi_i \quad \Longleftrightarrow \quad \exists i : \phi \models_T \psi_i.$$

Simple first-order equivalence transformations show that T satisfies the independence property if and only if the following equivalence is valid in T:

$$\tilde{\exists}(\phi \wedge \bigwedge_{i=1}^{n} \neg\psi_i) \quad \models_T \quad \bigwedge_{i=1}^{n} \tilde{\exists}(\phi \wedge \neg\psi_i).$$

Clearly, independence allows one to extend the entailment test to formulae containing disjunction. In the completeness proofs for FT′ and CFT′, we will prove a generalised independence property for FT′ and CFT′, namely

$$\exists X(\phi \wedge \bigwedge_{i=1}^{n} \neg\psi_i) \quad \models_T \quad \bigwedge_{i=1}^{n} \exists X(\phi \wedge \neg\psi_i),$$

where X is an arbitrary set of variables and T is FT′ or CFT′.

The results are summarised in table 2.1. Note that all results also hold for the theories of the rational feature tree interpretations of these languages. Since the solved form algorithm for FT′ in [Smo88] can be easily adapted for F′, and every solved form algorithm for F′ is also one for FT′, we do not distinguish the existential fragment of F′ and FT′ in this table. For completeness, we have added the language EF′, which was introduced by Treinen [Tre93]. The signature of EF′ is the signature of F′ extended by arity constraints. Since arity constraints are definable in F′ this language is no real extension if we consider the full first-order theory. But there is clearly a difference when considering restricted fragments of these languages. Treinen showed, that the existential fragment of this theory is decidable, and that the full theory is undecidable.

	F′	FT′	CFT′	RFT	EF′
Σ_1^+-fragment		√ [Smo88, Joh88]	√ [ST94]	√ Theorem 5.5	√ [Tre93]
Σ_1-fragment		√ [Smo88, Joh88]	√ [ST94]		√ [Tre93]
Entailment + ∃ quantifiers + independence		√ [AKPS94]	√ [ST94]		
Full theory	◯ [Tre93]	√ Theorem 4.4	√ Theorem 4.6	◯ Theorem 2.1 (adaption of [Ven87])	◯ [Tre93]

Table 2.1: Collection of decidability and undecidability results. √ means decidable, and ◯ means undecidable.

Chapter 3

Expressivity of F′

3.1 Some F′-definable Relations

We present a feature tree encoding for tuples and sets, and show that we can define the usual relations upon these representations. These concepts will play an important role in defining further concepts such as regular path expressions and definite equivalences. The chosen representation of sets allows to encode finite sets in the case of the rational tree model $\mathfrak{R}_{\mathrm{F}'}$, and infinite sets in the case of the feature tree model $\mathfrak{T}_{\mathrm{F}'}$. This fact is used to show that the theories of $\mathfrak{T}_{\mathrm{F}'}$ and $\mathfrak{R}_{\mathrm{F}'}$ are different. Note that these cardinality restrictions have to be respected when using the set representation for other definitions.

We proceed showing that, under certain restrictions, the transitive and reflexive closure of definable, binary relations is definable in F′. This part applies a simple variant of the general technique that we use to simulate computation within F′. The ability for defining the transitive and reflexive closure of definable relations is then used for defining regular path expressions, which are the descriptional primitives of the language RFT. A regular path expression is a subtree relation, where the path to the subtree is restricted to a regular expression. As in the case of the *subtree* relation, an implicit existentially quantification over the path to the subtree is used. We can gain additional expressivity if one splits regular path expressions into the *subtreeAt* relation and a relation which restricts a variable denoting a path to a given regular expression. Thus, paths become first class values. We will consider this generalisation for three reasons. First, these relations are the descriptional primitives that are used in the Chapter 5 for solving the satisfiability problem for RFT. Second, these relations can be used for defining the *simulate* and *subsume* relations. And third, the

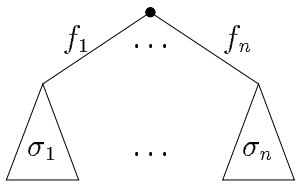
definition for the *subtreeAt* relation is exactly the kind of formula that is used in the definition for relations that are formed by definite equivalences.

Finally, we will show that we can encode natural numbers in feature trees, and that we can define the successor and predecessor relation, and the relation "x represents a natural number". The encoding of natural numbers will also play an important role when considering definite equivalences.

Note that all definitions can be used for both $\mathfrak{T}_{\mathrm{F}'}$ and $\mathfrak{R}_{\mathrm{F}'}$. Therefore, we use a meta-variable $\mathfrak{M}$ ranging over the two feature tree structures $\mathfrak{T}_{\mathrm{F}'}$ and $\mathfrak{R}_{\mathrm{F}'}$. This eases the technical details of the proofs for the parts which are common for both structures. This makes sense since even the proofs are very similar and must be distinguished only at some (rare) occasions.

3.1.1 Tuples and Sets

We start with the definition of tuples. Although we will later define sets, and a standard technique is to define tuples using sets, we use a more direct and simpler approach. For the representation of tuples we assume an arbitrary but fixed enumeration $f_1, \ldots, f_n, \ldots$ of $\mathcal{L}$. Using this enumeration, a tuple $(\sigma_1, \ldots, \sigma_n)$ of feature trees is represented by the feature tree



Now we define

$$TUPLE_n : \mathcal{T}^n \to \mathcal{T}$$

to be the function that maps a given tuple $(\sigma_1, \ldots, \sigma_n)$ to the corresponding feature tree representation.

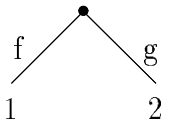
Proposition 3.1 *The relation "$TUPLE_n(\sigma_1, \ldots, \sigma_n) = \sigma$" is definable within* F′.

Proof. The definition for this relation is $y\{f_1, \ldots, f_n\} \wedge y[f_1]x_1 \wedge \ldots \wedge y[f_n]x_n$. □

In order to reduce the complexity of formulae, we use the syntactic sugar $\langle t_1, \ldots, t_n \rangle$ for tuples of length n. We treat $\langle t_1, \ldots, t_n \rangle$ as a functional term, i.e., for a formula $\phi(x)$ with $x \in \mathcal{V}(\phi)$ we write $\phi(\langle t_1, \ldots, t_n \rangle)$ as an abbreviation for

$$\exists y(\phi(y) \wedge y\{f_1, \ldots, f_n\} \wedge y[f_1]t_1 \wedge \ldots \wedge y[f_n]t_n).$$

Next we continue with the feature tree representation of sets. We use a technique introduced in [Tre93], where a set $\{\sigma_1, \ldots, \sigma_n\}$ of feature trees is represented by a feature tree σ which has $\sigma_1, \ldots, \sigma_n$ as direct subtrees. Thus, the feature tree



is one possible representation of the set $\{1, 2\}$. Note that we could have used any pair of distinct features instead of (f, g) for representing the set $\{1, 2\}$. The total function

$$SET : \mathcal{T} \to \wp(\mathcal{T})$$

maps every feature tree to the set of its direct subtrees. Note that we do not have a unique representation for a given set of feature trees. For example, all constant symbols represent the empty set. Furthermore, given a representation of an arbitrary set as a feature tree, we can generate a different feature tree representing the same set by renaming the features under the root consistently, or by taking additional occurrences of subtrees.

Proposition 3.2 *For every feature tree σ, $SET(\sigma)$ is a countable set. If σ is a rational feature tree, then $SET(\sigma)$ is finite.*

We say that a set m of feature trees is representable in $\mathfrak{M}$ if there is a feature tree $\sigma \in \mathbf{U}(\mathfrak{M})$ with $SET(\sigma) = m$. By the above proposition, the class of sets representable in $\mathfrak{R}_{\mathrm{F}'}$ is the class of all finite sets, and the class of sets representable in $\mathfrak{T}_{\mathrm{F}'}$ is the class of all countable sets.

Proposition 3.3 *The relations "$\sigma \in SET(\tau)$", "$SET(\sigma) \subseteq SET(\tau)$", "$SET(\sigma) \cup SET(\sigma') = SET(\tau)$" and "$SET(\sigma) \cap SET(\sigma') = SET(\tau)$" are definable within* F′.

Proof. The corresponding definitions are

$$\begin{aligned}
\texttt{in}(x,y) &:= \exists u(y[u]x)\\
\texttt{subset}(x,y) &:= \forall u[\texttt{in}(u,x) \rightarrow \texttt{in}(u,x)]\\
\texttt{union}(x,y,z) &:= \forall u[\texttt{in}(u,z) \leftrightarrow (\texttt{in}(u,x) \vee \texttt{in}(u,y))]\\
\texttt{intersec}(x,y,z) &:= \forall u[\texttt{in}(u,z) \leftrightarrow (\texttt{in}(u,x) \wedge \texttt{in}(u,y))].
\end{aligned}$$

□

Throughout the text the letters M and N are used for first-order variables which are intended to denote sets. Note that we could also use our representation of sets for encoding the extension of feature descriptions by set descriptions (so-called set-values) as they have been introduced in [Rou88, PM90, MP93, Man93b].

In the following, we will often use the feature tree representation of sets for representing n-ary relations, which are encoded as sets of tuples of length n. For simplicity reasons, we do not guarantee that these sets contain *only* feature trees representing tuples of length n. This implies that for every $n \in \mathbb{N}$, every feature tree represents an n-ary relation. Thus, we define the function

$$REL_n : \mathcal{T} \rightarrow \wp(\mathcal{T}^n)$$

by $REL_n(\sigma) = \{(\sigma_1, \ldots, \sigma_n) \mid TUPLE_n(\sigma_1, \ldots, \sigma_n) \in SET(\sigma)\}$.

Proposition 3.4 *The relation "$SET(\sigma)$ is infinite" is definable in* F′.

Proof. Recall that $SET(\sigma)$ is infinite if there is a function $f : SET(\sigma) \rightarrow SET(\sigma)$ which is injective but not surjective. Now we can define the relation "$REL_2(\tau)$ is a total, functional relation over $SET(\sigma)$" by

$$\begin{aligned}
\texttt{totalfun}(N,M) &:= \forall x,y[\texttt{in}(\langle x,y\rangle, N) \rightarrow (\texttt{in}(x,M) \wedge \texttt{in}(y,M))]\\
&\wedge \quad \forall x,y,y'[(\texttt{in}(\langle x,y\rangle, N) \wedge \texttt{in}(\langle x,y'\rangle, N)) \rightarrow y = y']\\
&\wedge \quad \forall x(\texttt{in}(x,M) \rightarrow \exists y\ \texttt{in}(\langle x,y\rangle, N))
\end{aligned}$$

The relations "$REL_2(\tau)$ is injective on $SET(\sigma)$" and "$REL_2(\tau)$ is surjective on $SET(\sigma)$" can be defined within F′ by

$$\begin{aligned}
\texttt{inj}(N,M) &:= \texttt{totalfun}(N,M)\\
&\wedge \quad \forall x,y,x'[(\texttt{in}(\langle x,y\rangle, N) \wedge \texttt{in}(\langle x',y\rangle, N)) \rightarrow x = x']\\
\texttt{surj}(N,M) &:= \texttt{totalfun}(N,M) \wedge \forall x[\texttt{in}(x,M) \rightarrow \exists y(\texttt{in}(\langle y,x\rangle, N))]
\end{aligned}$$

Hence, we can define the relation "$SET(\sigma)$ is infinite" by $\exists N(\texttt{inj}(N,M) \wedge \neg\, \texttt{surj}(N,M))$.
□

Proposition 3.5 (Treinen 93) $\mathrm{Th}(\mathfrak{R}_{\mathrm{F}'}) \neq \mathrm{Th}(\mathfrak{T}_{\mathrm{F}'})$.

Proof. Follows from the fact that the relation "$SET(\sigma)$ is infinite" is definable within F′, and that by Proposition 3.2 only finite sets are representable in $\mathfrak{R}_{\mathrm{F}'}$, whereas we can represent also infinite sets in $\mathfrak{T}_{\mathrm{F}'}$. □

We have proven this proposition using the definition for "$SET(\sigma)$ is infinite" since we wanted to show that the property of being an infinite set is definable in F′. Treinen [Tre93] used a more direct approach to prove Proposition 3.5. He constructed a formula $\phi(x)$ which expresses that x has an infinite number of children, which themselves have a strictly increasing number of children. Using $x \lhd y$ as an abbreviation for $\exists u(x[u]y)$ (which can be read as "y is a child of x"), this formula is defined by

$$\begin{aligned}\phi(x) \;:=\; & \exists y(x \lhd y) \wedge \forall y(x \lhd y \rightarrow \exists y'(x \lhd y' \wedge \forall z(y \lhd z \rightarrow y' \lhd z) \wedge \\ & \qquad\qquad \exists z(y' \lhd z \wedge \neg y \lhd z)))\end{aligned}$$

It is easy to verify that $\mathfrak{T}_{\mathrm{F}'} \models \exists x \phi(x)$, whereas $\mathfrak{R}_{\mathrm{F}'} \models \forall x \neg \phi(x)$.

The following proposition gives the definition for the *subtree* relation as presented in [Tre93].

Proposition 3.6 (Treinen 93) *The relation "$\exists p : p^{-1}\sigma = \tau$" is definable within* F′.

Proof. The definition for this relation is

$$\mathtt{subtree}(x, y) \;:=\; \forall M(\mathtt{in}(x, M) \wedge \mathtt{subtree\_clos}(x, M) \rightarrow \mathtt{in}(y, M)),$$

where $\mathtt{subtree\_clos}(x, M)$ is the formula

$$\forall y, z(\mathtt{in}(y, M) \wedge \exists u(y[u]z) \rightarrow \mathtt{in}(z, M)).$$

$\mathtt{subtree\_clos}(M)$ is the definition for the relation "$SET(\sigma)$ is closed under the direct subtree relation". We have to show that for every valuation α in $\mathfrak{M}$,

$$\mathfrak{M}, \alpha \models \mathtt{subtree}(x, y)$$

if and only if $\alpha(y)$ is a subtree of $\alpha(x)$. For every $\sigma \in \mathbf{U}(\mathfrak{M})$ let S_σ be the set of all subtrees of σ.

For the "if" direction note that if

$$\mathfrak{M}, \alpha \models \mathtt{in}(x, M) \wedge \mathtt{subtree\_clos}(M),$$

then $S_{\alpha(x)} \subseteq SET(\alpha(M))$. Hence, $\mathfrak{M}, \alpha \models \mathtt{in}(y, M)$.

For the "only if" direction we have to show that for every feature tree $\sigma \in \mathbf{U}(\mathfrak{M})$ the set S_σ is representable in $\mathfrak{M}$. If $\mathfrak{M} = \mathfrak{R}_{\mathrm{F}'}$, then S_σ is finite since σ is a rational tree. If $\mathfrak{M} = \mathfrak{T}_{\mathrm{F}'}$, then σ can have at most as many different subtrees as paths are defined on σ. This implies that the cardinality of S_σ is at most $|\mathcal{L}^*|$. Since we have assumed that $\mathcal{L}$ is countable, we know that $\mathcal{L}^*$ is also countable. Hence, the set S_σ is representable in $\mathfrak{M}$ by Proposition 3.2. □

The definition of `subtree` uses a general technique for defining reflexive and transitive closures of binary relations.

Definition 3.1 *Given a definition $\phi_R(x, y)$ for a binary relation $R \subseteq \mathbf{U}(\mathfrak{M}) \times \mathbf{U}(\mathfrak{M})$, we define $\mathtt{refl\text{-}trans}_{\phi_R}(x, y)$ to be the formula*

$$\mathtt{refl\text{-}trans}_{\phi_R}(x, y) \;:=\; \forall M((\mathtt{in}(x, M) \wedge \mathtt{closure}_{\phi_R}(M)) \rightarrow \mathtt{in}(y, M)),$$

where $\mathtt{closure}_{\phi_R}(M)$ is the formula

$$\mathtt{closure}_{\phi_R}(M) \;:=\; \forall z, z'((\mathtt{in}(z, M) \wedge \phi_R(z, z')) \rightarrow \mathtt{in}(z', M)))$$

Proposition 3.7

1. *For every binary relation $R \subseteq \mathbf{U}(\mathfrak{T}_{\mathrm{F}'}) \times \mathbf{U}(\mathfrak{T}_{\mathrm{F}'})$ which has an explicit definition $\phi_R(x, y)$ in $\mathfrak{T}_{\mathrm{F}'}$ in terms of F', the formula $\mathtt{refl\text{-}trans}_{\phi_R}(x, y)$ is an explicit definition for the reflexive and transitive closure of R in $\mathfrak{T}_{\mathrm{F}'}$ in terms of F' if for every $\sigma \in \mathbf{U}(\mathfrak{R}_{\mathrm{F}'})$ the set*

$$\{\tau \mid \exists n \in \mathbb{N} : (\underbrace{R \circ \ldots \circ R}_{n\text{-times}})(\sigma, \tau)\}$$

is countable.

2. *For a relation $R \subseteq \mathbf{U}(\mathfrak{R}_{\mathrm{F}'}) \times \mathbf{U}(\mathfrak{R}_{\mathrm{F}'})$ which has an explicit definition $\phi_R(x, y)$ in $\mathfrak{R}_{\mathrm{F}'}$ in terms of F', the formula $\mathtt{refl\text{-}trans}_{\phi_R}(x, y)$ is an explicit definition for the reflexive and transitive closure of R in $\mathfrak{R}_{\mathrm{F}'}$ in terms of F' if for every $\sigma \in \mathbf{U}(\mathfrak{R}_{\mathrm{F}'})$ the set*

$$\{\tau \mid \exists n \in \mathbb{N} : (\underbrace{R \circ \ldots \circ R}_{n\text{-times}})(\sigma, \tau)\}$$

is finite.

Proof. For every $\sigma \in \mathbf{U}(\mathfrak{M})$ let S_σ be the set

$$S_\sigma \;=\; \{\tau \mid \exists n \in \mathbb{N} : (\underbrace{R \circ \ldots \circ R}_{n\text{-times}})(\sigma, \tau)\}.$$

It is easy to check that for every α with

$$\mathfrak{M}, \alpha \models \mathtt{in}(x, M) \wedge \mathtt{closure}_{\phi_R}(M)$$

the set $S_{\alpha(x)}$ is a subset of $SET(\alpha(M))$. Hence, it suffices to show that for every $\sigma \in \mathbf{U}(\mathfrak{M})$ the set S_σ is representable in $\mathfrak{M}$. For $\mathfrak{M} = \mathfrak{R}_{\mathrm{F}'}$ we have assumed that S_σ is always finite, and for $\mathfrak{M} = \mathfrak{T}_{\mathrm{F}'}$ we have assumed that S_σ is always countable. This implies that S_σ is representable by Proposition 3.2. □

3.1.2 Regular Path Expressions

In this section, we show that the descriptional primitives of the language RFT are definable within F′. Recall that in RFT, a regular expression L is taken as a binary relation symbol whose interpretation in $\mathfrak{T}_{\mathrm{RFT}}$ is

$$(\sigma, \tau) \in L^{\mathfrak{T}_{\mathrm{RFT}}} \quad \text{iff} \quad \exists p \in [\![L]\!] : p^{-1}\sigma = \tau.$$

Lemma 3.1 *For every regular language L over the alphabet $\mathcal{L}$, the relation "$\exists p \in [\![L]\!] : p^{-1}\sigma = \tau$" is definable within* F′.

Proof. The explicit definition $\mathtt{regexp}_L$ for a some regular language L within F′ is given inductively as follows:

$$\begin{aligned}
\mathtt{regexp}_\emptyset(x,y) &:= \bot \\
\mathtt{regexp}_\epsilon(x,y) &:= x \doteq y \\
\mathtt{regexp}_F(x,y) &:= \bigvee_{f \in F} x[f]y \\
\mathtt{regexp}_{\overline{F}}(x,y) &:= \exists z(x[z]y \wedge \bigwedge_{f \in F} z \neq f) \\
\mathtt{regexp}_{L_1 \cup L_2}(x,y) &:= \mathtt{regexp}_{L_1}(x,y) \vee \mathtt{regexp}_{L_2}(x,y) \\
\mathtt{regexp}_{L_1 \circ L_2}(x,y) &:= \exists z(\mathtt{regexp}_{L_1}(x,z) \wedge \mathtt{regexp}_{L_2}(z,y)) \\
\mathtt{regexp}_{L^*}(x,y) &:= \mathtt{refl\text{-}trans}_{\mathtt{regexp}_L}(x,y).
\end{aligned}$$

We show by induction over the structure of regular expressions that

$$\mathfrak{M}, \alpha \models \mathtt{regexp}_L(x,y)$$

if and only if there is a path $p \in [\![L]\!]$ such that $subtreeAt(\alpha(x), p, \alpha(y))$ holds.

For the base cases $L = \emptyset$, $L = \epsilon$, $L = F$ or $L = \overline{F}$ the claim holds trivially. For the induction step the claim is easy to show if L is a regular expression of the form $L_1 \cup L_2$ or $L_1 \circ L_2$.

Otherwise, let L be of the form R^*. Let $\sigma = \alpha(x)$ and let S_σ be the set

$$S_\sigma \quad = \quad \{\tau \mid \exists p \in [\![R^*]\!] : subtreeAt(\sigma, p, \tau)\}.$$

By an similar argumentation as in the proof of Proposition 3.6 we get that S_σ is representable in $\mathfrak{M}$. Hence, the claim follows from Proposition 3.7. □

Next we want to generalise regular path expressions. The constraints xLy contains an implicit existential quantification of the path since $\mathfrak{M}, \alpha \models \mathtt{regexp}_L(x, y)$ iff $\exists p : (subtreeAt(\alpha(x), p, \alpha(y)) \ \wedge \ p \in [\![L]\!])$. We gain additional expressivity if we allow also universal quantification over the path p, which means that we make paths first class values. For this purpose we have first to find some feature tree representation of paths. Furthermore, we have to define the *subtreeAt* relation using this representation of paths, and the relation "the path represented by σ is in $[\![L]\!]$". Assuming that there is a symbol $\epsilon \in \mathcal{L}$, a path $f_1 \ldots f_n \in \mathcal{L}^*$ is represented by the feature tree

The partial function

$$PATH : \mathcal{T} \rightsquigarrow \mathcal{L}^*$$

maps these feature trees to the corresponding paths. For clarity, we use the letter v for first-order variables that are intended to denote paths.

Proposition 3.8 *The relation "PATH(σ) is defined" is definable within* F′.

Proof. The definition for this relation is

$$\mathtt{path}(v) \quad := \quad \forall y(\mathtt{subtree}(v, y) \rightarrow (y = \epsilon \vee \mathtt{onefeat}(y))) \ \wedge \ \mathtt{subtree}(v, \epsilon),$$

where $\mathtt{onefeat}(y)$ is the formula $\exists u \forall u'(\exists z(y[u']z) \leftrightarrow u \doteq u')$ □

Proposition 3.9 *The relation "subtreeAt$(\sigma, PATH(\tau), \sigma')$" is definable within* F′.

Proof. The definition for this relation is

$$\begin{aligned}\texttt{subtreeAt}(x,v,y) \;:=\; & \exists M(\,\texttt{in}(\langle x,v,y\rangle, M) \wedge \\ & \forall x,v,y(\texttt{in}(\langle x,v,y\rangle, M) \rightarrow \quad v = \epsilon \wedge x \doteq y \\ & \qquad\qquad \vee\, \texttt{onestep}(x,v,y,M))),\end{aligned} \tag{3.1}$$

where $\texttt{onestep}(x,v,y,M)$ is defined as

$$\begin{aligned}\texttt{onestep}(x,v,y,M) \;:=\; & \texttt{path}(v) \wedge \\ & \exists x', v', z(v[z]v' \wedge x[z]x' \wedge \texttt{in}(\langle x', v', y\rangle, M)\,).\end{aligned}$$

We have to show that

$$\mathfrak{M}, \alpha \models \texttt{subtreeAt}(x,v,y) \quad \text{iff} \quad subtreeAt(\alpha(x), PATH(\alpha(v)), \alpha(y)).$$

The "if" direction is left to the reader. For the "only if" it is sufficient to prove that if

$$\begin{aligned}\mathfrak{M}, \alpha \models \forall x,v,y(\texttt{in}(\langle x,v,y\rangle, M) \rightarrow \quad & v = \epsilon \wedge x \doteq y \\ & \vee\, \texttt{onestep}(x,v,y,M))),\end{aligned}$$

then every tuple $(\sigma_1, \tau, \sigma_2) \in REL_3(\alpha(M))$ satisfies $subtreeAt(\sigma_1, PATH(\tau), \sigma_2)$. Note that $(\sigma_1, \tau, \sigma_2)$ in $REL_3(\alpha(M))$ implies that τ is the feature tree representation of a path. Hence, we can use induction over the length of $PATH(\tau)$.

For the induction beginning $PATH(\tau) = \epsilon$ we know for every $\{x,v,y\}$-update α' of α with $\alpha'(x) = \sigma_1$, $\alpha'(v) = \tau$ and $\alpha'(y) = \sigma_2$ that

$$\mathfrak{M}, \alpha' \not\models \texttt{onestep}(x,v,y,M),$$

which implies $\sigma_1 = \sigma_2$.

For the induction step assume that we have proven the claim for all tuples $(\sigma_1, \tau, \sigma_2) \in REL_3(\alpha(M))$ with $|\,PATH(\tau)| \leq n$. Let $(\sigma_1, \tau, \sigma_2)$ be an element of $REL_3(\alpha(M))$ with $|\,PATH(\tau)| = n+1$, and let α' be a $\{x,v,y\}$-update of α with $\alpha'(x) = \sigma_1$, $\alpha'(v) = \tau$ and $\alpha'(y) = \sigma_2$. Since τ is not the empty path, we know that

$$\mathfrak{M}, \alpha' \models \texttt{onestep}(x,v,y,M).$$

Hence, there are trees σ_1', τ' and a feature f such that $PATH(\tau) = fp$, $PATH(\tau') = p$ and $subtreeAt(\sigma_1, f, \sigma_1')$. By induction hypotheses we get $subtreeAt(\sigma_1', p, \sigma_2)$. Hence, $subtreeAt(\sigma_1, fp, \sigma_2)$. □

$\texttt{subtreeAt}(x, v, y)$ is a good example for the general method of defining relations. As we will see in the next chapter, we can at least define all those relations within F′ which are definable by definite equivalences. To see this consider the following alternative definition of `subtreeAt` using definite equivalences:[1]

$$\begin{aligned}\texttt{subtreeAt}(x, v, y) \quad &\leftrightarrow \quad v = \epsilon \wedge x \doteq y \\ &\vee \quad (\,\texttt{path}(v) \wedge \\ &\qquad \exists x', v', z(v[z]v' \wedge x[z]x' \wedge \texttt{subtreeAt}(x', v', y)))\end{aligned}$$

Note the similarity between the second clause and the definition of $\texttt{onestep}(x, v, y, M)$. In fact, for every valuation σ_M of the variable M in the definition of `subtreeAt` (see (3.1)) the extension of σ_M encodes a subset of the `subtreeAt` relation. If $\mathfrak{M}$ is the rational tree model, then σ_M encodes a finite subset of `subtreeAt`, otherwise it may be infinite. Note that a finite extension is alway sufficient.

Proposition 3.10 *The relation "$PATH(\sigma) \in [\![L]\!]$" is definable within* F′.

We need an additional proposition for the proof.

Proposition 3.11 *Given two regular expressions L_1, L_2, then $[\![L_1]\!] \subseteq [\![L_2]\!]$ if and only if $\mathfrak{M} \models \forall x, y(\texttt{regexp}_{L_1}(x, y) \rightarrow \texttt{regexp}_{L_2}(x, y))$.*

Proof. The "only if" direction is trivial. For the "if" direction assume that $\mathfrak{M} \models \forall x, y(\texttt{regexp}_{L_1}(x, y) \rightarrow \texttt{regexp}_{L_2}(x, y))$. We have to show that $[\![L_1]\!] \subseteq [\![L_2]\!]$. Let $p \in [\![L_1]\!]$ be a path, let σ be the feature tree

$$(\{p' \mid p' \preceq p\}, \emptyset).$$

σ is rational tree, which implies that $\sigma \in \mathbf{U}(\mathfrak{M})$. Clearly, $subtreeAt(\sigma, p, \tau)$, where τ is the feature tree $(\{\epsilon\}, \emptyset)$. Furthermore, p is the only path connecting σ with τ. Now we have assumed that $\mathfrak{M} \models \forall x, y(\texttt{regexp}_{L_1}(x, y) \rightarrow \texttt{regexp}_{L_2}(x, y))$. This implies that a valuation α with $\alpha(x) = \sigma$ and $\alpha(y) = \tau$ satisfies $\mathfrak{M}, \alpha \models \texttt{regexp}_{L_2}(x, y)$, which shows $p \in [\![L_2]\!]$. □

Proof of 3.10. By the last proposition, the definition for the relation "$PATH(\sigma) \in [\![L]\!]$" is

$$\texttt{pathrestr}_L(v) \quad := \quad \texttt{path}(v) \wedge \forall x, y(\texttt{subtreeAt}(x, v, y) \rightarrow \texttt{regexp}_L(x, y)).$$

□

[1] Note that this is no explicit definition for *subtreeAt*, since the definition itself makes use of the symbol `subtreeAt` to be defined. Definitions of this form are sometimes called recursive definitions.

Proposition 3.12 *The relations simulate and subsume are definable within* F′.

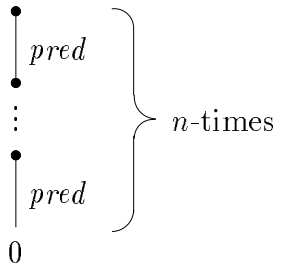
Proof. The definitions are

$$\begin{aligned}
\texttt{simulate}(x,y) &:= \forall v(\exists z\,\texttt{subtreeAt}(x,v,z) \rightarrow \exists z\,\texttt{subtreeAt}(y,v,z)) \\
&\wedge \quad \forall v,z(\texttt{subtreeAt}(x,v,z) \wedge \texttt{atom}(z) \rightarrow \texttt{subtreeAt}(y,v,z)) \\
\texttt{subsume}(x,y) &:= \forall v,v'[\exists z(\texttt{subtreeAt}(x,v,z) \wedge \texttt{subtreeAt}(x,v',z)) \\
&\qquad \rightarrow \exists z(\texttt{subtreeAt}(y,v,z) \wedge \texttt{subtreeAt}(y,v',z))].
\end{aligned}$$

□

3.1.3 Natural Numbers

In this section we show that we can encode natural numbers and the operations successor and predecessor. We use the atom $0 \in \mathcal{L}$ to represent zero. Hence, we can represent a natural number n by the feature tree



The partial function

$$NUM : \mathcal{T} \rightsquigarrow \mathbb{N}$$

maps these feature trees to the corresponding natural number. Note that for every $n \in \mathbb{N}$ there is exactly one feature tree σ with $NUM(\sigma) = n$.

Proposition 3.13 *The relations "$NUM(\sigma)$ is defined", "$NUM(\sigma)$ is the predecessor of $NUM(\tau)$" and "$NUM(\sigma)$ is the successor of $NUM(\tau)$" are definable within* F′.

Proof. The corresponding definitions are

$$\begin{aligned}
\texttt{nat}(x) &:= \forall y[subtree(x,y) \rightarrow (y = 0 \vee y\{pred\})] \wedge subtree(x,0) \\
\texttt{pred}(x,y) &:= x\{pred\} \wedge x[pred]y \\
\texttt{succ}(x,y) &:= \texttt{pred}(y,x)
\end{aligned}$$

□

In the following, the letter C is used for first-order variables which are intended to represent natural numbers. Since for every feature tree σ representing a number there is a unique τ representing the successor of $NUM(\sigma)$, and since for every σ representing a number different from 0 there is a unique τ representing the predecessor of $NUM(\sigma)$, we write $\phi(C+1)$ and $\phi(C-1)$ as an abbreviation for $\exists C'(\mathtt{succ}(C, C') \wedge \phi(C'))$ and $\exists C'(C \neq 0 \wedge \mathtt{pred}(C, C') \wedge \phi(C'))$, respectively.

3.2 Definite Constructions

In this section, we show that the language F′ is even expressive enough to encode definite constructions, i.e., every relation that can be defined using definite constructions has a definition in F′. Definite constructions allow one to extend the signature by a set of new relations symbols $\mathcal{R}$, and to provide definitions for these relations in form of so-called definite equivalences, which are introduced in [Smo93]. The extended language $\mathcal{R}(\mathrm{F}')$ is called the relational extension of F′. An equivalence definition for some relation R is a formula

$$R(x_1, \ldots, x_n) \quad \leftrightarrow \quad D(x_1, \ldots, x_n),$$

where D is a definite formula which has at most $x_1, \ldots, x_n$ as free variables. The set of definite $\mathcal{R}(\mathrm{F}')$-formulae is generated by the production rule

$$D, D' \quad ::= \quad R(t_1, \ldots, t_n) \mid D \wedge D' \mid D \vee D' \mid \exists x D \mid \phi,$$

where $R \in \mathcal{R}$ denotes a relation symbol of arity n and ϕ a F′-formula. In the following we refer to a set of definite equivalences as a definite program.

[Smo93] associates to every definite program P a continuous function T_P over the set of $\mathcal{R}(\mathrm{F}')$-interpretations which has the property that every fixpoint of T_P is a model of P. The result of applying T_P to some interpretation $\mathfrak{A}$ of P is an interpretation $\mathfrak{A}'$ such that for every equivalence $R(x_1, \ldots, x_n) \leftrightarrow \phi(x_1, \ldots, x_n)$,

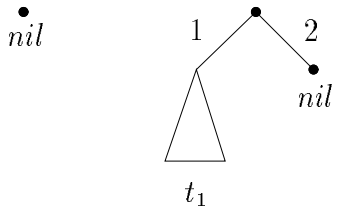
$$R^{\mathfrak{A}'} \quad = \quad \{(\alpha(x_1), \ldots, \alpha(x_n)) \mid \mathfrak{A}, \alpha \models \phi(x_1, \ldots, x_n)\}.$$

Since T_P is continuous, it has a least and a greatest fixpoint, which establishes a least and greatest model of P.

Usually, one is interested in the least model of a definite program. But the greatest model has also its applications. Recall the definition of the feature tree representation of lists that we have used in the introduction:

$$\begin{array}{rcl} \texttt{list}(x) & \leftrightarrow & x \doteq nil \\ & \vee & \exists y(x\{1,2\} \wedge x[2]y \wedge \texttt{list}(y)) \end{array}$$

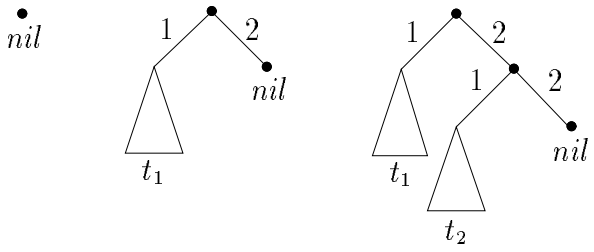
Now assume that we have an interpretation $\mathfrak{A}$, where $list^{\mathfrak{A}}$ contains all feature trees representing the empty list and unary lists, i.e., all feature trees of the form



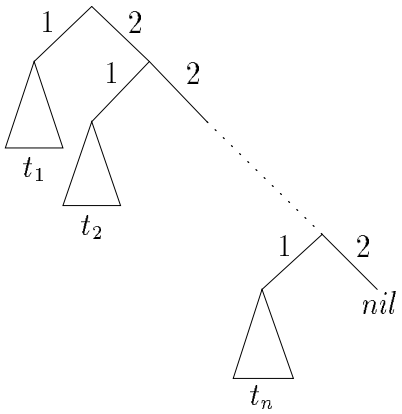
Let $\mathfrak{A}'$ be the result of applying the continuous function T_P associated to P on the interpretation $\mathfrak{A}$. Then the denotation of *list* in $\mathfrak{A}'$ is defined as

$$list^{\mathfrak{A}'} := \{\alpha(x) \mid \mathfrak{A}, \alpha \models x \doteq nil \vee \exists y(x\{1,2\} \wedge x[2]y \wedge \texttt{list}(y))\}.$$

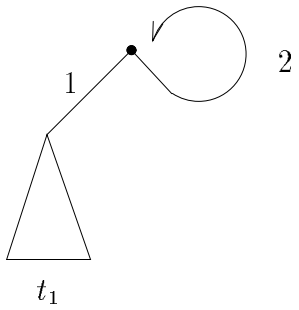
This implies that $list^{\mathfrak{A}'}$ contains all feature trees of the form



It is now easy to check that the least fixpoint of T_P contains only the representation of finite lists, i.e., the feature trees of the form



But this fixpoint does not contain the representation of cyclic or infinite lists. An example of the representation of an cyclic list is



These feature trees can be found in the greatest fixpoint of T_P.

The distinction between the least and greatest fixpoint of a definite program P can also be characterised in the following way. The computational service provided for definite programs usually tests whether a conjunction G of F′-formulae and atomic $\mathcal{R}$(F′)-formulae is satisfiable in the least model a definite program P. This is the same as asking whether P entails $\tilde{\exists}G$. If one is interested in whether $P \wedge \tilde{\exists}G$ is consistent, one has to use the greatest model. For this reason, the greatest fixpoint semantics was chosen for some formalisations of type systems, which are a special form of definite programs that use only unary predicates (see [AKPG93, Pol89, PM90, Kri95][2] for examples of type systems with greatest fixpoint semantics; for formalisations using the least fixpoint semantics see [AK86, EZ90b, EZ90a, Smo92, Car92][3]).

3.2.1 Definite Programs

The definitions presented in this section are similar to the one in [Smo93] and have been adapted for our purposes. In the following, let $\mathcal{R} = \bigcup_{n\in\mathbb{N}} \mathcal{R}_n$ be a set of relation symbols, where $\mathcal{R}_n$ contains exactly those relation symbols in $\mathcal{R}$ of arity n. As in the last section, we use $\mathfrak{M}$ as a meta-variable ranging over $\mathfrak{T}_{\mathrm{F}'}$ and $\mathfrak{R}_{\mathrm{F}'}$.

Definition 3.2 (Relational Extension) *The signature of the relational extension* $\mathcal{R}(\mathrm{F}')$ *of* F′ *is* $\mathcal{L} \cup \{\cdot[\cdot]\cdot\} \cup \mathcal{R}$.

[2] [Pol89, PM90] defined their semantics by a least fixpoint, but on a partial order that is just the inverse of partial order that is just by the other approaches. Hence, one has to use the dual notion for the comparison of the different approaches.

[3] In [EZ90a], the denotational semantics was defined via the least fixpoint; but the authors present also an operational semantics, which interestingly corresponds more to a greatest fixpoint semantics

Definition 3.3 ($\mathcal{R}(\mathrm{F}')$-interpretation) *An interpretation $\mathfrak{A}$ of $\mathcal{R}(\mathrm{F}')$ extending the feature tree model $\mathfrak{M}$ is a $\mathcal{R}(\mathrm{F}')$-structure with the following properties:*

- *the universe of $\mathfrak{A}$ is the universe of $\mathfrak{M}$*
- *$c^{\mathfrak{A}} = c^{\mathfrak{M}}$ for every $c \in \mathcal{L}$,*
- *$\cdot[\cdot]\cdot^{\mathfrak{A}} = \cdot[\cdot]\cdot^{\mathfrak{M}}$, and*
- *for every $R \in \mathcal{R}_n$ of arity n, $R^{\mathfrak{A}}$ is a subset of $\prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})$.*

The set of **definite** $\mathcal{R}(\mathrm{F}')$-formulae is given by the following formation rule

$$D, D' \quad ::= \quad R(t_1, \ldots, t_n) \mid D \wedge D' \mid D \vee D' \mid \exists x D \mid \phi,$$

where $R \in \mathcal{R}$ denotes a relation symbol of arity n and ϕ a F'-formula.

Definition 3.4 *An equivalence*

$$R(x_1, \ldots, x_n) \quad \leftrightarrow \quad D$$

is called definite *if*

- *R is a relation symbol of arity n,*
- *D is a definite formula, and*
- *$\mathcal{V}(D) \subseteq \{x_1, \ldots, x_n\}$.*

A finite set P of definite equivalences is called definite program *if P contains for every $R \in \mathcal{R}$ at most one definite equivalence $R(x_1, \ldots, x_n) \leftrightarrow D$.*

Since P is finite, it is sufficient to consider only those relation symbols that occur in P. For this reasons, we assume henceforth that $\mathcal{R}$ is a finite set of relation symbols, and that $\{R_1, \ldots, R_n\}$ is a fixed enumeration of $\mathcal{R}$. Furthermore, we assume that n_i denotes the arity of R_i. Since $\mathcal{R}$ is finite, we can encode $\mathcal{R}(\mathrm{F}')$ interpretations just as a finite tuple of sets. This view on interpretation is more appropriate for encoding definite programs within F'.

Definition 3.5 *The* interpretation domain $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ *for* $\mathcal{R} = \{R_1, \ldots, R_n\}$ *over* $\mathfrak{M}$ *is defined as*

$$\mathrm{intdom}(\mathcal{R}, \mathfrak{M}) = \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}),$$

where R_i *has arity* n_i *for every* $i \in 1 \ldots n$. *For every element* $m = (m_1, \ldots, m_n) \in \mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ *we define the associated interpretation* $\mathcal{I}(m)$ *by*

$$R_i^{\mathcal{I}(m)} = m_i.$$

Clearly, there is a one to one mapping of elements of $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ to $\mathcal{R}(\mathrm{F}')$-interpretations. We define the continuous function associated to a definite program P directly on elements of $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ and not on the $\mathcal{R}(\mathrm{F}')$-interpretations. In the following, we use $\subseteq$ for the component-wise subset relation, and $\cup$ for the component-wise union on elements of $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$.

Definition 3.6 *For a definite program* P *the mapping* $T_P^{\mathfrak{M}} : \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \to \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ *is defined by*

$$T_P^{\mathfrak{M}} : (m_1, \ldots, m_n) \mapsto (m_1', \ldots, m_n')$$

if and only if for every $i \in 1 \ldots n$

$$m_i' = \{(\alpha(x_1), \ldots, \alpha(x_n)) \mid \begin{array}{rl} & R_i(x_1, \ldots, x_{n_i}) \leftrightarrow D \in P \\ \textit{and} & \mathcal{I}((m_1, \ldots, m_n)), \alpha \models D\} \end{array}$$

Proposition 3.14 *Let* P *be a definite program and* $m \in \mathrm{intdom}(\mathcal{R}, \mathfrak{M})$. *Then* $\mathcal{I}(m)$ *is a model of* P *if and only if* m *is a fixpoint of* $T_P^{\mathfrak{M}}$.

Proof. Let $m = (m_1, \ldots m_n)$. Now $T_P^{\mathfrak{M}}(m) = m$ iff for all valuations α into $\mathcal{I}(m)$ and for every $R_i(x_1, \ldots, x_{n_i}) \leftrightarrow D$ in P

$$(\alpha(x_1), \ldots, \alpha(x_{n_i})) \in m_i \quad \Leftrightarrow \quad \mathcal{I}(m), \alpha \models D. \tag{3.2}$$

But (3.2) is the same as $\mathcal{I}(m), \alpha \models R_i(x_1, \ldots, x_{n_i}) \leftrightarrow D$. Hence, $T_P^{\mathfrak{M}}(m) = m$ iff $\mathcal{I}(m) \models P$. □

Next we show that $T_P^{\mathfrak{M}}$ is a continuous function for every definite program P, which implies that $T_P^{\mathfrak{M}}$ has a least and a greatest fixpoint. Therefore, there exist a least and a greatest model of P extending $\mathfrak{M}$. We assume the reader familiar with the basic lattice theory and the different fixpoint theorems for continuous functions (for an text book on lattice theory, see e.g. [DP90]; a short summary can be found in the appendix). For our purpose, we need one additional proposition.

Proposition 3.15 *Let* $T_P^{\mathfrak{M}} : \prod_{i=1}^n \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \to \prod_{i=1}^n \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ *be some function. Then* T *is continuous if and only if for every* $m \in \prod_{i=1}^n \wp(\mathbf{U}(\mathfrak{M})^{n_i})$,

$$T(m) = \bigcup\{f \in \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \mid f \subseteq m, f \text{ finite}\}.$$

Proof. Follows from Proposition A.1, where the more general case of algebraic lattices is considered, since $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ is an algebraic lattice. □

Proposition 3.16 *Let* D *be a definite formula,* m *an element of* $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ *and* α *be a valuation into* $\mathcal{I}(m)$ *such that* $\mathcal{I}(m), \alpha \models D$. *Then* $m', \alpha \models D$ *for every* $m \subseteq m'$.

Proof. By an easy induction proof on the structure of definite formulae. □

Corollary 3.1 $T_P^{\mathfrak{M}}$ *is monotone for every definite program* P.

Lemma 3.2 *Let* D *be a definite formula,* m *be an element of* $\mathrm{intdom}(\mathcal{R}, \mathfrak{M})$ *and* α *be a valuation into* $\mathcal{I}(m)$ *such that* $\mathcal{I}(m), \alpha \models D$. *Then there is a finite* $f \subseteq m$ *such that* $\mathcal{I}(f), \alpha \models D$.

Proof. By induction on the structure of definite formulae. Let m and α be given as described. If $D = \phi$ where ϕ is a F′-formula, then we define f as $(\emptyset, \ldots, \emptyset)$. If $D = R_i(t_1, \ldots, t_{n_i})$, then we can define f as $(f_1, \ldots, f_n)$ with $f_j = \emptyset$ for $j \neq i$, and $f_i = \{(\alpha(t_1), \ldots, \alpha(t_{m_i}))\}$. For the induction step we have the following cases:

- $D = D_1 \wedge D_2$. Then there exist $f_1 \subseteq m$ and $f_2 \subseteq m$ finite with

  $$\mathcal{I}(f_1), \alpha \models D_1 \quad \text{and} \quad \mathcal{I}(f_2), \alpha \models D_2$$

  by induction hypotheses. Let $f = f_1 \cup f_2$. Clearly, f is finite with $f \subseteq m$. By 3.16, we know that $\mathcal{I}(f), \alpha \models D_1$ and $\mathcal{I}(f), \alpha \models D_2$ Hence, $\mathcal{I}(f), \alpha \models D_1 \wedge D_2$

- $D = D_1 \vee D_2$. Similar to the above case.

- $D = \exists x D'$. Then there exists a α' such that α and α' differ only on x and

  $$\mathcal{I}(m), \alpha' \models D'$$

  By induction hypotheses, there is a finite $f \subseteq m$ with $\mathcal{I}(f), \alpha' \models D'$, which implies $\mathcal{I}(f), \alpha \models \exists x D'$.

□

Lemma 3.3 *$T_P^{\mathfrak{M}}$ is a continuous function for every definite program P.*

Proof. By Lemma 3.15, we have to show that for every $m \in \text{intdom}(\mathcal{R}, \mathfrak{M})$,

$$T_P^{\mathfrak{M}}(m) \;=\; \bigcup\{f \in \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \mid f \subseteq m, f \text{ finite}\}$$

The $\supseteq$-direction follows from the monotonicity of $T_P^{\mathfrak{M}}$ (Corollary 3.1). For the $\subseteq$-direction let $T_p^{\mathfrak{M}}(m) = (m'_1, \ldots, m'_n)$ and let $(\sigma_1, \ldots, \sigma_{n_i})$ be an element of m'_i. We have to show that there is a finite $f \subseteq m$ such that

$$(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T_P^{\mathfrak{M}}(f)),$$

where π_i is the i-th projection. Since P is a definite program, we know that there is a unique equivalence $R_i(x_1, \ldots, x_{n_i}) \leftrightarrow D$ in P. By definition of $T_P^{\mathfrak{M}}$, there exists a valuation α into $\mathcal{I}(m)$ such that $(\sigma_1, \ldots, \sigma_{n_i}) = (\alpha(x_1), \ldots, \alpha(x_{n_i}))$ and $\mathcal{I}(m), \alpha \models D$. By Lemma 3.2, there exists a finite $f \subseteq m$ with $\mathcal{I}(f), \alpha \models D$. Hence, $(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T_P^{\mathfrak{M}}(f))$. □

Corollary 3.2 *Every definite program has a least and a greatest model.*

Proof. Follows directly from the above lemma. □

A **goal** G is a possible empty conjunction of F′-constraints and $\mathcal{R}$(F′)-atoms. Considering the least model of a definite program P is the same as asking whether P entails $\tilde{\exists}G$. One has to choose the greatest model if one is interested in whether $P \wedge \tilde{\exists}G$ is consistent.

Proposition 3.17 *Let G be some goal and P be a definite program. Then $\tilde{\exists}G$ is valid in every model of P extending $\mathfrak{M}$ if and only if it is valid in the least model of P extending $\mathfrak{M}$, and $\tilde{\exists}G$ is satisfiable in every model of P extending $\mathfrak{M}$ if and only if it is valid in the greatest model of P extending $\mathfrak{M}$.*

3.2.2 Fixpoints of Continuous Functions

In this section, we show that for every continuous function

$$T : \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \to \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$$

which is definable within F′, the least and (under certain restrictions) the greatest fixpoint is definable. In the following the letter T denotes always a continuous function over $\prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$. Since the models of definite programs are continuous function of this type, we can use the encoding also for the least and greatest models of definite programs.

Definition 3.7 *Let* $T : \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \to \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ *be some function. A family of* F′*-formulae*

$$(\phi_{T,i}(M_1, \ldots, M_n, x_1, \ldots, x_{n_i}))_{i=1..n}$$

is called definition for T in $\mathfrak{M}$ in terms of F′ *iff for every* $i \in 1 \ldots n$ *and every valuation* α *into* $\mathfrak{M}$*,*

$$\begin{aligned}\mathfrak{M}, \alpha \models \phi_{T,i}(M_1, \ldots, M_n, x_1, \ldots, x_{n_i}) \iff \\ (\alpha(x_1), \ldots, \alpha(x_{n_i})) \in \pi_i(T(REL_{n_1}(\alpha(M_1)), \ldots, REL_{n_n}(\alpha(M_n)))),\end{aligned}$$

where π_i *is the* i^{th} *projection function.*

We say that T is **definable in** F′ if there exists a definition for T in terms of F′ in both feature tree models.

Lemma 3.4 *The function* $T_P^{\mathfrak{M}}$ *is definable in* F′ *for every definite program* P*.*

Proof. For a $\mathcal{R}$(F′)-formula ψ, we define the ψ^{repl} to be the F′-formula where every occurrence of an atomic formula $R_i(t_1, \ldots, t_n)$ in ψ is replaced by $\texttt{in}(\langle t_1, \ldots, t_n \rangle, M_i)$. Let $P = \{R_i(x_1, \ldots, x_{n_i}) \leftrightarrow D_i \mid i \in 1 \ldots n\}$. Then defining

$$\phi_{T,i}(M_1, \ldots, M_n, x_1, \ldots x_{n_i}) \quad := \quad D_i^{\text{repl}},$$

for every $i \in 1 \ldots n$ yields a $\mathfrak{M}$-definition for T. □

We start with the F′-definition for the least fixpoint. The definition for the greatest fixpoints will follow as an easy corollary. For the encoding of the least fixpoint, we

could apply either Tarski's fixpoint theorem or Kleene's fixpoint theorem. The first states that the least fixpoint $\mathrm{lfp}(T)$ of a continuous function T is the greatest lower bound of all pre-fixpoints, i.e.,

$$\mathrm{lfp}(T) \;=\; \bigcap\{m \mid T(m) \subseteq m\}. \tag{3.3}$$

The second fixpoint theorem constructs the fixpoint via an ω-iterative application of T to the least element $\bot$. Since the least element of $\prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ is $(\emptyset,\ldots,\emptyset)$, we get

$$\mathrm{lfp}(T) \;=\; T\uparrow\omega \;=\; \bigcup\{T^k((\emptyset,\ldots,\emptyset)) \mid k \in \mathbb{N}\}. \tag{3.4}$$

The characterisation of $\mathrm{lfp}(T)$ in (3.3) would yield the simplest definition for $\mathrm{lfp}(T)$ since it uses a direct representation of the least fixpoint. The formula

$$\begin{aligned}&\mathtt{prefixp}_T(M_1,\ldots,M_n) :=\\ &\qquad \bigwedge_{i=1}^{n} \forall x_1,\ldots,x_{n_i}(\phi_{T,i}(M_1,\ldots,M_n,x_1,\ldots,x_{n_i}) \to \mathtt{in}(\langle x_1,\ldots,x_{n_i}\rangle, M_i)),\end{aligned}$$

where $(\phi_{T,i}(M_1,\ldots,M_n,x_1,\ldots,x_{n_i}))_{i\in 1\ldots n}$ is a definition for T within F′, expresses that the tuple $M_1,\ldots,M_n$ represent a pre-fixpoint of T. Since the subset relation is definable in F′, we could also define the lower bound off all pre-fixpoints, and hence the least fixpoint of T. But this definition works only if all pre-fixpoints are representable in $\mathfrak{M}$, which is not normally not the case.

This implies that we must use (3.4) for the definition of the least fixpoint. Our aim is to find a family of formulae $(\phi_i^{\mathrm{lfp}(T)}(\vec{x_i}))_{i=1..n}$ such that for every valuation α,

$$\mathfrak{M},\alpha \models \phi_i^{\mathrm{lfp}(T)}(x_1,\ldots,x_{n_i}) \quad\Longleftrightarrow\quad (\alpha(x_1),\ldots,\alpha(x_{n_i})) \in \pi_i(T\uparrow\omega).$$

Since the definition shall also work for the case $\mathfrak{M} = \mathfrak{R}_{\mathrm{F}'}$, we impose the additional restriction that is sufficient for the set variables in the formulae $\phi_i^{\mathrm{lfp}(T)}(x_1,\ldots,x_{n_i})$ to denote finite sets.

The restriction to finite sets is the minor problem. By Proposition 3.15 we know that $(\sigma_1,\ldots,\sigma_{n_i}) \in \pi_i(T(m))$ if there is a finite $f \subseteq m$ with $(\sigma_1,\ldots,\sigma_{n_i}) \in \pi_i(T(f))$. Furthermore, $(\sigma_1,\ldots,\sigma_{n_i}) \in \pi_i(\mathrm{lfp}(T))$ if there is some $k \in \mathbb{N}$ such that $(\sigma_1,\ldots,\sigma_{n_i}) \in \pi_i(T^k((\emptyset,\ldots,\emptyset))$. Hence, an easy induction shows that $(\sigma_1,\ldots,\sigma_{n_i}) \in \pi_i(\mathrm{lfp}(T))$ iff there is a finite sequence

$$(m_1^0,\ldots,m_{n_i}^0),\ \ldots\ ,(m_1^k,\ldots,m_n^k)$$

of finite elements $(m_1^l,\ldots,m_n^l)$ of $\prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ such that

- $(m_1^0, \ldots, m_n^0) = (\emptyset, \ldots, \emptyset)$,
- $(m_1^{k+1}, \ldots, m_n^{k+1}) \subseteq T(m_1^k, \ldots, m_n^k)$ for every $l \in 0 \ldots k-1$, and
- $(\sigma_1, \ldots, \sigma_{n_i}) \in m_i^k$.

Note that the restriction to finite elements is a sufficient but not necessary condition (which is important for the case $\mathfrak{M} = \mathfrak{T}_{\mathrm{F}'}$, since we can represent also infinite sets in $\mathfrak{T}_{\mathrm{F}'}$).

Since we may only use a finite number of variables denoting sets, we have to find an appropriate representation of sequences. Given a sequence $(m_1^0, \ldots, m_n^0), \ldots, (m_1^k, \ldots, m_n^k)$, we can represent this sequence using a single tuple $(s_1, \ldots, s_n)$ if we extend each element of m_i^l by the number l:

$$s_i = \{(\sigma_1, \ldots, \sigma_{n_i}, l) \mid (\sigma_1, \ldots, \sigma_{n_i}) \in m_i^l\}$$

For a set s containing tuples of arity $n_i + 1$, we say that a set $m \in \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ is the **restriction** s **to** l if

$$m = \{(\sigma_1 \ldots, \sigma_{n_i}) \mid (\sigma_1 \ldots, \sigma_{n_i}, l) \in s\}.$$

A tuple $(m_1, \ldots, m_n)$ is the restriction of $(s_1, \ldots, s_n)$ to l if every m_i is the restriction of s_i to l. We say that a sequence $(m_1^0, \ldots, m_n^0), \ldots, (m_1^k, \ldots, m_n^k)$ of elements of $\prod_{i=1}^n \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ is **associated** to a tuple $(s_1, \ldots, s_n)$ iff $(m_1^l, \ldots, m_n^l)$ is the restriction of $(s_1, \ldots, s_n)$ to l for every $l = 0..k$. Note that for every $k \in \mathbb{N}$ there is a sequence associated to $(s_1, \ldots, s_n)$.

In the following, we use the variables $S_1, \ldots, S_n$ to encode the tuple $(s_1, \ldots, s_n)$. Instead of natural numbers we use the feature tree representation of natural numbers as introduced in Section 3.1.3. Note that there is a one-to-one mapping of natural numbers to the feature tree representation of natural numbers. Hence, we can represent sequences as tuples $(s_1, \ldots, s_n)$ with

$$(s_1, \ldots, s_n) \in \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i} \times \mathtt{nat}^{\mathfrak{M}}),$$

where $\mathtt{nat}^{\mathfrak{M}}$ denotes the set of feature trees representing natural numbers.

To show that $S_1, \ldots, S_n$ is a correct encoding of an iterative application of T, we must find a formula guaranteeing that every sequence $m^0, \ldots, m^k$ associated to the denotation of $S_1, \ldots, S_n$, satisfies $m^{l+1} \subseteq T(m^l)$ for every $l \in 1 \ldots k-1$. For this purpose we need to express that the sets denoted by the variables $M_1, \ldots, M_n$ are

the restriction of the sets encoded by $S_1, \ldots, S_n$ to a given natural number encoded by the variable C:

$$\begin{aligned}&\mathtt{restr}(M_1, \ldots, M_n, S_1, \ldots, S_n, C) \ :=\\ &\qquad \bigwedge_{i=1}^{n} \forall x_1, \ldots, x_{n_i} (\mathtt{in}(\langle x_1, \ldots, x_{n_i}\rangle, M_i) \leftrightarrow \mathtt{in}(\langle x_1, \ldots, x_{n_i}, C\rangle, S_i)))\end{aligned}$$

Proposition 3.18 *For every valuation α into $\mathfrak{M}$, if*

$$\mathfrak{M}, \alpha \models \mathtt{restr}(M_1, \ldots, M_n, S_1, \ldots, S_n, C]),$$

then $REL_{n_i}(\alpha(M_i))$ is the restriction of $REL_{n_i+1}(\alpha(S_i))$ to $NUM(\alpha(C))$.

The next formula $\mathtt{seq}_T(S_1, \ldots, S_n)$ guarantees that every sequence $m^0, \ldots, m^k$ of elements of $\prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ associated to the interpretation of $S_1 \ldots, S_n$ satisfies

$$m^{l+1} \subseteq T(m^l)$$

In the following, we assume that the formulae $(\phi_{T,i}(M_1, \ldots, M_n, x_1, \ldots, x_{n_i}))_{i=1..n}$ are a definition for T within F′.

The formula $\mathtt{app}_T(S_1, \ldots, S_n, x_1, \ldots, x_{n_i}, C)$ defined as

$$\exists M_1, \ldots, M_n \begin{pmatrix} \mathtt{restr}(M_1, \ldots, M_n, S_1, \ldots, S_n, C) \wedge \\ \phi_{T,i}(M_1, \ldots, M_n, x_1, \ldots, x_{n_i}) \end{pmatrix}$$

states that $(x_1, \ldots, x_{n_i})$ is an element of the application of T to the restriction of $S_1, \ldots, S_n$ to C. Thus, we can write $\mathtt{seq}_T(S_1, \ldots, S_n)$ as

$$\bigwedge_{i=1}^{n} \forall x_1, \ldots, x_{n_i}, C \left(\begin{array}{l} \mathtt{in}(\langle x_1, \ldots, x_{n_i}, C\rangle, S_i) \wedge \mathtt{nat}(C) \rightarrow \\ \qquad C = 0 \ \vee \ (C \neq 0 \wedge \mathtt{app}_T(S_1, \ldots, S_n, x_1, \ldots, x_{n_i}, C - 1)) \end{array} \right)$$

Proposition 3.19 *If $\mathfrak{M}, \alpha \models \mathtt{seq}_T(S_1, \ldots, S_n)$, then every sequence $m^0, \ldots, m^k$ associated to $(REL_{n_1+1}(\alpha(S_1)), \ldots, REL_{n_n+1}(\alpha(S_n)))$ satisfies $m^{l+1} \subseteq T(m^l)$ for every $l \in 0 \ldots k-1$.*

Lemma 3.5 *Let $T : \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \rightarrow \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ be a continuous function which is definable in F′. Then there is a family of formulae*

$$(\psi_i(M_1, \ldots, M_n, C, x_1, \ldots, x_{n_i}))_{i=1..n}$$

such that for all valuations α into $\mathfrak{M}$,

$$\mathfrak{M}, \alpha \models \psi_i(M_1, \ldots, M_1, C, x_1, \ldots, x_{n_i})$$
$$\Longleftrightarrow$$
$$(\alpha(x_1), \ldots, \alpha(x_{n_i})) \in \pi_i(T^{NUM(\alpha(C))}((m_1, \ldots, m_n)))$$

where $(m_1, \ldots, m_n) = (REL_{n_1}(\alpha(M_1)), \ldots, REL_{n_n}(\alpha(M_1)))$.

Proof. It is easy to check that the family of formulae

$$\psi_i \quad := \quad \exists S_1, \ldots, S_n, \left(\begin{array}{l} \mathtt{nat}(C) \wedge \\ \mathtt{in}(\langle x_1, \ldots, x_{n_i}, C\rangle, S_i) \wedge \\ \mathtt{seq}_T(S_1, \ldots, S_n) \wedge \\ \mathtt{restr}(M_1, \ldots, M_n, S_1, \ldots, S_n, 0) \end{array} \right)$$

satisfies the required properties. □

Using this proposition, we can define the least and (under certain restrictions) the greatest fixpoint of a continuous function.

Theorem 3.1 *For any continuous function* $T : \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \to \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ *which is definable in* F′ *there is a family of formulae*

$$(\phi_i^{\mathrm{lfp}(T)}(x_1, \ldots, x_{n_i}))_{i=1..n}$$

such that $\mathfrak{M}, \alpha \models \phi_i^{\mathrm{lfp}(T)}(x_1, \ldots, x_{n_i})$ *iff* $(\alpha(x_1), \ldots, \alpha(x_{n_i})) \in \pi_i(\mathrm{lfp}(T))$.

Proof. By Proposition A.1 we know that $\mathrm{lfp}(T) = T \uparrow \omega$. This implies that for every tuple $(\sigma_1, \ldots, \sigma_{n_i})$ of feature trees, $(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(\mathrm{lfp}(T))$ iff there is some $k \in \mathbb{N}$ with $(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T^k(\emptyset, \ldots, \emptyset))$. By lemma 3.5 there is a formula $\psi_i(M_1, \ldots, M_n, C, x_1, \ldots, x_{n_i})$ such that for all valuations α into $\mathfrak{M}$,

$$\mathfrak{M}, \alpha \models \psi_i(M_1, \ldots, M_1, C, x_1, \ldots, x_{n_i})$$
$$\Longleftrightarrow$$
$$(\alpha(x_1), \ldots, \alpha(x_{n_i})) \in \pi_i(T^{NUM(\alpha(C))}((REL_{n_1}(\alpha(M_1)), \ldots, REL_{n_n}(\alpha(M_1))))).$$

Since for every k there is a feature tree representing k, we can define $\phi_i^{\mathrm{lfp}(T)}(x_1, \ldots, x_{n_i})$ as $\exists C\, \psi_i(eset, \ldots, eset, C, x_1, \ldots, x_{n_i})$, where *eset* is an arbitrary constant symbol in $\mathcal{L}$. □

Theorem 3.2 *Let* $T : \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i}) \to \prod_{i=1}^{n} \wp(\mathbf{U}(\mathfrak{M})^{n_i})$ *be a continuous function which is definable in* F′ *If* $\mathrm{gfp}(T) = T \downarrow \omega$*, then there is a family of formulae*

$$(\phi_i^{\mathrm{gfp}(T)}(x_1, \ldots, x_{n_i}))_{i=1..n}$$

such that $\mathfrak{M}, \alpha \models \phi_i^{\mathrm{gfp}(T)}(x_1, \ldots, x_{n_i})$ *iff* $(\alpha(x_1), \ldots, \alpha(x_{n_i})) \in \pi_i(\mathrm{gfp}(T))$.

Proof. By the monotonicity of T we know that $(T^k(\prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})^{n_i}))_{k \in \mathbb{N}}$ is a decreasing sequence of sets, i.e.

$$T^{k+1}(\prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})^{n_i}) \subseteq T^k(\prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})^{n_i}).$$

Hence,a tuple $(\sigma_1, \ldots, \sigma_{n_i})$ of feature trees is an element of $\pi_i(T \downarrow \omega)$ if and only if for every $k \in \mathbb{N}$

$$(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T^k(\prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})^{n_i})).$$

This implies by Proposition 3.15 that $(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T \downarrow \omega)$ iff for every $k \in \mathbb{N}$ there is a *finite* $m \in \prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})^{n_i}$ with $(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T^k(m))$. The restriction to finite elements of $\prod_{i=1}^{n} \mathbf{U}(\mathfrak{M})^{n_i}$ is not necessary, but sufficient.

Using the formulae $\psi_i(M_1, \ldots, M_n, C, x_1, \ldots, x_{n_i})$ of lemma 3.5, we can define the relation $(\sigma_1, \ldots, \sigma_{n_i}) \in \pi_i(T \downarrow \omega)$ by the formula

$$\phi_i^{\mathrm{gfp}(T)}(x_1, \ldots, x_{n_i}) \quad := \quad \forall C \exists M_1, \ldots, M_n(\mathtt{nat}(C) \to \psi_i(M_1, \ldots, M_n, C, x_1, \ldots, x_{n_i})).$$

□

Using these theorems, we can show the definability of the least and greatest model of definite programs.

Theorem 3.3 *The relations defined by the least model* $\mathfrak{A}_P^{\perp}$ *of a definite program* P *extending* $\mathfrak{M}$ *are definable in* F′ *i.e., there are formulae* $(\phi_{R_i}(x_1, \ldots, x_{n_i}))_{\in 1 \ldots n}$ *such that*

$$\mathfrak{M}, \alpha \models \phi_{R_i}(x_1, \ldots, x_{n_i}) \quad \Longleftrightarrow \quad (\alpha(x_1), \ldots, \alpha(x_{n_i})) \in R_i^{\mathfrak{A}_P^{\perp}}$$

If the greatest fixpoint of the associated continuous function $T_P^{\mathfrak{M}}$ *can be generated in* ω*-many steps (i.e.,* $\mathrm{gfp}(T_P^{\mathfrak{M}}) = T \downarrow \omega$*), then there exist formulae* $(\psi_{R_i}(x_1, \ldots, x_{n_i}))_{\in 1 \ldots n}$ *such that*

$$\mathfrak{M}, \alpha \models \psi_{R_i}(x_1, \ldots, x_{n_i}) \quad \Longleftrightarrow \quad (\alpha(x_1), \ldots, \alpha(x_{n_i})) \in R_i^{\mathfrak{A}_P^{\top}}$$

Proof. Follows from Proposition 3.14, Lemmas 3.3 and 3.4, and Theorems 3.1 and 3.2. □

Chapter 4

Recursive Axiomatisations of FT′ and CFT′

There are two complementary ways of specifying a feature theory: either by explicitly constructing a standard interpretation and taking all sentences valid in it, or by stating axioms and proving their consistency. Both approaches to fixing a feature theory have their advantages. The construction of a standard interpretation provides for a clear intuition and yields a complete theory (i.e., if ϕ is a closed feature description, then either ϕ or $\neg\phi$ is a consequence of the theory). The presentation of a recursively enumerable axiomatisation has the advantage that we inherit from predicate logic a sound and complete deduction system for valid feature descriptions. Note that all models of a complete theory are elementarily equivalent. The ideal is to specify a feature theory by both a standard interpretation and a corresponding recursively enumerable axiomatisation. The existence of such a double characterisation, however, is by no means obvious, since it implies that the theory is decidable.

In the first two chapters, we have exemplified the first approach by defining the feature tree interpretation of the language F′ and investigating the theory of this interpretation. Since we have shown that this theory is undecidable, it is clear that a recursive enumerable axiomatisation of this theory cannot exists. In this chapter, we present an axiomatisation for the theory of standard interpretations of FT′ and CFT′, which we have defined in the first chapter. In the following, we deliberately confuse the language FT′ (resp. CFT′) with the axiomatisation of the theory of $\mathfrak{T}_{\mathrm{FT}'}$ (resp. $\mathfrak{T}_{\mathrm{CFT}'}$).

In Section 4.1, we give an informal guide through the completeness proofs and compare the proofs for FT′ and CFT′. Section 4.2 formally introduces the method for proving completeness. In Section 4.3, we define the tool of path constraints which

helps us to keep the completeness proofs simple. Section 4.4 presents the axiomatisation for FT′ and proves it completeness. Furthermore, we show that the feature graph interpretation and both the infinite and rational feature tree interpretation are models of FT′. This implies that all these models are elementarily equivalent with respect to the language of FT′. In this section, we will also show that the arity predicates of CFT′ are not definable in FT′. As an example, we will show how the quantifier elimination can be used for checking entailment of existentially quantified formulae containing negated equations. Section 4.5 finally introduces the theory CFT′ and proves it completeness. Again we will show that both the rational and infinite tree interpretation are models of CFT′. Note that the feature tree graph interpretation is not a model of CFT′.

4.1 The Method

4.1.1 Quantifier Elimination

The completeness proofs use a version of the method of quantifier elimination used by [Mah88]. To proof the completeness of a theory T using this method, it is necessary to find a class of formulae (here called prime formulae) satisfying certain properties. For both FT′ and CFT′, the set of prime formulae is the set of existentially quantified solved formulae. A solved formula is a normal form of conjunction of atomic formulae with some nice properties. In particular, it is always satisfiable.

The first property is that every closed prime formula is valid in T, which will be a trivial consequence of the set of axioms. The second property is that the class of prime formulae is closed under conjunction and existential quantification. Again, this is easy to show in our case.

The third (and difficult to prove) property is that the following two equivalences are valid in T: (1) Given prime formulae $\beta, \beta_1, \ldots, \beta_n$, then

$$\exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\beta_i) \models\!\!\!\models \bigwedge_{i=1}^{n} \exists X(\beta \wedge \neg\beta_i), \tag{4.1}$$

and (2) there exists for all prime formulae β, β' a Boolean combination of prime formulae δ such that

$$\exists X(\beta \wedge \neg\beta') \models\!\!\!\models \delta, \tag{4.2}$$

where X is a set of variables. These schemes can now be used for a system transforming every closed formula in the language of T a Boolean combination of solved

prime formulae. Since every closed prime formula is valid in T, we know that the result of transforming the sentence reduces either to $\top$ or to $\bot$. In the first case, ϕ is valid in T. Otherwise, $\neg\phi$ is valid in T.

The transformation works as follows. An invariant of the transformation is that both the input and output formulae of a single transformation step are of the form

$$Q_1 \ldots Q_n \gamma$$

where $Q_1 \ldots Q_n$ are quantifiers and γ is a Boolean combination of prime formulae. A single transformation step now eliminates the innermost quantifier.

If the innermost quantifier Q_n is an existential one, then we first transform γ into disjunctive normal form, treating the prime formulae as atoms. Then we can distribute the existential quantifier over the disjuncts, yielding a disjunction of formulae of the form

$$\exists x(\bigwedge_{i=1}^{n} \beta_i \wedge \bigwedge_{j=1}^{k} \neg\beta'_j)$$

where all β_i and β'_j are prime formulae. Since prime formulae are closed under conjunction, we can assume that the disjuncts are of the form

$$\exists x(\beta \wedge \bigwedge_{j=1}^{k} \neg\beta'_j.)$$

Now we can apply scheme 4.1, transforming each disjunct into a conjunction of the form

$$\bigwedge_{j=1}^{k} \exists x(\beta \wedge \neg\beta'_j),$$

which can be transformed into a Boolean combination of prime formulae δ by scheme 4.2. All together, we have eliminated the innermost existential quantifier.

In the second case that the innermost quantifier is a universal one, we substitute $\neg\exists x\neg\gamma$ for $\forall x\gamma$. Then we put $\neg\gamma$ into its negation normal form γ', treating the prime formulae as atoms. Applying the elimination method as described for existential quantification on $\exists x\gamma'$ yields a Boolean combination of prime formulae δ. Now putting $\neg\delta$ into negation normal form again (treating prime formulae as atoms) yields a Boolean combination of prime formulae that is equivalent to $\forall x\gamma$.

We have described the elimination of a single quantifier. But as the schemes 4.1 and 4.2 use an existential quantification over a whole set of variables X, the elimination methods apply also to a whole set of quantifiers of the same type (i.e., if we start with a formula $Q_1 \ldots Q_k \ldots Q_{k+n}\gamma$ where $Q_k \ldots Q_{k+n}$ are either of the form $\exists x_k \ldots \exists x_{k+n}$ or of the form $\forall x_k \ldots \forall x_{k+n}$, then we can eliminate $Q_k \ldots Q_{k+n}$ in one step.

4.1.2 Comparison of the Completeness Proofs for FT′ and CFT′

The difference between FT′ and CFT′ is that CFT′ additionally has arity constraints. This implies that every FT′ formula is also a CFT′ formula. Hence, the completeness proof for CFT′ is an extension of the completeness proof of FT′. However, we have to extend the completeness proof for FT′ in a non-trivial way, since we have to handle additional equations imposed by the arity constraints. E.g.

$$x\{f\} \wedge xfx \wedge y\{f\} \wedge yfy \models_{\mathrm{CFT}'} x \doteq y$$

holds in CFT′ stating that there is only one solution for the formula $x\{f\} \wedge xfx$. In FT′ it is not possible to identify one element of the domain by a formula. Thus, CFT′ requires records to be extensional (i.e., two records are identical if they have the same set of attributes and identical values under the corresponding attributes). Note that this property could not be guaranteed using the language of FT′ (i.e., FT′ has non-extensional models).

In the following we give a concrete example where the additional problems that arise in the completeness proof for CFT′. Consider the FT′-formula $\exists x(\beta \wedge \neg\beta')$ with

$$\begin{aligned} \beta &:= \exists x_1, x_2(xfx_1 \wedge xgx_2) \\ \beta' &:= \exists y(xfy \wedge xgy), \end{aligned}$$

which is an instance of left hand side of scheme 4.2. In the standard model of FT′ (which is the same as for CFT′), there always exists a valuation for x satisfying β such that the values under the features f and g are different. This implies that the equivalence

$$\exists x(\beta \wedge \neg\beta') \;\models\!\!\models\; \exists x\beta \tag{4.3}$$

is valid in FT′. Hence, $\exists x\beta$ is the Boolean combination of prime formulae as required by scheme 4.2. Roughly speaking, this equivalence is proven by extending β to a prime formula β_{ext} which makes x_1 and x_2 different, e.g. the prime formula

$$\exists x_1, x_2 \left(\; xfx_1 \wedge xgx_2 \wedge x_1fa \wedge x_2fa' \; \right)$$

with a, a' being two different constant symbols. Clearly, $\exists x\beta_{ext}$ is satisfiable in FT′. Hence, there exists in every model of FT′ a valuation for x satisfying β_{ext}. Since this valuation must also satisfy β and cannot satisfy β', this shows the equivalence in 4.3.

Therefore, it is necessary in the proof to characterise the variables for which such additional constraints must be added. In the case of FT′ this is easy; they are

exactly the variables where an additional equation is added when applying the solved form algorithm on

$$\beta \wedge \beta' \quad = \quad \exists x_1, x_2, y(xfx_1 \wedge xgx_2 \wedge xfy \wedge xgy).$$

But in the case of CFT′, the situation is more complex, since variables can be determined using the arity constraints. Consider the following two formulae β_1 and β_2:

$$\beta_1 \quad = \quad \exists x_1, x_2, x_3, x_4 \left(\begin{array}{l} xfx_1 \wedge xgx_2 \wedge \\ x_1\{f\} \wedge x_1fx_3 \wedge \\ x_2\{f\} \wedge x_2fx_4 \end{array} \right)$$

$$\beta_2 \quad = \quad \exists x_1, x_2 \left(\begin{array}{l} xfx_1 \wedge xgx_2 \wedge \\ x_1\{f\} \wedge x_1fx_1 \wedge \\ x_2\{f\} \wedge x_2fx_2 \end{array} \right)$$

We let β' again be $\exists y(xfy \wedge xgy)$. Although in both cases an additional equation $x_1 \doteq x_2$ is added when solving $\beta_1 \wedge \beta'$ or $\beta_2 \wedge \beta'$, the equivalence $\exists x(\beta_1 \wedge \neg\beta') \models\!\!\!\models \exists x\beta_1$ is valid in CFT′, whereas the equivalence $\exists x(\beta_2 \wedge \neg\beta') \models\!\!\!\models \exists x\beta_2$ is not.

4.2 Overall Structure of the Completeness Proofs

The completeness of FT′ and CFT′ will be shown by exhibiting simplification algorithms for both FT′ and CFT′. The following lemma gives the overall structure of the algorithms, which is the same as in Maher's [Mah88] completeness proof for the theory of constructor trees.

Lemma 4.1 *Let T be a first order theory. Suppose there exists a set of prime formulae such that:*

1. *for every atomic formula ϕ one can compute a Boolean combination δ of prime formulae such that*
$$\phi \models\!\!\!\models_T \delta \quad \text{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\phi),$$
2. *$\top$ is a prime formula, and there is no other closed prime formula*
3. *for every two prime formulae β and β' one can compute a formula δ that is either prime or $\bot$ and satisfies*
$$\beta \wedge \beta' \models\!\!\!\models_T \delta \quad \text{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\beta \wedge \beta')$$

4. *for every prime formula β and every variable x one can compute a prime formula β' such that*

$$\exists x\beta \models\!\!\!\models_T \beta' \quad \text{and} \quad \mathcal{V}(\beta') \subseteq \mathcal{V}(\exists x\beta)$$

5. *if β, $\beta_1, \ldots, \beta_n$ are prime formulae, then*

$$\exists x(\beta \wedge \bigwedge_{i=1}^{n} \neg\beta_i) \models\!\!\!\models_T \bigwedge_{i=1}^{n} \exists x(\beta \wedge \neg\beta_i)$$

6. *for every two prime formulae β, β' and every variable x one can compute a Boolean combination δ of prime formulae such that*

$$\exists x(\beta \wedge \neg\beta') \models\!\!\!\models_T \delta \quad \text{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\exists x(\beta \wedge \neg\beta')).$$

Then one can compute for every formula ϕ a Boolean combination δ of prime formulae such that $\phi \models\!\!\!\models_T \delta$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\phi)$.

Proof. Suppose a set of prime formulae exists as required. Let ϕ be a formula. We show by induction on the structure of ϕ how to compute a Boolean combination δ of prime formulae such that $\phi \models\!\!\!\models_T \delta$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\phi)$.

If ϕ is an atomic formula, then it can be transformed into an equivalent Boolean combination of prime formulae by assumption (1).

If ϕ is $\neg\psi$, $\psi \wedge \psi'$ or $\psi \vee \psi'$, then the claim follows immediately with the induction hypothesis.

It remains to show the claim for $\phi = \exists x\psi$. By the induction hypothesis we know that we can compute a Boolean combination δ of prime formulae such that $\delta \models\!\!\!\models_T \psi$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\psi)$. Now δ can be transformed to a disjunctive normal form where prime formulae play the role of atomic formulae; that is, δ is equivalent to $\sigma_1 \vee \ldots \vee \sigma_n$, where every "clause" σ_i is a conjunction of prime and negated prime formulae. Hence

$$\exists x\psi \models\!\!\!\models \exists x(\sigma_1 \vee \ldots \vee \sigma_n) \models\!\!\!\models \exists x\sigma_1 \vee \ldots \vee \exists x\sigma_n,$$

where all three formulae have exactly the same free variables. It remains to show that one can compute for every clause σ a Boolean combination δ of prime formulae such that $\exists x\sigma \models\!\!\!\models_T \delta$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\exists x\sigma)$. We distinguish the following cases.

(i) $\sigma = \beta$ for some basic constraint β. Then the claim follows by assumption (4).

(ii) $\sigma = \beta \wedge \bigwedge_{i=1}^{n} \neg\beta_i$, $n > 0$. Then the claim follows with assumptions (5) and (6).

(iii) $\sigma = \bigwedge_{i=1}^{n} \neg\beta_i$, $n > 0$. Then $\sigma \models\!\!\!\models_T \top \wedge \bigwedge_{i=1}^{n} \neg\beta_i$ and the claim follows with case (ii) since $\top$ is a prime formula by assumption (2).

(iv) $\sigma = \beta_1 \wedge \ldots \wedge \beta_k \wedge \neg\beta'_1 \wedge \ldots \wedge \neg\beta'_n$, $k > 1$, $n \geq 0$. Then we know by assumption (3) that either $\beta_1 \wedge \ldots \wedge \beta_k \models\!\!\!\models_T \bot$ or $\beta_1 \wedge \ldots \wedge \beta_k \models\!\!\!\models_T \beta$ for some prime formula β. In the former case we choose $\delta = \neg\top$, and in the latter case the claim follows with case (i) or (ii).

□

Note that, provided a set of prime formulae with the required properties exists for FT′ (resp. CFT′), the preceding lemma yields the completeness of FT′ (resp. CFT′) since every closed formula can be simplified to $\top$ or $\neg\top$ (since $\top$ is the only closed prime formula).

Section 4.4 establishes the set of prime formulae as required for FT′, whereas in section 4.5 we will do the same for CFT′. In the next section we will define the tool of path constraints that will help us to keep the proofs for both FT′ and CFT′ simple.

4.3 Path Constraints

Path constraints are a flexible syntax for atomic formulae closed under conjunction and existential quantification. We will see that for every prime formula there is an equivalent quantifier-free formula consisting only of path constraints.

The propositions to come will be proven for a common sub-theory of FT′ and CFT′, which expresses exactly the minimal properties of features. This theory is called FT_0. In a slightly modified version, this theory has been introduced in [Smo92]. Since CFT′ contains unary predicates in form of arity constraints, FT_0 must also contain unary predicates. Thus, the signature of the common sub-theory consists of all elements of $\mathcal{L}$ acting both as constants and features, a set of unary predicates $\mathcal{P}$ and a distinguished unary predicate symbol *atom*. If nothing else is stated, the letter P denotes a unary predicate symbol in $\mathcal{P} \cup \{atom\}$. Since CFT′ contains unary predicates, whereas FT′ does not, we formalise the theory FT_0 with different sets of unary predicates. Thus, we use $\mathrm{FT}_0^{\mathcal{P}}$ to denote the instance of FT_0 whose signature contains exactly the unary predicate symbols listed in $\mathcal{P}$. We will see that $\mathrm{FT}_0^{\{\}}$ is a subset of FT′ and that $\mathrm{FT}_0^{\{F_i\}_{i\in\mathbb{N}}}$ is a subset of CFT′, where $\{F_i\}_{i\in\mathbb{N}}$ is an enumeration of all arity constraints.

As mentioned, $\mathrm{FT}_0^{\mathcal{P}}$ is a minimal theory expressing the properties of features. In the models of $\mathrm{FT}_0^{\mathcal{P}}$, features are interpreted as binary, functional relations. All predicates in $\mathcal{P}$ are free, i.e. they are just interpreted as unary predicates without additional conditions. Therefore, we have the following axiom schemes for $\mathrm{FT}_0^{\mathcal{P}}$ stating that every feature is functional, that features are not defined on constants, that the unique name assumption holds for the constants, and that the denotation of *atom* should be the set of all constants[1]:

(Ax1)	$\tilde{\forall}(xfy \wedge xfz \rightarrow y \doteq z)$	for every feature f
(Ax2)	$\neg(c_1 \doteq c_2)$	if c_1 and c_2 are different constants
(Ax3)	$\tilde{\forall}(cfx \rightarrow \bot)$	for every constant c
(Ax4)	$atom(c)$	for every constant c
(Ax5)	$\tilde{\forall}(xfy \wedge atom(x) \rightarrow \bot)$	for every feature f.

Using $\mathrm{FT}_0^{\mathcal{P}}$ we are able to define path constraints. We start by recalling the definition of the denotation of a path.

The interpretations $f^{\mathfrak{A}}$, $g^{\mathfrak{A}}$ of two features f, g in a structure $\mathfrak{A}$ are binary relations on the universe $\mathbf{U}(\mathfrak{A})$ of $\mathfrak{A}$; hence their composition $f^{\mathfrak{A}} \circ g^{\mathfrak{A}}$ is again a binary relation on $\mathbf{U}(\mathfrak{A})$ satisfying

$$a(f^{\mathfrak{A}} \circ g^{\mathfrak{A}})b \iff \exists c \in \mathbf{U}(\mathfrak{A}): \; af^{\mathfrak{A}}c \; \wedge \; cg^{\mathfrak{A}}b$$

for all $a, b \in \mathbf{U}(\mathfrak{A})$. Consequently we define the **denotation $p^{\mathfrak{A}}$ of a path** $p = f_1 \cdots f_n$ in a structure $\mathfrak{A}$ as the composition

$$(f_1 \cdots f_n)^{\mathfrak{A}} := f_1^{\mathfrak{A}} \circ \cdots \circ f_n^{\mathfrak{A}},$$

where the empty path ε is taken to denote the identity relation. If $\mathfrak{A}$ is a model of the theory $\mathrm{FT}_0^{\mathcal{P}}$, then every path denotes a unary partial function on the universe of $\mathfrak{A}$. Given an element $a \in \mathbf{U}(\mathfrak{A})$, $p^{\mathfrak{A}}$ is thus either undefined on a or leads from a to exactly one $b \in \mathbf{U}(\mathfrak{A})$.

Definition 4.1 (Path Constraints) *Let p, q be paths, x, y be variables, P be a unary predicate symbol, and c be a constant symbol. Then* path constraints *are defined as follows:*

$$\mathfrak{A}, \alpha \models xpc \; :\iff \; \alpha(x)\, p^{\mathfrak{A}}\, c^{\mathfrak{A}}$$

[1]Note that this property can only be approximated. For guaranteeing this property, an infinite disjunction would be needed.

$$\begin{array}{rcl}
\mathfrak{A}, \alpha \models xpy & :\Longleftrightarrow & \alpha(x)\, p^{\mathfrak{A}}\, \alpha(y) \\
\mathfrak{A}, \alpha \models xp \downarrow yq & :\Longleftrightarrow & \exists a \in \mathbf{U}(\mathfrak{A}){:}\ \ \alpha(x)\, p^{\mathfrak{A}}\, a \ \wedge\ \alpha(y)\, q^{\mathfrak{A}}\, a \\
\mathfrak{A}, \alpha \models P(xp) & :\Longleftrightarrow & \exists a \in \mathbf{U}(\mathfrak{A}){:}\ \ \alpha(x)\, p^{\mathfrak{A}}\, a \ \wedge\ a \in P^{\mathfrak{A}}
\end{array}$$

A proper path constraint *is a path constraint of the form "xpc", "$P(xp)$" or "$xp{\downarrow}yq$".*

Note that path constraints xpy generalise feature constraints xfy. We use $xp{\downarrow}$ as a shortcut for $xp \downarrow xp$. By definition, $xp{\downarrow}$ is satisfied by some valuation α into some structure $\mathfrak{A}$ iff the path $p^{\mathfrak{A}}$ is defined on $\alpha(x)$.

Every path constraint can be expressed with the already existing formulae, as can be seen from the following equivalences:

$$\begin{array}{rcll}
x \varepsilon t & \models & x \doteq t & \\
xfpt & \models & \exists z(xfz \wedge zpt) & (z \neq x, t) \\
xp \downarrow yq & \models & \exists z(xpz \wedge yqz) & (z \neq x, y) \\
P(xp) & \models & \exists y(xpy \wedge P(y)) & (y \neq x).
\end{array}$$

We are now going to define the closure of quantifier-free and of existential quantified formulae. The closure of a formulae β is a set of all path constraints that is equivalent to β. In general, the closure can be infinite. But we will show that there is a finite subset that is also equivalent to β (which we will call projection).

We will define the closure only for special classes of formulae in order to guarantee some nice properties for the closure. In the case of quantifier-free formulae, this will be the class of solved formulae. In the case of existential quantified formulae, this will be the class of prime formulae. We will later prove that for the theory FT$'$ the prime formulae satisfies all condition required by lemma 4.1. For CFT$'$ we have to redefine the notions of solved formula in order to handle the arities predicates.

The framework of path constraints will later be used for proving claims 5 and 6 of Lemma 4.1. Recall the example of Section 4.1.2, page 62. There we considered the FT$'$-formulae

$$\begin{array}{rcl}
\beta \ = \ \exists x_1 \exists x_2\, \gamma & := & \exists x_1, x_2(xfx_1 \wedge xgx_2) \\
\beta' & := & \exists y(xfy \wedge xgy),
\end{array}$$

and argued that the equivalence

$$\exists x(\beta \wedge \neg \beta') \ \models\ \exists x \beta$$

is valid in FT′. This will be shown in the completeness proof for FT′ by extending β to a formula

$$\beta_{ext} := \exists x_1, x_2 \left(xfx_1 \wedge xgx_2 \wedge x_1fa \wedge x_2fa' \right).$$

For constructing this formula we introduce the notions of projection, rooted path and value of a rooted path in a solved formula.

The first step in constructing β_{ext} is to generate a projection for β', which is

$$xf \downarrow xg.$$

xf and xg are called rooted paths. The value $|xf|_\gamma$ and $|xg|_\gamma$ of xf and xg in γ are x_1 and x_2, respectively. Since $xf \downarrow xg$ is not entailed by β, we know that the values of xf and xg in γ must be different variables. Hence, we can add consistently the constraints x_1fa and x_2fa' to generate β_{ext}.

Definition 4.2 (Basic Constraint) *A* basic constraint *is either $\bot$ or a possibly empty conjunction of atomic formulae.*

In the case of FT′, we have t_1ft_2 and $t_1 \doteq t_2$ as atomic formulae. In CFT′, we have the additional atomic formulae of form tF. Note that $\top$ is a basic constraint since $\top$ is the empty conjunction. In the following, we will always use the Greek letter ϕ to denote basic constraints.

We say that a basic constraint ϕ **binds** x **to** y **(resp.** c**)** if $x \doteq y \in \phi$ (resp. $x \doteq c \in \phi$) and x occurs only once in ϕ. Here it is important to note that we consider equations as directed, that is, assume that $x \doteq y$ is different from $y \doteq x$ if $x \neq y$. We say that ϕ **eliminates** x if ϕ binds x to some variable y or some constant c.

Definition 4.3 (Solved Formula) *A* **solved formula** *is a basic constraint $\gamma \neq \bot$ such that the following conditions are satisfied:*

1. *no atomic constraint occurs twice in ϕ;*
2. *an equation $x \doteq t$ appears in γ if and only if γ eliminates x;*
3. *if $xft \in \phi$ and $xft' \in \phi$, then $t = t'$;*
4. *ϕ contains no atomic constraint of the form $c \doteq t$, cft or $atom(c)$;*
5. *if $xft \in \phi$, then $atom(x) \notin \phi$.*

(Cong) $\dfrac{x f t_1 \wedge x f t_2 \wedge \phi}{x f t_1 \wedge t_1 \doteq t_2 \wedge \phi}$

(Elim) $\dfrac{x \doteq t \wedge \phi}{x \doteq t \wedge \phi[x \leftarrow t]}$ $x \in \mathcal{V}(\phi)$ and $x \neq y$

(CCl) $\dfrac{c_1 \doteq c_2}{\bot}$ $c_1 \neq c_1$

(CFCl1) $\dfrac{c f t}{\bot}$

(CFCl2) $\dfrac{atom(x) \wedge x f t \wedge \phi}{\bot}$

(Orient) $\dfrac{c \doteq x}{x \doteq c}$

(Triv) $\dfrac{t \doteq t \wedge \phi}{\phi}$

(Simpl1) $\dfrac{P(x) \wedge P(x) \wedge \phi}{P(x) \wedge \phi}$

(Simpl2) $\dfrac{atom(c) \wedge \phi}{\phi}$

Figure 4.1: The basic simplification rules.

Every solved formula γ has a unique decomposition $\gamma = \gamma_N \wedge \gamma_G$ into a possibly empty conjunction γ_N of equations "$x \doteq y$" and a possibly empty conjunction γ_G of constraints "$P(x)$" and feature constraints "$x f y$". We call γ_N the normaliser *and γ_G the* graph *of γ.*

The letter γ always denotes a solved formula. We will see that every basic constraint is equivalent in $\mathrm{FT}_0^{\mathcal{P}}$ to either $\bot$ or a solved formula.

Note that the basic simplification rules (Cong), (CFCl1), (CCl), (Simpl2) and (CFCl2) correspond to the axioms schemes (Ax1), (Ax3), (Ax2), (Ax4) and (Ax5) , respectively. Thus, they are equivalence transformation with respect to $\mathrm{FT}_0^{\mathcal{P}}$. The remaining simplification rules are equivalence transformations in general.

Proposition 4.1 *The basic simplification rules are terminating and perform equivalence transformations with respect to* $\mathrm{FT}_0^{\mathcal{P}}$. *Moreover, a basic constraint* $\phi \neq \bot$ *is solved if and only if no basic simplification rule applies to it.*

Proof. To see that the basic simplification rules are terminating, observe that no rule adds a new variable and that every rule preserves eliminated variables. Since rule (Elim) increases the number of eliminated variables, and the remaining rules obviously terminate, the entire system must terminate. The other claims are easy to verify. □

Proposition 4.2 *Let* ϕ *be a formula built from atomic formulae with conjunction. Then one can compute a formula* δ *that is either solved or* $\bot$ *such that* $\phi \models\!\!\!\models_{\mathrm{FT}_0^{\mathcal{P}}} \delta$ *and* $\mathcal{V}(\delta) \subseteq \mathcal{V}(\phi)$.

Proof. Follows from the preceding proposition and the fact that the basic simplification rules do not introduce new variables. □

Definition 4.4 (Closure) *The* closure $[\gamma]$ of a solved formula γ *is the closure of the atomic formulae occurring in* γ *with respect to the following deduction rules:*

$$\frac{}{x\varepsilon x} \qquad \frac{x \doteq t}{x\varepsilon t} \qquad \frac{xpy \quad yft}{xpft} \qquad \frac{xpt \quad yqt}{xp \downarrow yq} \qquad \frac{P(t) \quad xpt}{P(xp)} \qquad \frac{xpc}{atom(xp)}$$

Recall that we assume that equations $x \doteq y$ *are directed, that is, are ordered pairs of variables. Hence,* $x\epsilon y \in [\gamma]$ *and* $y\epsilon x \notin [\gamma]$ *if* $x \doteq y \in \gamma$.

Proposition 4.3 *Let* γ *be a solved formula. Then:*

1. *if* $\pi \in [\gamma]$, *then* $\gamma \models\!\!\!\models_{\mathrm{FT}_0^{\mathcal{P}}} \pi$
2. $x\varepsilon t \in [\gamma]$ *iff* $x = t$ *or* $x \doteq t \in \gamma$
3. $xft \in [\gamma]$ *iff* $xft \in \gamma$ or $\exists z$: $x \doteq z \in \gamma$ and $zft \in \gamma$
4. $xpft \in [\gamma]$ *iff* $\exists z$: $xpz \in [\gamma]$ and $zft \in \gamma$
5. *if* $p \neq \varepsilon$ *and* $xpt, xpt' \in [\gamma]$, *then* $t = t'$
6. *or all* $P \in \mathcal{P}$, $P(xp) \in [\gamma]$ *iff* $xpt \in [\gamma]$ *and* $P(t) \in \gamma$
7. $atom(xp) \in [\gamma]$ *iff* $xpc \in [\gamma]$ *for some* $c \in \mathcal{L}$ *or* $xpy \in [\gamma]$ *and* $atom(y) \in \gamma$.

8. it is decidable whether a path constraint is in $[\gamma]$.

Proof. For the first claim one verifies the soundness of the deduction rules for path constraints. The verification of the other claims is straightforward. □

Now we extend the notion of path constraints to include also existential quantified formulae.

Definition 4.5 (Prime Formula) *Let* ϕ *be a basic constraint. A formula* $\beta = \exists X \phi$ *is called* prime *if it satisfies the following conditions:*

1. ϕ *is solved;*
2. X *has no variable in common with the normaliser of* ϕ*;*
3. *for every* $x \in X$ *there is a variable* $y \in \mathcal{V}(\beta)$ *and a path* p *such that* $ypx \in [\phi]$.

Given a formula $\beta = \exists X \phi$ where ϕ is a basic constraint, we can simply transform β into an equivalent formula β' such that the first two conditions are satisfied. We will later see that in the case of the complete theories FT′ and CFT′, we can find also an β' that satisfies additionally the third conditions. In the following, the letter β denotes always a prime formula if nothing else is stated.

Definition 4.6 (Closure of Prime Formulae) *The* closure of a prime formula $\beta = \exists X \gamma$ *is defined as follows:*

$$[\beta] \;:=\; \{\pi \in [\gamma] \mid \pi = x\varepsilon\downarrow \text{ or } \pi \text{ proper path constraint with } \mathcal{V}(\pi) \cap X = \emptyset\}.$$

Proposition 4.4 *If* β *is a prime formula and* $\pi \in [\beta]$*, then* $\beta \models \pi$ *(and hence* $\neg\pi \models \neg\beta$*).*

Proof. Let $\beta = \exists X \gamma$ be a prime formula, $\mathfrak{A}, \alpha \models \beta$, and $\pi \in [\beta]$. Let α' be an arbitrary X-update of α such that $\mathfrak{A}, \alpha' \models \gamma$. Since $[\beta] \subseteq [\gamma]$, we have $\pi \in [\gamma]$ and thus $\mathfrak{A}, \alpha' \models \pi$. If π has no variable in common with X, then $\mathfrak{A}, \alpha \models \pi$. Otherwise, π has the form "$x\varepsilon\downarrow$" and hence $\mathfrak{A}, \alpha \models \pi$ holds trivially. □

We now know that the closure $[\beta]$, taken as an infinite conjunction, is entailed by β. We are going to show that, conversely, β is entailed by certain finite subsets of its closure $[\beta]$. For this we need first the definition of a rooted path.

Definition 4.7 (Rooted Path) *A* rooted path *xp consists of a variable x and a path p. The* value *$|xp|_\gamma$ of a rooted path xp in some solved formula γ is defined as follows:*

$$|xp|_\gamma := \begin{cases} x & \text{iff } p = \varepsilon \wedge x \doteq t \notin \gamma \\ t & \text{iff } p = \varepsilon \wedge x \doteq t \in \gamma \\ t & \text{iff } xpt \in [\gamma] \\ \text{undefined} & \text{else.} \end{cases}$$

A rooted path xp is called realized *in a solved formula γ iff $|xp|_\gamma$ is defined. A rooted path xp is* realized *in a prime formula $\beta = \exists X\gamma$ if either $p = \varepsilon$ or $x \in \mathcal{V}(\beta)$ and xp is realized in γ.*

We say that a proper path constraint π **contains** a rooted path xp if $\pi = xp{\downarrow}$, $\pi = xpc$, $\pi = xp \downarrow yq$ or $\pi = yq \downarrow xp$

Proposition 4.5 *$|\cdot|_\gamma$ is a partial function for every solved formula γ.*

Proof. Follows from proposition 4.3 (5). □

Proposition 4.6 *Let xp be a rooted path. If xp is realized in some solved formula γ, then $|xp|_\gamma$ is either a constant or a variable z with $z \in \mathcal{V}(\gamma_G)$.*

Proposition 4.7 *Let $\beta = \exists X\gamma$ be a prime formula and $\pi = xp \downarrow yq$ be a proper path constraint with $x, y \notin X$. If both xp and yq are realized in β, then*

$$\exists X\gamma \wedge \pi \models\!|_{\mathrm{FT}_0^{\mathcal{P}}} \exists X(\gamma \wedge |xp|_\gamma \doteq |yq|_\gamma).$$

Definition 4.8 (Access Function) *An* access function *for a prime formula $\beta = \exists X\gamma$ is a function that maps every $x \in \mathcal{V}(\gamma) - X$ to the rooted path $x\varepsilon$, and every $x \in X$ to a rooted path $x'p$ such that $x'px \in [\gamma]$ and $x' \notin X$.*

Proposition 4.8 *For every prime formula $\beta = \exists X\gamma$ and every access function @ of β,*

$$|@x|_\gamma = x.$$

Thus, $|\cdot|_\gamma$ is the left inverse of @. But the converse is not true. Given the prime formula $\beta = \exists z\gamma$ with

$$\gamma = xfz \wedge ygz$$

and the access function with $@z = xf$, we have $@|yg|_\gamma = xf$.

Note that every prime formula has at least one access function, and that the access function of a prime formula $\exists X\gamma$ is injective on $\mathcal{V}(\gamma)$ (follows from Proposition 4.3 (5)).

Definition 4.9 (Projection) *The* projection *of a prime formula $\beta = \exists X\gamma$ with respect to an access function @ for β is the conjunction of the following proper path constraints:*

$$\begin{aligned}
&\{x\varepsilon \downarrow y\varepsilon \mid x \doteq y \in \gamma\} \cup \\
&\{P(x'p) \mid P(x) \in \gamma,\ x'p = @x\} \cup \\
&\{x'pf \downarrow y'q \mid xfy \in \gamma,\ x'p = @x,\ y'q = @y\}.
\end{aligned}$$

Obviously, one can compute for every prime formula an access function and hence a projection. Furthermore, if λ is a projection of a prime formula β, then λ taken as a set is a finite subset of the closure $[\beta]$.

Proposition 4.9 *Let λ be a projection of a prime formula β. Then $\lambda \subseteq [\beta]$ and $\lambda \models\!\!\models_{\mathrm{FT}_0^{\mathcal{P}}} \beta$.*

Proof. Let λ be the projection of a prime formula $\beta = \exists X\gamma$ with respect to an access function @.

Since every path constraint $\pi \in \lambda$ is in $[\beta]$ and thus satisfies $\beta \models \pi$, we have $\beta \models \lambda$.

To show the other direction, suppose $\mathfrak{A}, \alpha \models \lambda$, where $\mathfrak{A}$ is a model of $\mathrm{FT}_0^{\mathcal{P}}$. Then $\mathfrak{A}, \alpha' \models x'px$ for every $x \in X$ with $@x = x'p$ defines a unique X-update α' of α. From the definition of a projection it is clear that $\mathfrak{A}, \alpha' \models \gamma$. Hence $\mathfrak{A}, \alpha \models \beta$. □

As a consequence of this proposition one can compute for every prime formula an equivalent quantifier-free conjunction of proper path constraints.

Proposition 4.10 *If β is a prime formula, then $\beta \models\!\!\models_{\mathrm{FT}'} [\beta]$*

Proof. By Proposition 4.4 we have $\beta \models_{\mathrm{FT}'} [\beta]$, and by Proposition 4.9 we have $[\beta] \models_{\mathrm{FT}'} \beta$ since β has a projection $\lambda \subseteq [\beta]$. □

4.4 The Theory FT′

4.4.1 The Axioms

The first five axiom schemes of FT′ are the axiom schemes of $\mathrm{FT}_0^{\{\}}$:

(Ax1)	$\tilde{\forall}(xfy \wedge xfz \rightarrow y \doteq z)$	for every feature f.
(Ax2)	$\neg(c_1 \doteq c_2)$	if c_1 and c_2 are different constants
(Ax3)	$\tilde{\forall}(cfx \rightarrow \bot)$	for all constants c.
(Ax4)	$atom(c)$	for all constants c
(Ax5)	$\forall x, y(xfy \wedge atom(x) \rightarrow \bot)$.	

The sixth and final axiom scheme will say that certain "consistent feature descriptions" are satisfiable. For its formulation we need the important notion of a solved clause.

An **exclusion constraint** is an additional atomic formula of the form $xf{\uparrow}$ ("f undefined on x") taken to be equivalent to $\neg\exists y\,(xfy)$ (for some variable $y \neq x$).

Definition 4.10 (Solved Clause) *A* solved clause *is a possibly empty conjunction ϕ of atomic formulae of the form xft and $xf{\uparrow}$ such that the following conditions are satisfied:*

1. *no atomic formula occurs twice in ϕ*
2. *there are no constraints of the form cft, $cf{\uparrow}$ or $atom(c)$ in ϕ*
3. *if $xft \in \phi$, then there exists no $t \neq t'$ such that $xft' \in \phi$*
4. *if $xft \in \phi$, then $xf{\uparrow} \notin \phi$*
5. *if $atom(x) \in \phi$, then neither xfy nor $xf{\uparrow}$ is in ϕ.*

Proposition 4.11 *Given a solved clause ϕ, the subset ϕ' of ϕ containing all atomic constraints of the form t_1ft_2 is a graph. Vice versa, the graph of a solved formulae is a solved clause.*

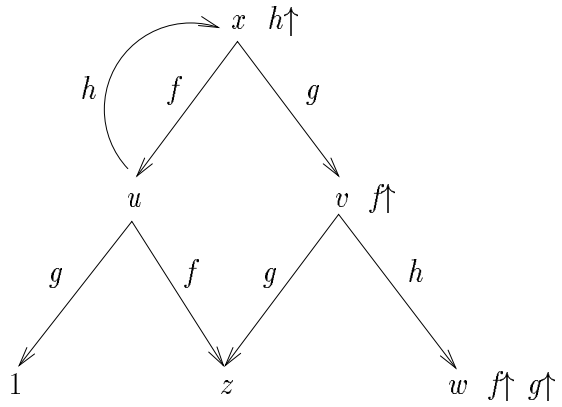


Figure 4.2: A graph representation of a solved clause.

Figure 4.2 gives a graph representation of the solved clause

$$\begin{aligned}
&xfu \wedge xgv \wedge xh\uparrow \wedge \\
&uhx \wedge ug1 \wedge ufz \wedge \\
&vgz \wedge vhw \wedge vf\uparrow \wedge \\
&wf\uparrow \wedge \ wg\uparrow.
\end{aligned}$$

Here, the symbols 1 denotes a constant. A more readable textual representation of this solved clause is

$$\begin{aligned}
x &: [f\!:\!u \ \ g\!:\!v \ \ h\uparrow] \\
u &: [h\!:\!x \ \ g\!:\!1 \ \ f\!:\!z] \\
v &: [g\!:\!z \ \ h\!:\!w \ \ f\uparrow] \\
w &: [f\uparrow \ \ g\uparrow].
\end{aligned}$$

As in the example, a solved clause can always be seen as a graph whose nodes are the variables and constants appearing in the clause and whose arcs are given by the feature constraints xft. The constraints $xf\uparrow$ appear as labels of the node x. The graphical representation of solved clauses should be very helpful in understanding the proofs to come.

A variable x is **constrained** in a solved clause ϕ if ϕ contains a constraint of the form xft or $xf\uparrow$. We use $\mathcal{CV}(\phi)$ to denote the set of all variables that are constrained in ϕ. The variables in $\mathcal{V}(\phi) - \mathcal{CV}(\phi)$ are called the **parameters** of a solved clause ϕ. In the graph representation of a solved clause the parameters appear as leaves that are not not labelled with a feature exclusion. The parameter of the solved clause in Figure 4.2 is y.

We can now state the final axiom scheme. It says that the constrained variables of a solved clause have solutions for all values of the parameters:

(Ax6) $\tilde{\forall}\exists X\phi$ for $X = \mathcal{CV}(\phi)$ and every solved clause ϕ that contains no constraints of the form $atom(x)$.

Proposition 4.12 *Let ϕ be a solved clause and $X = \mathcal{CV}(\phi)$. Then*

$$\mathrm{FT}' \models \tilde{\forall}\exists X\phi.$$

The theory FT′ is the set of all sentences that can be obtained as instances of the axiom schemes (Ax1), (Ax2), (Ax3), (Ax4), (Ax5) and (Ax6). In the next sections we will show that FT′ is a complete theory. By using an adaption of the proof of Theorem 8.3 in [Smo92] one can show that FT_0 is undecidable.

4.4.2 Feature Trees and Feature Graphs

In this section we establish three models of FT′ consisting of either feature trees or feature graphs. Since we will show that FT′ is a complete theory, all three models are in fact elementarily equivalent.

Theorem 4.1 *The feature tree structures $\mathfrak{T}_{\mathrm{FT}'}$ and $\mathfrak{R}_{\mathrm{FT}'}$ are models of the theory* FT′.

Proof. We will first show that $\mathfrak{T}_{\mathrm{FT}'}$ is a model of FT′.

The first five axiom schemes are obviously satisfied by $\mathfrak{T}_{\mathrm{FT}'}$. To see that $\mathfrak{T}_{\mathrm{FT}'}$ satisfies the sixth axiom scheme, let δ be a solved clause, X be the variables constrained in δ, and α be a valuation into $\mathfrak{T}_{\mathrm{FT}'}$. It suffices to show that there exists an X-update α' of α such that $\mathfrak{T}_{\mathrm{FT}'}, \alpha' \models \delta$.

For a feature tree $\sigma = (D, \lambda)$ and a path p we define $p\sigma$ to be the feature tree (D', σ') with $D' = \{pq \mid q \in D\}$ and $\lambda' = \{(pq, a) \mid (q, a) \in \lambda\}$. Clearly, $p^{-1}p\sigma = \sigma$, but the converse may not hold. Now one can verify that

$$\forall x \in X \quad \alpha'(x) \;:=\; (D_x, \lambda_x) \;\cup \bigcup_{xpy \in [\delta],\ y \notin X} p\alpha(y)$$

with

$$\begin{aligned} D_x &:= \{p \mid xp{\downarrow} \in [\delta]\} \\ \lambda_x &:= \{(p, c) \mid xpc \in [\delta]\} \end{aligned}$$

defines an X-update α' of α such that $\mathfrak{T}_{\mathrm{FT}'}, \alpha' \models \delta$.

The same construction shows that $\mathfrak{R}_{\mathrm{FT}'}$ is a model of FT′. □

Now we will define the third FT′-model, namely the feature graph model. A **feature pregraph** is a pair (t, γ) consisting of a term t (called the **root**) and a solved clause γ not containing exclusion constraints and constraints of the form $atom(x)$ such that the following conditions are satisfied:

- if t is a constant c, then γ is the empty clause, and
- if t is a variable x, then for every variable y occurring in γ, there exists a path p satisfying $xpy \in [\gamma]$.

If one deletes the exclusion constraints in Figure 4.2, one obtains the graphical representation of a feature pregraph whose root is x.

A feature pregraph (t, γ) is called a **subpregraph** of a feature pregraph (t', δ) if $\gamma \subseteq \delta$ and $t = t'$ or t' is a variable x and $xpt \in [\delta]$ for some path p. Note that a feature pregraph has only finitely many subpregraphs.

We say that two feature pregraphs are **equivalent** if they are equal up to consistent variable renaming. For instance, $(x,\ xfy \wedge ygx)$ and $(u,\ ufx \wedge xgu)$ are equivalent feature pregraphs.

A **feature graph** is an element of the quotient of the set of all feature pregraphs with respect to equivalence as defined above. Put differently, a feature graph is an isomorphism class of feature pregraphs. We use $\overline{(t, \gamma)}$ to denote the feature graph obtained as the equivalence class of the feature pregraph (t, γ).

The **feature graph structure** $\mathfrak{G}$ is the FT′-structure defined as follows:

- the universe of $\mathfrak{G}$ is the set of all feature graphs
- $c^{\mathfrak{G}} = (c, \{\})$ for every $c \in \mathcal{L}$.
- $atom^{\mathfrak{G}} = \{c^{\mathfrak{G}} \mid c \in \mathcal{L}\}$
- $(\overline{(x, \gamma)}, \sigma) \in f^{\mathfrak{G}}$ iff there exists a maximal feature subpregraph (t, δ) of (x, γ) such that $xft \in \gamma$ and $\sigma = \overline{(t, \delta)}$.

Theorem 4.2 *The feature graph structure* $\mathfrak{G}$ *is a model of the theory* FT′.

Proof. The first five axiom schemes are obviously satisfied by $\mathfrak{G}$. To see that $\mathfrak{G}$ satisfies the sixth axiom scheme, let δ be a solved clause and α a valuation into $\mathfrak{T}_{\mathrm{FT}'}$. It suffices to show that there exists an $\mathcal{CV}(\delta)$-update α' of α such that $\mathfrak{G}, \alpha' \models \delta$.

First we choose for the parameters $y \in \mathcal{V}(\delta) - \mathcal{CV}(\delta)$ variable disjoint feature pregraphs (y, γ_y) such that $\alpha(y) = \overline{(y, \gamma_y)}$. Moreover, we can assume without loss of generality that every pregraph (y, γ_y) has with δ exactly its root variable y in common. Hence

$$\delta' \;:=\; \delta \wedge \bigwedge_{y \in \mathcal{V}(\delta) - \mathcal{CV}(\delta)} \gamma_y$$

is a solved clause. Now, for every constrained variable $x \in \mathcal{CV}(\delta)$, let ρ_x be the maximal solved clause such that $\rho_x \subseteq \delta'$ and (x, ρ_x) is a feature pregraph. Then the $\mathcal{CV}(\delta)$-update α' of α such that $\alpha'(x) = \overline{(x, \rho_x)}$ for every $x \in \mathcal{CV}(\delta)$ satisfies $\mathfrak{G}, \alpha' \models \delta$. □

Let $\mathfrak{F}$ be the structure whose domain consists of all feature pregraphs and that is otherwise defined analogous to $\mathfrak{G}$. Note that $\mathfrak{G}$ is in fact the quotient of $\mathfrak{F}$ with respect to equivalence of feature pregraphs.

Proposition 4.13 *The feature pregraph structure $\mathfrak{F}$ is a model of FT_0 but not of FT'.*

Proof. It is easy to see that $\mathfrak{F}$ satisfies the first five axiom schemes. To see that $\mathfrak{F}$ does not satisfy the sixth axiom scheme, consider the solved clause

$$\delta \;=\; xfy \wedge xgz$$

and a valuation α into $\mathfrak{F}$ such that $\alpha(y) = (x, xha)$, $\alpha(z) = (x, xhb)$, where a and b are two different constants. Then there exists no x-update α' of α satisfying $\mathfrak{F}, \alpha' \models \delta$ since a feature pregraph cannot contain both xha and xhb. □

4.4.3 Some Properties of Prime Formulae

We will now show that the class of prime formulae defined in section 4.2 satisfies the requirements 1– 4 of Lemma 4.1. Note that all propositions proven for $\mathrm{FT}_0^{\{\}}$ in Section 4.3 can also be used for the theory FT′ as $\mathrm{FT}_0^{\{\}}$ is a sub-theory of FT′.

Requirement 2 is trivial. For requirement 1 note that if ϕ is a basic formula tft' or $t \doteq t'$ with $t \neq t'$, then ϕ is equivalent to $\bot$, which is $\neq \top$. If ϕ is a trivial equation

$t \doteq t$, then ϕ is equivalent to $\top$. All other atomic formulae are prime formulae by definition. In this section we are going to prove requirements 3 and 4.

For this we define the notion of decided variables of an existential quantified solved formula $\exists X\gamma$. The decided variables are those variables in $\mathcal{V}(\gamma)$ whose valuation is uniquely determined by the valuation of the free variables.

Definition 4.11 (Decided Variables) *Let γ be a solved formula and X be a set of variables. A variable $x \in \mathcal{V}(\gamma)$ is said to be* decided in *a formula $\delta = \exists X\gamma$ if $x \in \mathcal{V}(\delta)$ or there is a variable $y \in \mathcal{V}(\delta)$ with $y \neq x$ and a path p such that*

$$ypx \in [\gamma].$$

For a solved clause ϕ we say that x is decided *in $\delta = \exists X\phi$ if x is decided in $\exists X\gamma$, where $\gamma \subseteq \phi$ is the solved formula containing all constraints of ϕ that are of the form xfy.*

We say that a variable is undecided if it is not decided. The set of decided variables of a formula δ will be denoted by $\mathcal{D}ec(\delta)$. Consider the formula

$$\delta \;:=\; \exists x, x_1, x_2(xfx_1 \wedge xgx_2 \wedge xhy \wedge zfx_2) \tag{4.4}$$

Then the variables $\mathcal{D}ec(\delta) = \{y, z, x_2\}$. x and x_1 are both undecided in δ.

In the following we show that every existential quantified solved formula $\exists X\gamma$ is equivalent to $\exists X\gamma'$, where γ' is the set of constraints on the decided variables. Thus, we above defined formula δ is equivalent in FT′ to the formula

$$\delta' \;:=\; \exists x_2(zfx_2).$$

We will use this equivalence to show that we can transform every existential quantified solved formula in a prime formula thus proving requirement 4 of Lemma 4.1. Note that in a prime formula $\beta = \exists X\gamma$ every variable in $\mathcal{V}(\gamma)$ is decided. Furthermore, this will be used in the next section for the proofs of assumptions 5 and 6 of Lemma 4.1.

Proposition 4.14 *Let Y be a set of variables that are existential quantified and decided in $\delta = \exists X\gamma$ and let α be some valuation into a* FT′ *model $\mathfrak{A}$ with $\mathfrak{A}, \alpha \models \delta$. Then there exists a unique Y-update α' of α such that*

$$\mathfrak{A}, \alpha' \models \exists X \backslash Y\gamma.$$

A constraint π is called a **constraint for** x if π is of the form xft, $atom(x)$ or $x \doteq t$.

Lemma 4.2 (Garbage Collection) *Let ϕ, ϕ' be solved clauses and let $\delta = \exists X \phi$ and $\delta' = \exists X' \phi'$ be formulae with $\mathcal{V}(\delta) = \mathcal{V}(\delta')$ and $\mathcal{D}ec(\delta) = \mathcal{D}ec(\delta')$. If δ and δ' contain exactly the same constraints for the decided variables, then*

$$\delta \models\!\!\!|_{\mathrm{FT}'} \delta'.$$

Proof. Let δ, δ' be given as described and let $Z = \mathcal{D}ec(\delta) \cap X = \mathcal{D}ec(\delta') \cap X'$. As δ and δ' contain the same constraints for the decided variables, we can write δ and δ' as

$$\delta = \exists Z \exists Y (\gamma \wedge \psi) \quad \text{and} \quad \delta' = \exists Z \exists Y' (\gamma' \wedge \psi),$$

where $Y = X \backslash Z$, $Y' = X' \backslash Z$ and ψ contains all constraints for the variables in Z. Note that all variables of ψ are decided in δ and δ'. Hence $\mathcal{V}(\psi) \cap Y = \emptyset$ and $\mathcal{V}(\psi) \cap Y' = \emptyset$. This implies that

$$\delta \models\!\!\!| \exists Z (\exists Y \gamma \wedge \psi) \quad \text{and} \quad \delta' \models\!\!\!| \exists Z (\exists Y' \gamma' \wedge \psi),$$

Now γ and γ' are solved clauses. The variables which are free in $\exists Y \gamma$ and $\exists Y' \gamma'$ are decided in δ and δ'. Since we have put all constraints for the decided variables into ψ, we know that γ and γ' contain no constraints for the free variables in $\exists Y \gamma$ and $\exists Y' \gamma'$. This implies that the free variables of $\exists Y \gamma$ (resp. $\exists Y' \gamma'$) are parameters of γ (resp. γ'). Hence

$$\mathrm{FT}' \models \tilde{\forall} \exists Y \gamma \quad \text{and} \quad \mathrm{FT}' \models \tilde{\forall} \exists Y' \gamma'$$

by Proposition (4.12). This shows $\delta \models\!\!\!| \exists Z \psi$ and $\delta' \models\!\!\!| \exists Z \psi$. □

We will say that two formula $\delta = \exists X \gamma$ and $\delta' = \exists X' \gamma'$ **differ only on the undecided variables** if $\mathcal{V}(\delta) = \mathcal{V}(\delta')$, $\mathcal{D}ec(\delta) = \mathcal{D}ec(\delta')$, and δ and δ' contain exactly the same constraints for the decided variables.

Proposition 4.15 *For every prime formula β and every set of variables X one can compute a prime formula β' such that*

$$\exists X \beta \models\!\!\!|_{\mathrm{FT}'} \beta' \quad \text{and} \quad \mathcal{V}(\beta') \subseteq \mathcal{V}(\exists X \beta).$$

Proof. We will proof that we can compute a formula β' as required by the lemma for the special case $X = \{x\}$. For arbitrary sets X we can compute a β' by iterative application of the method for this special case.

Let $\beta = \exists Y \gamma$ be a prime formula and x be a variable. We construct a prime formula β' such that $\exists x \beta \models\!\!\!|_{\mathrm{FT}'} \beta'$ and $\mathcal{V}(\beta') \subseteq \mathcal{V}(\exists x \beta)$. We distinguish the following cases.

1. $x \notin \mathcal{V}(\beta)$. Then $\beta' := \beta$ does the job.

2. $\gamma = (x \doteq y \wedge \gamma')$. Then $\beta' := \exists Y \gamma'$ does the job.

3. $\gamma = (y \doteq x \wedge \gamma')$. Then $\beta' := \exists Y(\gamma'[x \leftarrow y])$ does the job since $\gamma \models\!\!\!\models x \doteq y \wedge \gamma'[x \leftarrow y]$.

4. $x \notin Y$ *and* x *occurs in the graph but not in the normaliser of* γ. Then $\exists x \exists Y \gamma \models\!\!\!\models \gamma_N \wedge \exists x \exists Y \gamma_G$. Let γ'_G contain all the constraints for the variable that are decided in $\exists x \exists Y \gamma_G$. Then $\exists x \exists Y \gamma_G$ and $\exists x \exists Y \gamma'_G$ have the same set of decided variables and contain the same constraints for the decided variables. Since γ_G and γ'_G contain no equations, they are solved clauses. Hence, by proposition 4.2

$$\exists x \exists Y \gamma_G \models\!\!\!\models_{\mathrm{FT}'} \exists x \exists Y \gamma'_G.$$

This implies that $\beta' = \gamma_N \wedge \exists x \exists Y \gamma'_G$ is a prime formulae with $\beta \models\!\!\!\models_{\mathrm{FT}'} \beta'$ and $\mathcal{V}(\beta') \subseteq \mathcal{V}(\beta)$. □

Proposition 4.16 *For every two prime formulae* β *and* β' *one can compute a formula* δ *that is either prime or* $\bot$ *and satisfies*

$$\beta \wedge \beta' \models\!\!\!\models_{\mathrm{FT}'} \delta \quad \text{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\beta \wedge \beta').$$

Proof. Let $\beta = \exists X \gamma$ and $\beta' = \exists X' \gamma'$ be prime formulae. Without loss of generality we can assume that X and X' are disjoint. Hence

$$\beta \wedge \beta' \quad \models\!\!\!\models \quad \exists X \exists X'(\gamma \wedge \gamma').$$

Since $\gamma \wedge \gamma'$ is a basic constraint, Proposition 4.2 tells us that we can compute a formula ϕ that is either solved or $\bot$ and satisfies $\gamma \wedge \gamma' \models\!\!\!\models_{\mathrm{FT}'} \phi$ and $\mathcal{V}(\phi) \subseteq \mathcal{V}(\gamma \wedge \gamma')$. If $\phi = \bot$, then $\delta := \bot$ does the job. Otherwise, ϕ is solved. Since

$$\beta \wedge \beta' \quad \models\!\!\!\models_{\mathrm{FT}'} \quad \exists X \exists X' \phi,$$

we know by Proposition 4.15 how to compute a prime formula β'' such that $\beta \wedge \beta' \models\!\!\!\models_{\mathrm{FT}'} \beta''$. From the construction of β'' one verifies easily that $\mathcal{V}(\beta'') \subseteq \mathcal{V}(\beta \wedge \beta')$. □

4.4.4 Proof of the Main Lemmas

In this section we show that our prime formulae satisfy the requirements (5) and (6) of Lemma 4.1 and thus obtain the completeness of FT′. We start with the definition of the central notion of a joker.

Definition 4.12 *Let $\beta = \exists Y \gamma$ be a prime formula and X be a set of variables. A rooted path xp is called* decided in β wrt. X *if either $x \notin X$ or there is some prefix p' such that xp' is realized and $|xp'|_\gamma$ is constant or a variable decided in $\exists X \beta$.*

xp is called undecided in β wrt. X is xp is not decided in β wrt. X. Note that in this case x must be an element of X. If β and X are clear from the context, we will just say decided instead of decided in β wrt. X.

Proposition 4.17 *If π is a proper path constrained such that all rooted paths contained in π are decided in β wrt. X, then either $\mathfrak{A}, \alpha \models \forall X(\beta \to \pi)$ or $\mathfrak{A}, \alpha \models \forall X(\beta \to \neg\pi)$.*

Definition 4.13 (X-Joker) *A proper path constraint π is called an* **X-joker** *for a prime formula β if $\pi \notin [\beta]$ and one of the following conditions is satisfied:*

1. *$\pi = xpc$ and xp is undecided in β wrt. X*
2. *$\pi = atom(xp)$ and xp is undecided in β wrt. X*
3. *$\pi = xp \downarrow yq$ and xp is undecided in β wrt. X*
4. *$\pi = yp \downarrow xq$ and xq is undecided in β wrt. X.*

We will now give some examples for X-jokers. The path constraint $xf \downarrow xg$ is an $\{x\}$-joker for the formulae

$$\exists x_1, x_2(xfx_1 \wedge xgx_2)$$
$$\exists x_1(xfx_1 \wedge xgy)$$
$$\exists x_1(xfx_1).$$

On the other hand, $xf \downarrow xg$ is no $\{x\}$-joker for the formulae

$$\begin{aligned} \beta_1 &:= \exists x_1(xfx_1 \wedge xgx_1) \\ \beta_2 &:= \exists x_1(xfy \wedge xgx_1 \wedge zhx_1), \end{aligned}$$

since $xf \downarrow xg \in [\beta_1]$, and both xf and xg are decided in β_2 wrt. $\{x\}$.

Proposition 4.18 *It is decidable whether a rooted path is undecided in a prime formula wrt. a set of variables, and whether a path constraint is an X-joker for a prime formula.*

Proof. Follows with Proposition 4.3. □

Lemma 4.3 *Let β be a prime formula and $\pi_1, \ldots, \pi_n$ be X-jokers for β. Then*

$$\exists X\beta \quad \models_{\mathrm{FT}'} \quad \exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\pi_i).$$

Proof. Let $\beta = \exists Y\gamma$ be a prime formula, $\pi_1, \ldots, \pi_n$ ($n > 0$) be X-jokers for β, $\mathfrak{A}$ be some model of FT′, and α be some valuation into $\mathfrak{A}$ with $\mathfrak{A}, \alpha \models \exists X\beta$. We have to show that $\mathfrak{A}, \alpha \models \exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\pi)$. We will define a prime formula β' satisfying the following:

- $\beta' \models \beta$,
- $\exists X\beta \models\!|_{\mathrm{FT}'} \exists X\beta'$,
- $\mathfrak{A}, \alpha \models \forall X(\beta' \to \neg\pi_i)$ for all $i = 1..n$

Once we have defined a β' satisfying these conditions, we can prove the claim using the following argumentation. Since $\exists X\beta \models\!|_{\mathrm{FT}'} \exists X\beta'$ and $\mathfrak{A}, \alpha \models \exists X\beta$, there must be an X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta'$. But as $\beta' \models \beta$ and for all $i = 1..n$ $\mathfrak{A}, \alpha \models \forall X(\beta' \to \neg\pi_i)$, we know that $\mathfrak{A}, \alpha' \models \beta \wedge \bigwedge_{i=1}^{n} \neg\pi_i$. This shows $\mathfrak{A}, \alpha \models \exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\pi_i)$.

For the construction of β' we define F_{used} to be the set of all features that occur in the path constraints π_i. In the following, we will just say that a rooted path xp is decided when meaning that xp is decided in β wrt. X, and we will use undecided in a similar way. Let $Z \subseteq \mathcal{V}(\gamma_G)$ be the set of all variables of the graph of γ that are undecided, and let h be a new feature. Note that if a rooted path xp is undecided and realized in β, then $|xp|_\gamma$ is a variable z with $z \in Z$. Furthermore, let $Z_C \subseteq Z$ be the set of variables $z \in Z$ with $atom(z) \in \gamma$.

By Proposition 4.14 there exists a unique update α' of α to the variables that are decided in $\exists X\beta$. For each $z \in Z_C$ we fix a constant c_z that does not appear in γ and π_i for $i = 1 \ldots n$ such that

$$\forall x \in ((X \cup Y) \cap \mathcal{D}ec(\exists X\beta)) : \; c_z^{\mathfrak{A}} \neq \alpha'(x).$$

By the construction we know that for every rooted path yq which is contained in some π_i and which is both realized and decided the proposition

$$\mathfrak{A}, \alpha \models \forall X(\beta \to \neg ypc_z)$$

holds. Similarly, we fix for every $z \in Z \backslash Z_C$ a constant c_z that does not appear in γ and π_i for $i = 1 \ldots n$ such that

$$\forall x \in ((X \cup Y) \cap \mathcal{D}ec(\exists X \beta)) : \ \neg \alpha'(x)\, h^{\mathfrak{A}}\, c_z^{\mathfrak{A}}.$$

Again we know for every rooted path yq which is contained in some π_i and which is both realized and decided that

$$\mathfrak{A}, \alpha \models \forall X(\beta \rightarrow \neg yphc_z)$$

is true. It is understood that $c_z \neq c_{z'}$ for $z \neq z'$ and $z, z' \in Z$.

Let ψ be obtained from γ_G be deleting all constraints $atom(z)$ with $z \in Z_C$, and let $z_1, \ldots, z_m$ be an enumeration of Z_C. The formula $\beta' = \exists Y(\gamma'_N \wedge \gamma'_G)$ is now defined by

$$\begin{aligned} \gamma'_N &= \gamma_N \wedge z_1 \doteq c_{z_1} \wedge \ldots \wedge z_m \doteq c_{z_m} \\ \gamma'_G &= \psi[z_1 \leftarrow c_{z_1}, \ldots, z_m \leftarrow c_{z_m}] \wedge \bigwedge_{z \in (Z \backslash Z_C)} \left(zhc_z \wedge \bigwedge_{f \in F_{\text{used}},\ zft \notin \gamma} zf\uparrow \right). \end{aligned}$$

We will show that β' satisfies the requirements stated above. Clearly, $\beta' \models_{\text{CFT}'} \beta$. Next we will show that $\exists X \beta \models\!\!\models_{\text{FT}'} \exists X \beta'$.

By the definition of Z we know that γ_N contains no constraint of the form $z \doteq t$ with $z \in Z$. Since the variables in Z are undecided, we know that $z' \doteq z \in \gamma_N$ and $z \in Z$ implies $z' \in X$. Since γ_N eliminates the variables on the left sides of the equation, this implies

$$\exists X \exists Y(\gamma_N \wedge \gamma_G) \models\!\!\models \exists X \exists Y(\gamma''_N \wedge \gamma_G),$$

where γ''_N is the biggest subset of γ_N with $\mathcal{V}(\gamma''_N) \cap Z = \emptyset$. Similarly, $\exists X \beta' = \exists X \exists Y(\gamma'_N \wedge \gamma'_G) \models\!\!\models \exists X \exists Y(\gamma''_N \wedge \gamma'_G)$. Now $\mathcal{V}(\gamma''_N) \cap Z = \emptyset$ implies

$$\exists X \beta \models\!\!\models \exists Z'(\gamma''_N \wedge \exists Z \gamma_G) \quad \text{and} \quad \exists X \beta' \models\!\!\models \exists Z'(\gamma''_N \wedge \exists Z \gamma'_G),$$

where $Z' = (X \cup Y) \backslash Z$.

Since $\mathcal{V}(\exists Z \gamma_G) \subseteq \mathcal{D}ec(\exists X \beta)$, we know that every variable in Z is also undecided in $\exists Z \gamma_G$: if there would be a variable $z \in Z$ such that there is a p with $ypz \in [\gamma_G]$ and $y \in \mathcal{V}(\exists Z \gamma_G)$, then $y \in \mathcal{D}ec(\exists X \beta)$, which implies $z \in \mathcal{D}ec(\exists X \beta)$. But this is contradictory to the definition of Z. Hence, $\exists Z \gamma_G$ and $\exists Z \gamma'_G$ are solved clauses that differ only on the undecided variables. Then proposition 4.2 shows that $\exists Z \gamma_G \models\!\!\models_{\text{FT}'} \exists Z \gamma'_G$ and therefore $\exists X \beta \models\!\!\models_{\text{FT}'} \exists X \beta'$.

The remaining part is to show that $\mathfrak{A}, \alpha \models \forall X(\beta' \rightarrow \bigwedge_{i=1}^{n} \neg \pi_i)$ for all $i = 1..n$. We distinguish the following cases for π_i:

1. π_i *contains a rooted path* xp *that is undecided and not realized in* β: Let p' be the longest path such that xp' is realized in β (such a path must exists since at least $x\epsilon$ is realized in β), and let $p = p'fq$. As xp is undecided, we know that $|xp'|_\gamma$ is a variable z with $z \in Z$. As p' is the longest subpath of p' with xp' realized in β we know that $zft \notin \gamma$. If $z \in Z_C$, then we have substituted z by a constant c_z. Otherwise we have added $zf\uparrow$ in γ'_G since $f \in F_{\text{used}}$.

2. *every undecided rooted path contained in* π_i *is realized in* β. Note that in this case π_i cannot be of the form $xp\downarrow$, since xp realized in β implies that $xp\downarrow \in [\beta]$. We will split up this case as follows:

   (a) $\pi_i = xpc$. Then xp is undecided. Since xp is realized in β, we know that $|xp|_\gamma$ is a variable z with $z \in Z$. If $z \in Z_c$, then we have substituted z by a constant symbol c' different from c. Otherwise, we have added at least one feature constraint zhc_z in γ'_G, which implies $\beta' \models \neg\pi_i$.

   (b) $\pi_i = atom(xp)$. Then xp is undecided. Since xp is realized in β, we know that $|xp|_\gamma$ is a variable z with $z \in Z$. Now $\pi_i \notin [\beta]$ implies that $atom(z) \notin \gamma_G$ and therefore $z \notin Z_C$. Hence, we have added at least one feature constraint zhc_z in γ'_G.

   (c) $\pi_i = xp \downarrow yq$ *or* $\pi_i = yq \downarrow xp$ *where* xp *is undecided but realized in* β. Again we get $|xp|_\gamma = z \in Z$. There are two cases, namely $z \in Z_C$ and $z \in (Z \backslash Z_C)$.

   If $z \in Z_C$, then we have substituted z by c_z in γ'_G. If yq is undecided, we can assume that yq is also realized in β (otherwise case 1 would be applicable). This implies that we have either added a constraint $z'hc_{z'}$ in γ'_G or we have substituted z' by $c_{z'}$ in γ'_G, where $z' = |yq|_\gamma$. Since c_z and $c_{z'}$ are different, this shows $\beta' \models \neg\pi_i$.

   If yq is decided, then $\mathfrak{A}, \alpha \models \forall X(\beta \rightarrow \neg yqc_z)$ by the definition of c_z. As $\beta' \models \beta$, we get $\mathfrak{A}, \alpha \models \forall X(\beta' \rightarrow \neg xp \downarrow yq)$.

   The other case $z \in (Z \backslash Z_C)$ is handled analogously.

□

Note that the proof uses the axiom scheme (Ax6), the existence of infinitely many features, and the existence of infinitely many constants.

Lemma 4.4 *Let* β, β' *be prime formulae and* α *be a valuation into a model* $\mathfrak{A}$ *of FT such that*

$$\mathfrak{A}, \alpha \models \exists X(\beta \wedge \beta') \quad \textit{and} \quad \mathfrak{A}, \alpha \models \exists X(\beta \wedge \neg\beta').$$

Then every projection of β' contains an X-joker for β.

Proof. Without loss of generality we can assume that $\mathfrak{A}, \alpha \models \beta \wedge \beta'$. Furthermore, there exists an X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta \wedge \neg\beta'$. Let λ be a projection of β'. Since $\mathfrak{A}, \alpha' \not\models \beta'$, we know by Proposition 4.9 that $\mathfrak{A}, \alpha' \not\models \lambda$. Hence, there exists a proper path constraint $\pi \in \lambda$ such that $\mathfrak{A}, \alpha' \not\models \pi$. Since $\mathfrak{A}, \alpha \models \beta'$, we know by Proposition 4.4 that $\mathfrak{A}, \alpha \models \pi$. Hence, we know by Proposition 4.17 that π must be an X-joker for β. □

Lemma 4.5 *If β, $\beta_1, \ldots, \beta_n$ are prime formulae, then*

$$\exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\beta_i) \quad \models\!\!\!|_{\mathrm{FT}'} \quad \bigwedge_{i=1}^{n} \exists X(\beta \wedge \neg\beta_i).$$

Proof. Let $\beta, \beta_1, \ldots, \beta_n$ be prime formulae. Then $\exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\beta_i) \models \bigwedge_{i=1}^{n} \exists X(\beta \wedge \neg\beta_i)$ is trivial. To see the other direction, suppose that $\mathfrak{A}$ is a model of FT′ and $\mathfrak{A}, \alpha \models \bigwedge_{i=1}^{n} \exists X(\beta \wedge \neg\beta_i)$. We have to exhibit some X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta$ and $\mathfrak{A}, \alpha' \models \neg\beta_i$ for $i = 1, \ldots, n$.

Without loss of generality we can assume that $\mathfrak{A}, \alpha' \models \exists X(\beta \wedge \beta_i)$ for $i = 1, \ldots, m$ and $\mathfrak{A}, \alpha' \models \neg\exists X(\beta \wedge \beta_i)$ for $i = m+1, \ldots, n$.

By Lemma 4.4 there exists, for every $i = 1, \ldots, m$, an X-joker $\pi_i \in [\beta_i]$ for β. By Lemma 4.3 we have

$$\exists X\beta \models \exists X(\beta \wedge \bigwedge_{i=1}^{m} \neg\pi_i).$$

Since $\neg\pi \models \neg\beta_i$ by Proposition 4.4, we have

$$\exists X\beta \models \exists X(\beta \wedge \bigwedge_{i=1}^{m} \neg\beta_i).$$

Hence we know that there exists an X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta$ and $\mathfrak{A}, \alpha' \models \neg\beta_i$ for $i = 1, \ldots, m$. Since we know that $\mathfrak{A}, \alpha \models \neg\exists X(\beta \wedge \beta_i)$ for $i = m+1, \ldots, n$, we have $\mathfrak{A}, \alpha' \models \neg\beta_i$ for $i = m+1, \ldots, n$. □

Lemma 4.6 *For every two prime formulae β, β' and every set of variables X one can compute a Boolean combination δ of prime formulae such that*

$$\exists X(\beta \wedge \neg\beta') \models\!\!\!|_{\mathrm{FT}'} \delta \quad \text{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\exists X(\beta \wedge \neg\beta')).$$

Proof. Let β, β' be prime formulae, λ be a projection of β', X be a set of variables and $\mathfrak{A}$ be a model of FT′. We distinguish two cases:

1. λ contains an X-joker π for β. Then we know that $\exists X\beta \models \exists X(\beta \wedge \neg\pi)$ by Lemma 4.3. Since $\beta' \models_{\mathrm{FT}'} \lambda \models \pi$, we know that $\neg\pi \models \neg\beta'$ and hence $\exists X\beta \models_{\mathrm{FT}'} \exists X(\beta \wedge \neg\beta')$. Thus

$$\exists X(\beta \wedge \neg\beta') \models\!\!\!\models_{\mathrm{FT}'} \exists X\beta.$$

Now the claim follows with Proposition 4.15.

2. λ contains no X-joker π for β. Then we know by Lemma 4.4 that there exists no valuation α into $\mathfrak{A}$ such that

$$\mathfrak{A}, \alpha \models \exists X(\beta \wedge \beta') \quad \text{and} \quad \mathfrak{A}, \alpha \models \exists X(\beta \wedge \neg\beta').$$

Hence,

$$\exists X(\beta \wedge \neg\beta') \models\!\!\!\models_{\mathrm{FT}'} \exists X\beta \wedge \neg\exists X(\beta \wedge \beta').$$

Now the claim follows with Propositions 4.15, 4.16 and 4.18.

The above shows the existence of δ. Moreover, δ can be computed since we can compute a projection λ of β', and since we can decide whether λ contains an X-joker for β by Proposition 4.18 (λ is finite). □

Theorem 4.3 *For every formula ϕ one can compute a Boolean combination δ of prime formulae such that $\phi \models\!\!\!\models_{\mathrm{FT}'} \delta$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\beta)$.*

Proof. Follows from Lemma 4.1, Propositions 4.16 and 4.15, and Lemmas 4.5 and 4.6. □

Theorem 4.4 FT′ *is a complete and decidable theory.*

Proof. The completeness of FT′ follows from the preceding theorem and the fact that $\top$ is the only closed prime formula. The decidability follows from the completeness and the fact that FT′ is given by a recursive set of sentences. □

4.4.5 Applications of the Simplification Algorithm

As a first application, we want to show that CFT′ is less expressive than FT′, which is established by the existence of a quantifier elimination for FT′. To show that FT′ is less expressive we must show that the arity predicates are not definable in FT′. This claim is a trivial consequence of the following lemma.

Lemma 4.7 *Let $\phi(x)$ be a* FT′*-formula with one free variable x such that* FT′ $\models \exists x\phi(x)$. *If there is some feature g with* FT′ $\models \exists x(\exists y(xgy) \wedge \phi(x))$, *then there are infinitely many features f such that for all constant symbols c,*

$$\mathrm{FT}' \models \exists x(xfc \wedge \phi(x)).$$

Proof. Let $\phi(x)$ be a formula with one free variable, and let $\gamma(x)$ be the corresponding Boolean combination of prime formulae equivalent to $\phi(x)$ which is the result of quantifier elimination. Note that x is the only free variable in $\gamma(x)$ by the definition of the quantifier elimination. Without loss of generality, we can assume that $\gamma(x)$ is in disjunctive normal form. Since prime formulae are closed under conjunction, we can furthermore assume that every disjunct of $\gamma(x)$ is of the form $\beta(x) \wedge \bigwedge_{j=1}^{k} \neg\beta_j(x)$, where $\beta(x), \beta_1(x), \ldots \beta_k(x)$ are prime formulae such that x is the only variable free in $\beta(x), \beta_1(x), \ldots \beta_k(x)$.[2] Furthermore, we can assume that every disjunct of $\gamma(x)$ is satisfiable in FT′.

Now we choose some disjunct $\beta \wedge \bigwedge_{j=1}^{k} \neg\beta_j$ of $\gamma(x)$. Since FT′ $\models \exists x(\exists y(xgy) \wedge \phi)$ for some feature g, we can assume without loss of generality that β does not contain a constraint $x \doteq c$ or $atom(x)$. Let f be an arbitrary feature that is not used in $\beta, \beta_1, \ldots \beta_k$, c be some arbitrary constant symbol and β' be the prime formula equivalent to the conjunction of xfc and β. Note that every $\{x\}$-joker π for β with the property that f is not used in π is also an $\{x\}$-joker for β'. For the claim it is sufficient to proof that

$$\mathrm{FT}' \models \exists x(\beta' \wedge \bigwedge_{j=1}^{k} \neg\beta_j). \tag{4.5}$$

Since $\beta \wedge \bigwedge_{j=1}^{k} \neg\beta_j$ is satisfiable, we know by Lemma 4.4 that for every $j \in 1 \ldots k$ either $\beta \models_{\mathrm{FT}'} \neg\beta_j$ or there exists a $\{x\}$-joker $\pi_j \in [\beta_j]$ for β, which must also be an $\{x\}$-joker for β'. Since $\beta' \models \beta$, a simple argumentation using Lemma 4.3 shows (4.5). □

Corollary 4.1 *The arity constraints are not definable in* FT′, *i.e., for every finite set of features F there is no formula $\phi(x)$ such that $\phi(x) \rightarrow \bigwedge_{f \in F} \exists y\, xfy$ and $\phi(x) \rightarrow xf\!\uparrow$ for $f \notin F$.*

Proof. Follows directly from the last lemma. □

[2] Recall that every closed prime formula is valid in FT′ and hence equivalent to $\top$. This implies that we can assume without loss of generality that $\beta(x), \beta_1(x), \ldots \beta_k(x)$ have x as a free variable.

Finally, we want to give a concrete example of how the quantifier elimination works. Suppose we shall prove that if c_1 and c_2 are two different constant symbols, then

$$\mathrm{FT}' \models \forall x[(xfc_1 \wedge xgc_2) \rightarrow \exists y_1, y_2(xfy_1 \wedge xgy_2 \wedge \neg(y_1 \doteq y_2))]. \tag{4.6}$$

This is the same as showing that $xfc_1 \wedge xgc_2$ entails $\exists y_1, y_2(xfy_1 \wedge xgy_2 \wedge y_1 \neq y_2)$. In the following, we will abbreviate $xfc_1 \wedge xgc_2$ by β, and $xfy_1 \wedge xgy_2$ by β'. Note that both β, β' are prime formulae. The first step is to eliminate the quantifiers $\exists y_1 \exists y_2$. A projection for $y_1 \doteq y_2$ is $y_1\epsilon \downarrow y_2\epsilon$. Since both $y_1\epsilon$ and $y_2\epsilon$ are decided in β' wrt. $\{y_1, y_2\}$, we know that $y_1\epsilon \downarrow y_2\epsilon$ is no $\{y_1, y_2\}$-joker for β'. Hence, we can apply case 2 of lemma 4.6:

$$\begin{gathered}\forall x[\neg\beta \ \vee \ \exists y_1, y_2(\beta' \wedge \neg(y_1 \doteq y_2))] \\ \Downarrow \text{ case 2 of lemma 4.6} \\ \forall x[\neg\beta \ \vee \ (\exists y_1, y_2\beta' \wedge \neg\exists y_1, y_2(\beta' \wedge y_1 \doteq y_2))].\end{gathered}$$

Now $\exists y_1, y_2(\beta' \wedge y_1 \doteq y_2)$ is no prime formula. An equivalent prime formula is $\beta'' = \exists y(xfy \wedge xgy)$. Now we have to eliminate the out-most quantifier $\forall x$, for which purpose we have first to apply some first-order equivalence transformation:

$$\begin{gathered}\forall x[\neg\beta \ \vee \ (\exists y_1, y_2\beta' \wedge \neg\beta'')]. \\ \Downarrow \\ \neg\exists x[\beta \ \wedge \ (\neg\exists y_1, y_2\beta' \vee \beta'')]. \\ \Downarrow \\ \neg[\exists x(\beta \wedge \neg\exists y_1, y_2\beta') \ \vee \ \exists x(\beta \wedge \beta'')].\end{gathered}$$

Since $c_1 \neq c_2$, we get $\exists x(\beta \wedge \beta'') = \exists x(\beta \wedge \exists y(xfy \wedge xgy)) \models\!\!\!|_{\mathrm{CFT}'} \bot$. Hence, we have to consider only $\neg\exists x(\beta \wedge \neg\exists y_1, y_2\beta')$. Now a projection λ for $\exists y_1, y_2\beta'$ is $\{xf\downarrow, xg\downarrow\}$. Since $\lambda \subseteq [\beta]$, we can again apply case 2 of lemma 4.6, yielding

$$\neg[\exists x\beta \wedge \neg\exists x(\beta \wedge \exists y_1, y_2\beta')].$$

But $\exists x\beta \models\!\!\!|_{\mathrm{CFT}'} \top$ and $\exists x(\beta \wedge \exists y_1, y_2\beta') \models\!\!\!|_{\mathrm{CFT}'} \top$, which implies that we get $\neg[\top \wedge \neg\top]$, which is the same as $\neg\bot$ or $\top$. This proves (4.6).

4.5 Adding Arity Constraints: CFT′

4.5.1 The Axioms

The language of CFT′ contains in addition to FT′ for every finite set of features $F \subseteq \mathcal{L}$ a unary predicate xF (called **arity**), which is written in postfix notation. The theory CFT′ has the following axiom schemes:

(Ax1)	$\tilde{\forall}(xfy \wedge xfz \rightarrow y \doteq z)$	for every feature f.
(Ax2)	$\tilde{\forall}(cfx \rightarrow \bot)$	for all constants c.
(Ax3)	$\neg(c_1 \doteq c_2)$	if c_1 and c_s are different constants
(Ax4)	$\tilde{\forall}(xF \wedge xfy \rightarrow \bot)$	if $f \notin F$.
(Ax5)	$cF \rightarrow \bot$	for every constant c and arity F.
(Ax6)	$\tilde{\forall}(xF \rightarrow \exists y(xfy))$	if $f \in F$ and $x \neq y$.

The first three axiom schemes are the same as in FT_0 and FT′. The last three axiom schemes handle the arity constraints. They guarantee that if x has arity F, then exactly the features $f \in F$ are defined on x.

In order to achieve a complete theory, we must add an axiom scheme that is similar to the axiom (Ax6) of FT′. In contrast to FT′, it is not enough to guarantee that solved forms are consistent in the intended models. Consider the formula

$$x\{f\} \wedge xfx.$$

Then there exists exactly one element of $\mathfrak{T}_{\mathrm{CFT}'}$ (resp. $\mathfrak{R}_{\mathrm{CFT}'}$) that satisfies this description. The uniqueness of the solution of such descriptions must also be expressed in the axioms. Note that it was not possible to fix one element of the domain in the theory FT′ since we cannot restrict the arities of the variables in FT′. The axiom scheme that guarantees both the existence and under certain conditions also the uniqueness of solutions of solved forms was first introduced by [ST94]. They also introduced a complete axiomatisation for CFT in this paper without actually proving completeness. Before stating the required axiom scheme, we will recall the important notion of a determinant as presented in [ST94].

Definition 4.14 (Determinant) *A* determinant for x *is a formula of the form*

$$x\{f_1, \ldots, f_n\} \wedge xf_1t_1 \wedge \ldots \wedge xf_nt_n,$$

where each t_i is a variable or constant. We will write the above formula for convenience as

$$x \doteq (f_1 : t_1, \ldots, f_n : t_n).$$

Given a basic constraint ϕ, we say that x is determined in ϕ *if ϕ contains a determinant for x. A* determinant for *pairwise distinct variables $x_1, \ldots, x_n$ is a conjunction*

$$x_1 \doteq D_1 \wedge \ldots \wedge x_n \doteq D_n,$$

where $D_1, \ldots, D_n$ are determinants for $x_1, \ldots, x_n$. For a basic constraint ϕ we define $\mathcal{D}(\phi)$ to be the set of variables that are determined in ϕ.

The variables in $\mathcal{V}(\delta)\backslash\mathcal{D}(\delta)$ are called the **parameters** of δ. Now we can define the last axiom scheme as introduced by [ST94], which states that for every valuation of the parameters of a determinant δ there is exactly one valuation for the variables determined by δ:

(Ax7) $\tilde{\forall}(\exists! D(\delta)\delta)$ if δ is a determinant.

An example of an instance of scheme (Ax7) is

$$\forall y, z, w \exists! x, u, v \begin{pmatrix} x & \doteq & (f\!:u \;\; g\!:v) \\ u & \doteq & (h\!:x \;\; g\!:y \;\; f\!:z) \\ v & \doteq & (g\!:z \;\; h\!:w) \end{pmatrix}$$

The theory CFT′ consists of the axiom schemes $(Ax1)$–$(Ax7)$.

Proposition 4.19 *The structures $\mathfrak{T}_{\mathrm{CFT}'}$ and $\mathfrak{R}_{\mathrm{CFT}'}$ are models of* CFT′.

Proof. That the first six axioms schemes are satisfied is obvious. To show that $\mathfrak{T}_{\mathrm{CFT}'}$ (resp. $\mathfrak{R}_{\mathrm{CFT}'}$) satisfies the last axiom scheme, one assumes arbitrary feature trees for the universally quantified variables and constructs feature trees for the existentially quantified variables. □

We can also define a feature graph interpretation $\mathfrak{G}_{\mathrm{CFT}'}$ of the language CFT′, where

- the universe of $\mathfrak{G}_{\mathrm{CFT}'}$ and the interpretations of *atom*, the constant symbols and the feature symbols are defined as for the feature graph interpretation $\mathfrak{G}$ of FT′ (see Section 4.4.2, page 77), and
- $\overline{(x,\gamma)} \in F^{\mathfrak{G}_{\mathrm{CFT}'}}$ iff $F = \{f \mid \exists t : xft \in \gamma\}$.

Proposition 4.20 *The feature graph structure $\mathfrak{G}_{\mathrm{CFT}'}$ is no model of the theory* CFT′.

Proof. $\mathfrak{G}_{\mathrm{CFT}'}$ does not satisfy the axiom scheme $(Ax7)$. Consider the two different feature graphs

$$\sigma = \overline{(x, xfy \wedge yh1 \wedge xgz \wedge zh1)} \quad \text{and} \quad \sigma' = \overline{(x, xfy \wedge yh1 \wedge xgy)},$$

and let $\tau = \overline{(y, yh1)}$. Both σ and σ' have the arity $\{f, g\}$ and τ as a subgraph under the features f and g. Hence, $\mathfrak{G}_{\mathrm{CFT}'} \not\models \forall y, z \exists! x (x\{f, g\} \wedge xfy \wedge xgz)$. □

4.5.2 Solved Formulae, Congruences and Normaliser

For CFT′ we have to redefine the notion of a solved form, since the arity constraints are no free predicates in CFT′.

Definition 4.15 (Solved Formula) *A basic constraint γ is a* solved formula *if*

1. *no atomic constraint occurs twice in ϕ;*
2. *an equation $x \doteq t$ appears in γ if and only if γ eliminates x;*
3. *if $xft \in \phi$ and $xft' \in \phi$, then $t = t'$;*
4. *if $xF, xG \in \phi$, then $F = G$;*
5. *if $xF \in \phi$ and $f \notin F$, then $xfy \notin \phi$;*
6. *ϕ does not contain an atomic formula of the form $c \doteq t$, cF or cft.*

Every solved form γ has a unique decomposition $\gamma = \gamma_N \wedge \gamma_G$ into a possibly empty conjunction γ_N of equations "$x \doteq y$" and a possibly empty conjunction γ_G of constraints "xF" and feature constraints "xfy". We call γ_N the normaliser *and γ_G the* graph *of γ.*

Proposition 4.21 *Let γ be the graph of a solved formula. A variable x is called* constrained *in γ if γ contains a constraint xft or xF. Let $\mathcal{CV}(\gamma)$ be the set of all variables constrained in γ. Then*

$$\mathrm{CFT}' \models \tilde{\forall}\exists\mathcal{CV}(\gamma)\gamma$$

Proof. We will extend γ to a determinant δ with $\mathcal{D}(\delta) = \mathcal{CV}(\gamma)$.

For every $x \in \mathcal{CV}(\gamma)$ and $x \notin \mathcal{D}(\gamma)$ let F_x be a set of features such that F_x contains exactly the features f with $xfy \in \gamma$ and let δ be defined as

$$\delta = \gamma \cup \{xF_x \mid x \in \mathcal{CV}(\gamma)\}$$

By definition, δ is a determinant. By axiom $(Ax7)$ we know that

$$\mathrm{CFT}' \models \tilde{\forall}\exists\mathcal{D}(\delta)\delta$$

which proves $\mathrm{CFT}' \models \tilde{\forall}\exists\mathcal{CV}(\gamma)\gamma$. □

Again, every basic constraint is equivalent in CFT′ to either $\bot$ or a solved formula. The basic simplification rules for achieving a solved form are the basic simplification rules of FT_0 (see figure 4.1, page 69) plus the following clash rules:

$$\text{(ArCl)} \quad \frac{xF \wedge xG \wedge \phi}{\bot} \quad F \neq G$$

$$\text{(FArCl)} \quad \frac{xfy \wedge xF \wedge \phi}{\bot} \quad f \notin F$$

$$\text{(CArCl)} \quad \frac{cF \wedge \phi}{\bot}$$

We say that a basic constraint **clashes** if it can be reduced to $\bot$ with one of the clash-rules (i.e., one of the rules (CCl), (CFCl1), (CFCl2), (ArCl), (FArCl), or (CArCl)). We say that a basic constraint is **clash-free** if it does not clash.

Proposition 4.22 *The basic simplification rules for* CFT′ *are terminating and perform equivalence transformations with respect to* CFT′*. Moreover, a basic formula $\phi \neq \bot$ is solved if and only if no basic simplification rule applies to it.*

Proposition 4.23 *Let ϕ be a formula built from atomic formulae with conjunction. Then one can compute a formula δ that is either solved or $\bot$ such that $\phi \models\!\!\!\models_{\mathrm{CFT}'} \delta$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\phi)$.*

Proof. Follows from the preceding proposition and the fact that the basic simplification rules do not introduce new variables. □

In the completeness proof for FT′ we have defined the notion of normaliser, which was the set of equations attached to a solved formula. For CFT′ we need a more detailed definition of a normaliser. To this end we use the notion of congruence of a basic constraint. The definitions of congruence and normalisers are taken out of [ST94], where they have been defined and used for the first time.

A **congruence of a basic constraint** ϕ is an equivalence relation $\approx$ on terms satisfying the following:

- $t_1 \doteq t_2 \in \phi$ implies $t_1 \approx t_2$
- $t_1 f t_2, t_1' f t_2' \in \phi$ and $t_1 \approx t_1'$ implies $t_1 \approx t_2'$.

It is easy to see that the set of congruences of a basic constraint is closed under intersection. Since the equivalence relation identifying all terms is a congruence for every basic constraint, we know that every basic constraint has a least congruence.

It will be convenient to represent congruences as idempotent substitutions. Since our congruences also relates constants, we define a substitution to be a function on the set of terms.

Definition 4.16 *A* normaliser of a congruence $\approx$ *is an idempotent substitution* θ *that satisfies*

$$\forall t_1, t_2 : (\theta(t_1) = \theta(t_2) \Leftrightarrow t_1 \approx t_2).$$

We say that substitution θ is finite if there are only finitely many terms t with $\theta(t) \neq t$. A finite substitution can be represented as

$$\bigwedge\{t \doteq \theta(t) \mid t \neq \theta(t)\}.$$

For convenience, we will simply use θ to denote this formula. Clearly, for every basic constraint ϕ and every substitution θ we have

$$\theta \wedge \phi \models\!\!\!\models \theta \wedge \theta\phi.$$

Definition 4.17 (Normaliser) *A* normaliser *of a basic constraint* ϕ *is a normaliser of the least congruence of* ϕ.

We will now recall some properties of normalisers that have been proven in [ST94]. A **graph constraint** is a basic constraint that contains no equations. A graph constraint is called a **graph** if it is a solved formula. We say for a substitution θ and a graph constraint ϕ that $\theta\phi$ clashes if either θ clashes or the result of applying θ to ϕ clashes. Note that for every normaliser θ which is clash-free we can assume without loss of generality that $\theta(c) = c$ for every constant symbol c (which we will do henceforth).

Proposition 4.24 *Let* $\mathfrak{A}$ *be a model of* CFT′, ϕ *a basic constraint and* θ *a normaliser of* ϕ. *Then* ϕ *is unsatisfiable in* $\mathfrak{A}$ *if and only if* $\theta\phi_G$ *clashes, where* ϕ_G *is a graph constraint containing all constraints of* ϕ *of the form* tF *and* tft'.

Proposition 4.25 *Let* $\gamma = \gamma_G \wedge \gamma_N$ *be the normal form of a basic constraint* ϕ *that is normal with respect to the rules (Triv), (Cong), (Orient) and (Elim). Then* $\theta = \gamma_N$ *is a normaliser of* ϕ *satisfying* $\gamma_G = \theta\gamma_G$ *and* $\mathcal{V}(\theta) \subseteq \mathcal{V}(\phi)$.

This proposition allows us to calculate normalisers. Note that this also implies that for a solved formula the two notions of normaliser as defined in Definition 4.15 and in Definition 4.17 agree.

Definition 4.18 (Saturated Formula) *A basic constraint ϕ is called saturated if for every arity constraint $xF \in \phi$ and every feature $f \in F$ there exists a feature constraint $xft \in \phi$.*

Lemma 4.8 *Let γ be a saturated graph constraint and θ be a normaliser of some congruence of γ. If $\theta\gamma$ is clash-free and if $\mathcal{V}(\theta) \subseteq \mathcal{D}(\gamma)$, then*

$$\gamma \models_{\mathrm{CFT}'} \theta.$$

For our purposes, we need two additional propositions.

Proposition 4.26 *Let the substitution θ be a normaliser of some congruence of a graph constraint ϕ such that $\theta\phi$ is clash-free. Then $\theta\phi$ is a graph.*

Proposition 4.27 *Let θ be the normaliser of some congruence of graph γ and let $\theta = \theta' \cup \theta''$ be a partition of θ. If θ' is a normaliser of some congruence of γ, then θ'' is a normaliser of some congruence of $\theta'\gamma$.*

Proof. Let γ, θ, θ', and θ'' be given as described. If θ' is a normaliser of some congruence of γ, we have to show that θ'' is a normaliser of some congruence of $\theta'\gamma$. Clearly, θ'' is an idempotent substitution. The congruence property follows from the fact that $\theta''(\theta'(x)) = \theta(x)$. □

4.5.3 Prime Formulae

We now define a class of prime formula for the theory CFT′ that have the properties as required by lemma 4.1.

Definition 4.19 (Prime Formula) *Let ϕ be a basic constraint. A formula $\beta = \exists X\phi$ is called* prime *if it satisfies the following conditions:*

1. *ϕ is solved and saturated;*
2. *X has no variable in common with the normaliser of ϕ;*

3. for every $x \in X$ there is a variable $y \in \mathcal{V}(\beta)$ and a path p such that $ypx \in [\phi]$.

The letter β will always denote a prime formula if nothing else is stated. Again, $\top$ is the only closed prime formula. Note that we can use all definitions and propositions of section 4.3 since a solved formula in CFT′ is also a solved formula as defined for $\mathrm{FT}_0^{F_1,\ldots,F_i,\ldots}$ and $\mathrm{FT}_0^{F_1,\ldots,F_i,\ldots} \subseteq \mathrm{CFT}'$, where $F_1, \ldots, F_i, \ldots$ is an enumeration of all arity constraints.

As in the completeness proof for FT′, we have to define the notion of decided variables in order to show that every existential quantified basic constraint can be transformed into a prime formula. Decided variables of an existential quantified formula are those variables, which are reachable from a free variable via a feature path. We have shown for FT′, that every existential quantified solved formula is equivalent to the set of constraints on the decided variables. But for CFT′, we will define a more general notion of decidedness, which is more appropriate for the proofs to come.

Definition 4.20 (Decided Variables) *Let γ be some solved formula, and let $\psi = \exists X\gamma$. A variable $x \in \mathcal{V}(\gamma)$ is said to be* explicitly decided *in ψ if there is a variable y free in ψ and a path p such that*

$$ypx \in [\gamma].$$

A variable $x \in \mathcal{V}(\gamma)$ is called implicitly decided *in ψ if γ contains a determinant D for x where each parameter of D is explicitly decided in ψ. We say that $x \in \mathcal{V}(\gamma)$ is* decided *in ψ if there is a z with $x\varepsilon z \in [\gamma]$ and z is explicitly or implicitly decided in ψ.*

We say that a variable is undecided if it is not decided. The set of decided variables of a formula ψ will be denoted by $\mathcal{D}ec(\psi)$. The set of explicitly decided variables is denoted by $\mathcal{D}ec_e(\psi)$. Note that if $\exists X\gamma$ is a prime formula, then every variable in $\mathcal{V}(\gamma)$ is explicitly decided. For the formula

$$\psi = \exists x, x_1, x_2(xfy \wedge x_1\{f, g\} \wedge x_1fy \wedge x_1gx_2 \wedge zhx_2)$$

we get $\mathcal{D}ec_e(\psi) = \{y, z, x_2\}$ and $\mathcal{D}ec(\psi) = \mathcal{D}ec_e(\psi) \cup \{x_1\}$. The variable x is the only one which is undecided in ψ.

Note that the explicitly decided variables are the variables we have called decided in the completeness proof for FT′.

Proposition 4.28 *Let γ be a solved formula, $\psi = \exists X\gamma$, and Y be the subset of X containing all variables that are decided in ψ. Then for every valuation α into a* CFT′ *model $\mathfrak{A}$ with $\mathfrak{A}, \alpha \models \psi$ there exists a unique Y-update α' of α such that*

$$\mathfrak{A}, \alpha' \models \exists X \backslash Y \gamma.$$

Proposition 4.29 *Let γ be a solved formula and X be a set of variables. If x is a variable that is decided in $\exists X\gamma$ and γ contains a constraint xfy, then y is also decided in $\exists X\gamma$.*

The following lemmas and propositions will show that we can transform every existential quantified basic constraint into a prime formula. A constraint c is called a **constraint for** x if c is of the form xft, xF or $x \doteq t$. We will say that two formulae $\psi = \exists X\gamma$ and $\psi' = \exists X'\gamma'$ **differ only on the undecided variables** if $\mathcal{V}(\psi) = \mathcal{V}(\psi')$, $\mathcal{D}ec_e(\psi) = \mathcal{D}ec_e(\psi')$, and ψ and ψ' contain exactly the same constraints for the explicitly decided variables.

Lemma 4.9 (Garbage Collection) *Let $\psi = \exists X\gamma$ and $\psi' = \exists X'\gamma'$ be existentially quantified solved formulae that differ only on the undecided variables. Then*

$$\psi \models\!\!\models_{\mathrm{CFT}'} \psi'.$$

Proof. Let $Y = \mathcal{D}ec_e(\psi) \cap X = \mathcal{D}ec_e(\psi') \cap X'$, $Z = X \backslash Y$ and $Z' = X' \backslash Y$. Y contains the existentially quantified, explicitly decided variables, whereas Z (resp. Z') contains the variables that are not explicitly decided in ψ (resp. ψ'). We will show that there is a possible empty conjunction of equations ϕ such that

$$\exists X\gamma \models\!\!\models_{\mathrm{CFT}'} \exists Y(\phi \wedge \exists Z\gamma_G) \quad \text{and} \quad \exists X\gamma \models\!\!\models_{\mathrm{CFT}'} \exists Y(\phi \wedge \exists Z'\gamma'_G). \tag{4.7}$$

Once we have shown this, the lemma can be proven as follows. Since $\mathcal{V}(\exists Z\gamma_G) \subseteq \mathcal{D}ec_e(\psi)$, we know that every variable explicitly decided in $\exists Z\gamma_G$ must also be explicitly decided in ψ: A variable x is explicitly decided in $\exists Z\gamma_G$ if there is a variable $y \in \mathcal{V}(\exists Z\gamma_G)$ with $ypx \in [\gamma_G]$. Since $y \in \mathcal{D}ec_e(\psi)$, we know that there is variable $z \in \mathcal{V}(\psi)$ with $zqy \in [\gamma]$ for some path q. Hence, $zpqx \in [\gamma]$, which implies that z is explicitly decided in ψ.

Similarly we can show that $\mathcal{D}ec_e(\exists Z'\gamma'_G) \subseteq \mathcal{D}ec_e(\psi)$. This implies that $(\exists Z\gamma_G)$ and $(\exists Z'\gamma'_G)$ are graphs that do not differ on the decided variables. An adaption of the proof of lemma 4.2 shows that

$$\exists Z\gamma_G \models\!\!\models_{\mathrm{CFT}'} \exists Z'\gamma'_G,$$

which proves $\psi \models\!|_{\mathrm{CFT}'} \psi'$.

For the proof of (4.7) let ϕ be the subset of equations $x \doteq t$ in $\gamma_N \cap \gamma'_N$ with $\mathcal{V}(x \doteq t) \subseteq \mathcal{D}ec_e(\psi)$. Then all variables occuring on the left side of an equation in $\gamma_N \backslash \phi$ (resp. $\gamma'_N \backslash \phi$) cannot be explicitly decided in ψ (resp. ψ'). Since γ and γ' eliminate the variables on the left side of the equations, we get

$$\exists X\gamma \models\!| \exists X(\phi \wedge \gamma_G) \quad \text{and} \quad \exists X\gamma' \models\!| \exists X(\phi \wedge \gamma'_G)$$

Now (4.7) follows from the fact that $\mathcal{V}(\phi) \cap Z = \emptyset$ and $\mathcal{V}(\phi) \cap Z' = \emptyset$. □

Proposition 4.30 *For every prime formula β and every set of variables X one can compute a prime formula β' such that*

$$\exists X\beta \models\!|_{\mathrm{CFT}'} \beta' \quad \text{and} \quad \mathcal{V}(\beta') \subseteq \mathcal{V}(\exists X\beta).$$

Proof. See proof of proposition 4.15. □

Proposition 4.31 *For every two prime formulae β and β' one can compute a formula δ that is either prime or $\perp$ and satisfies*

$$\beta \wedge \beta' \models\!|_{\mathrm{CFT}'} \delta \quad \text{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\beta \wedge \beta').$$

Proof. See proof of proposition 4.16. □

4.5.4 Proof of the Main Lemmas

In this section we will show that our prime formulae for CFT′ satisfy requirements (5) and (6) of lemma 4.1. The procedure is similar to the one in the proof for FT′, i.e. we will define the central notion of X-jokers.

Definition 4.21 *A rooted path xp is said to be* determined *in $\beta = \exists X\gamma$ if $|xp|_\gamma$ is defined and $|xp|_\gamma \in \mathcal{D}(\gamma)$.*

Proposition 4.32 *Let $\beta = \exists X\phi$ be some prime formula and $x \in \mathcal{D}(\gamma)$ be an variable that is undecided in β. Then there is a variable y and path p such that $xpy \in [\gamma]$, $y \notin \mathcal{D}(\gamma)$ and y is undecided in β.*

Proof. Since x is in $\mathcal{D}(\gamma)$ *and* is undecided, every determinant $\delta \subseteq \gamma$ with $x \in \mathcal{D}(\delta)$ must contain an undecided parameter. Now let δ be the largest determinant such that $\delta \subseteq \gamma$, $x \in \mathcal{D}(\delta)$ and for every $z \in \mathcal{V}(\delta)$ there is a path p with

$$xpz \in [\gamma].$$

Such a determinant must exists since β is saturated. Now let y be one parameter of δ that is undecided. y cannot be determined in γ. If γ would contain a determinant D for y, then $\delta' = \delta \wedge y \doteq D$ would be a determinant that is larger than δ and satisfies $\delta' \subseteq \gamma$, $x \in \mathcal{D}(\delta')$ and $\forall z \in \mathcal{V}(\delta')\exists p : xpz \in [\gamma]$. Hence, y is the variable we have searched for. □

Definition 4.22 *Let $\beta = \exists Y\gamma$ be a prime formula and X be a set of variables. A rooted path xp is said to be* decided in β wrt. X *if either $x \notin X$ or there is some prefix p' such that xp' is realized and $|xp'|_\gamma$ is constant or a variable that is decided in $\exists X\beta$.*

Note that this definition differs from the one in 4.12, as the decided variables in FT′ are the explicitly decided variables in CFT′.

Proposition 4.33 *If π is a proper path constraint such that all rooted paths contained in π are decided in β wrt. X, then either $\mathfrak{A}, \alpha \models \forall X(\beta \to \pi)$ or $\mathfrak{A}, \alpha \models \forall X(\beta \to \neg\pi)$.*

Definition 4.23 (X-Joker) *Let $\beta = \exists Y\phi$ be a prime formula and X be a set of variables. We say that a rooted path xp is* free *in β wrt. X if xp is neither determined in β nor decided in β wrt. X. A proper path constraint is called an* X-joker for β *if $\pi \notin [\beta]$ and one of the following conditions is satisfied:*

- *$\pi = xpc$ and xp is free in β wrt. X,*
- *$\pi = atom(xp)$ and xp is free in β wrt. X,*
- *$\pi = xp \downarrow yq$ and xp is free in β wrt. X,*
- *$\pi = yq \downarrow xp$ and xp is free in β wrt. X.*

This definition and the definition of X-jokers for FT′ differ in that an X-joker for CFT′ must contain an undecided rooted path that is additionally undetermined.

Thus, the path constraint $\pi = xf \downarrow xg$ is no $\{x\}$-joker for the formulae

$$\beta_1 = \exists x_1, x_2, x_3, x_4 \left(\begin{array}{l} xfx_1 \wedge xgx_2 \wedge \\ x_1\{f\} \wedge x_1fx_3 \wedge \\ x_2\{f\} \wedge x_2fx_4 \end{array} \right)$$

$$\beta_2 = \exists x_1, x_2 \left(\begin{array}{l} xfx_1 \wedge xgx_2 \wedge \\ x_1\{f\} \wedge x_1fx_1 \wedge \\ x_2\{f\} \wedge x_2fx_2 \end{array} \right)$$

But we can calculate an $\{x\}$-joker π' for β_1 with the property $\beta_1 \wedge \pi \models_{\mathrm{CFT}'} \pi'$, namely the constraint $xff \downarrow xgf$. On the other hand, π' is no $\{x\}$-joker for β_2, and there exists no $\{x\}$-joker π'' with the property that $\beta_2 \wedge \pi \models_{\mathrm{CFT}'} \pi''$. The calculation of entailed X-jokers is the subject of Lemma 4.10. Note that the differences between the X-jokers for FT′ and CFT′ is also reflected in the following observation. In FT′, given a proper path constraint π such that there is some FT′ model $\mathfrak{A}$ and a valuation α with

$$\mathfrak{A}, \alpha \models \exists X(\beta \wedge \pi) \quad \text{and} \quad \mathfrak{A}, \alpha \models \exists X(\beta \wedge \neg\pi),$$

then π must be an X-joker (see lemma 4.4). For CFT′, this does not hold.

Proposition 4.34 *It is decidable whether a rooted path is free in a prime formula wrt. a set of variables, and whether a path constraint is an X-joker for a prime formula.*

Proof. Follows from proposition 4.3. □

Lemma 4.10 *Let $\beta = \exists Y\gamma$ be a prime formula and π be a proper path constraint. Then either we can calculate an X-joker π' for β with*

$$\beta \wedge \pi \models \pi'$$

or for every CFT′ *model $\mathfrak{A}$ and every valuation α we have*

$$\mathfrak{A}, \alpha \models \forall X(\beta \to \pi) \quad \textit{or} \quad \mathfrak{A}, \alpha \models \forall X(\beta \to \neg\pi).$$

Proof. Without loss of generality we can assume that $\mathcal{V}(\pi) \cap Y = \emptyset$. If π is an element of $[\beta]$, then $\beta \models_{\mathrm{CFT}'} \pi$ by proposition 4.4. If the normal form of $\beta \wedge \pi$ is $\bot$, then $\beta \models \neg\pi$. If both fail, then we distinguish the cases listed below. We will say that a rooted path xp is decided to mean that xp is decided in β wrt. X, and we will use the term undecided in a similar way. Analogous, we will say that a variable is (un-)decided if it is (un-)decided in $\exists X\beta$.

1. *all rooted paths contained in π are decided.* Then proposition 4.33 shows that for every CFT′ model $\mathfrak{A}$ and every α either $\mathfrak{A}, \alpha \models \forall X(\beta \to \pi)$ or $\mathfrak{A}, \alpha \models \forall X(\beta \to \neg\pi)$.

2. *π contains a rooted path xp that is undecided and not realized in β.* Then $xp{\downarrow}$ is an X-joker since γ is saturated.

3. *π contains at least one undecided rooted path, and the undecided rooted paths contained in π are realized in β.* We will split up this case as follows:

3.a $\pi = xp{\downarrow}$. Then π is in $[\beta]$.

3.b $\pi = xpc$ *and xp is undecided but realized in β.* By our assumption we can assume that xp is not determined in β since this would imply $\beta \wedge \pi \models\!\!\!|_{\mathrm{CFT}'} \bot$. Hence, π must be an X-joker.

3.c $\pi = xpF$. Analogous to case (3.b).

3.d $\pi = atom(xp)$. Analogous to case (3.b).

3.e $\pi = xp \downarrow yq$ *and xp is decided and yq is undecided.* Then yq is realized. If yq is undetermined in β, then π is an X-joker.

Otherwise let $z = |yq|_\gamma$ with $z \in \mathcal{D}(\gamma)$. Since z is undecided, proposition 4.32 shows that there is a variable $u \notin \mathcal{D}(\gamma)$ that is undecided and a path r such that $zru \in [\gamma]$. Then yqr is a rooted path that is both undecided and not determined in β.

Now $|xpr|_\gamma$ must be either undefined or a variable z' with $z' \neq z$, since otherwise u would be a decided variable. Hence, $\pi' = xpr \downarrow yqr$ is not in $[\beta]$. This shows that π' is an X-joker with $\pi \models\!\!\!|_{\mathrm{CFT}'} \pi'$.

3.f $\pi = xp \downarrow yq$ *and both xp and yq are undecided.* Then xp and yq are realized in β.

If $|xp|_\gamma$ (resp. $|yq|_\gamma$) is not an element of $\mathcal{V}(\gamma_G)$, then xp (resp. yq) is not determined in β, which implies that π is an X-joker.

Otherwise, let @ be some access function of β and θ be a normaliser of $\gamma_G \wedge |xp|_\gamma \doteq |yq|_\gamma$. Note that

$$\beta \wedge \pi \models\!\!\!|_{\mathrm{CFT}'} \exists Y(\gamma_N \wedge \gamma_G \wedge |xp|_\gamma \doteq |yq|_\gamma)$$

and

$$\gamma_G \wedge |xp|_\gamma \doteq |yq|_\gamma \;\models\!\!\!|_{\mathrm{CFT}'}\; \gamma_G \wedge \theta.$$

Since $\mathcal{V}(|xp|_\gamma \doteq |yq|_\gamma) \subseteq \mathcal{V}(\gamma_G)$, we can assume by proposition 4.25 that $\mathcal{V}(\theta) \subseteq \mathcal{V}(\gamma_G)$. Since γ_N eliminates the variable on the left side of the equations, this implies

$$\gamma_N \wedge \gamma_G \wedge |xp|_\gamma \doteq |yq|_\gamma \models\!\!\!|_{\mathrm{CFT}'} \gamma_N \wedge \gamma_G \wedge \theta.$$

Furthermore, we can assume without loss of generality that θ contains no trivial equations of form $z \doteq z$. Hence, $@z_1 \downarrow @z_2 \notin [\beta]$ for every equation $z_1 \doteq z_2$ in θ. Since we have assumed $\beta \wedge \pi \not\models_{\mathrm{CFT}'} \bot$, we know that $\theta\gamma_G$ is clash-free.

If θ contains an equation $z \doteq c$ where z is undecided, then $z \in \mathcal{V}(\gamma_G)$. Now z cannot be determined in γ_G as $\theta\gamma_G$ is clash-free. Hence, $@zc$ is an X-joker π' with $\beta \wedge \pi \models_{\mathrm{CFT}'} \pi'$.

If θ contains an equation $z_1 \doteq z_1$ or $z_2 \doteq z_1$ where z_1 is undecided and z_2 is decided, then $\pi' = @z_1 \downarrow @z_2$ is a proper path constraint with $\beta \wedge \pi \models \pi'$. Furthermore, we can apply case (3.e) on π' yielding an X-joker π'' with $\beta \wedge \pi \models \pi''$.

If θ contains an equation $z_1 \doteq z_1$ or $z_2 \doteq z_1$ where z_1 and z_2 are undecided and z_1 is not determined in γ_G, then $\pi' = @z_1 \downarrow @z_2$ is an X-joker with $\beta \wedge \pi \models \pi'$.

The remaining case is that θ contains only equations of the form $z \doteq c$ with z decided or equations of the form $z_1 \doteq z_2$ where either both variables are decided or both variables are undecided but determined in γ_G. We will show that in this case $\mathfrak{A}, \alpha \models \exists X(\beta \wedge \pi)$ implies $\mathfrak{A}, \alpha \models \forall X(\beta \to \pi)$.

Now assume that $\mathfrak{A}, \alpha \models \exists X(\beta \wedge \pi)$. We will show that then

$$\mathfrak{A}, \alpha \models \forall X \forall Y(\gamma_G \to \theta). \tag{4.8}$$

This implies that $\mathfrak{A}, \alpha \models \forall X \forall Y(\gamma_N \wedge \gamma_G \to |xp|_\gamma \doteq |yq|_\gamma)$, which is equivalent to $\mathfrak{A}, \alpha \models \forall X(\exists Y \gamma \to \pi)$.

Let θ' be the subset of θ containing all equations among decided variables, and let θ'' be the rest of θ. θ'' contains only equations between variables that are determined in γ_G. It is easy to check that θ' is a normaliser of some congruence of γ_G. This implies by proposition 4.26 that $\theta'\gamma_G$ is a solved graph.

Let α' be the unique extension of α to the variables that are decided, and let $Z \subseteq X \cup Y$ be the set of undecided variables. Clearly, $\mathfrak{A}, \alpha' \models \theta'$. Furthermore, $\mathcal{V}(\theta') \cap Z = \emptyset$. This implies

$$\mathfrak{A}, \alpha' \models \forall Z(\gamma_G \leftrightarrow \theta'\gamma_G).$$

Since θ'' is a normaliser of some congruence of $\theta'\gamma_G$ by proposition 4.27, $\theta'\gamma_G$ is a solved graph and $\mathcal{V}(\theta'') \subseteq \mathcal{D}(\theta'\gamma_G)$, we know by lemma 4.8 that

$$\theta'\gamma_G \models \theta''.$$

Hence, $\mathfrak{A}, \alpha' \models \forall Z(\gamma_G \to \theta' \wedge \theta'')$, which implies $\mathfrak{A}, \alpha' \models \forall Z(\gamma_G \to \pi)$. From this follows (4.8) as α' was the unique update of α to $\mathcal{D}ec(\exists X\beta)$.

□

Corollary 4.2 *Let β be a prime formula and π be a proper path constraint. If there is a* CFT′ *model $\mathfrak{A}$ and a valuation α into $\mathfrak{A}$ with*

$$\mathfrak{A}, \alpha \models \exists X(\beta \wedge \pi) \quad \textit{and} \quad \mathfrak{A}, \alpha \models \exists X(\beta \wedge \neg\pi),$$

then we can calculate an X-joker for β with $\beta \wedge \pi \models \pi'$.

Lemma 4.11 *Let $\beta = \exists Y\gamma$ be a prime formula and $\pi_1, \ldots, \pi_n$ be X-jokers for β. Then*

$$\exists X\beta \models_{\mathrm{CFT}'} \exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\pi_i)$$

Proof. Let $\beta = \exists Y\gamma$ be a prime formula, $\pi_1, \ldots, \pi_n$ $(n > 0)$ be X-jokers for β, $\mathfrak{A}$ be some model of CFT′, and α be some valuation into $\mathfrak{A}$ with $\mathfrak{A}, \alpha \models \exists X\beta$. We have to show that $\mathfrak{A}, \alpha \models \exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\pi)$. We will define a prime formula β' satisfying the following:

- $\beta' \models \beta$,
- $\exists X\beta \models\!|_{\mathrm{CFT}'} \exists X\beta'$,
- $\mathfrak{A}, \alpha \models \forall X(\beta' \to \neg\pi_i)$ for all $i = 1..n$

Once we have defined a β' satisfying these conditions, we can prove the claim using the following argumentation. Since $\exists X\beta \models\!|_{\mathrm{CFT}'} \exists X\beta'$ and $\mathfrak{A}, \alpha \models \exists X\beta$, there must be an X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta'$. But as $\beta' \models \beta$ and for all $i = 1..n$ $\mathfrak{A}, \alpha \models \forall X(\beta' \to \neg\pi_i)$, we know that $\mathfrak{A}, \alpha' \models \beta \wedge \bigwedge_{i=1}^{n} \neg\pi_i$. This shows $\mathfrak{A}, \alpha \models \exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\pi_i)$.

In the following, we will just say that a rooted path xp is decided when meaning that xp is decided in β wrt. X, and we will use undecided in a similar way. Let $Z \subseteq \mathcal{V}(\gamma_G)$ be the set of all variables of the graph of γ that are undecided and undetermined. Note that if a rooted path xp is undecided, undetermined and realized in β, then $|xp|_\gamma$ is a variable z with $z \in Z$. Furthermore, let $Z_C \subseteq Z$ be the set of variables $z \in Z$ with $atom(z) \in \gamma$.

By Proposition 4.28 there exists a the unique update α' of α to the variables that are decided in $\exists X\beta$. For each $z \in Z_C$ we fix a constant c_z that does not appear in γ and π_i for $i = 1 \ldots n$ such that

$$\forall x \in ((X \cup Y) \cap \mathcal{D}ec(\exists X\beta)) : \ c_z^{\mathfrak{A}} \neq \alpha'(x).$$

By the construction we know that for every rooted path yq which is contained in some π_i and which is both realized and decided, that the proposition

$$\mathfrak{A}, \alpha \models \forall X(\beta \rightarrow \neg ypc_z)$$

holds. Similarly, we fix for every $z \in Z \backslash Z_C$ an arity $F_z = \{f \mid zfy \in \gamma\} \cup \{h\}$, where h is a new feature such that

$$\forall x \in ((X \cup Y) \cap \mathcal{D}ec(\exists X\beta)) : \ \neg\alpha'(x)F_z^{\mathfrak{A}}.$$

Again we know for every rooted path yq which is contained in some π_i and which is both realized and decided, that the proposition

$$\mathfrak{A}, \alpha \models \forall X(\beta \rightarrow \neg ypF_z)$$

is true. It is understood that $c_z \neq c_{z'}$ for $z \neq z'$ and $z, z' \in Z_C$, and $F_z \neq F_{z'}$ for $z \neq z'$ and $z, z' \in Z \backslash Z_C$.

Let ψ be obtained from γ_G be deleting all constraints $atom(z)$ with $z \in Z_C$, and let $z_1, \ldots, z_m$ be an enumeration of Z_C. The formula $\beta' = \exists Y(\gamma'_N \wedge \gamma'_G)$ is now defined by

$$\begin{aligned} \gamma'_N &= \gamma_N \wedge z_1 \doteq c_{z_1} \wedge \ldots \wedge z_m \doteq c_{z_m} \\ \gamma'_G &= \psi[z_1 \leftarrow c_{z_1}, \ldots, z_m \leftarrow c_{z_m}] \wedge \bigwedge_{z \in (Z \backslash Z_C)} zF_z \end{aligned}$$

We will show that β' satisfies the requirements stated above. Clearly, $\beta' \models_{\mathrm{CFT}'} \beta$. Furthermore, $\exists X\beta \models\!\!\models \exists X\beta'$ by proposition 4.9.

The remaining part is to show that $\mathfrak{A}, \alpha \models \forall X(\beta' \rightarrow \bigwedge_{i=1}^{n} \neg\pi_i)$ for all $i = 1..n$. We distinguish the following cases for π_i:

1. *π_i contains a rooted path xp that is undecided and not realized in β*: Let p' be the longest path such that xp' is realized in β (such a path must exists since at least $x\epsilon$ is realized), and let $p = p'fq$. If xp' is determined in β, then $\beta \models xp'F$ with $f \notin F$ as β is saturated. Hence, $\beta \models \neg xp{\downarrow}$.

   If xp' is undetermined, we know that $|xp'|_\gamma$ is a variable z with $z \in Z$ since xp is undecided. As p' is the longest subpath of p' with xp' realized in β we know that $zft \notin \gamma$. If $z \in Z_C$, then we have substituted z by a constant c_z. Otherwise, we have added an arity constraint zF_z with $f \notin F_z$. Hence, $\beta \models \neg xp{\downarrow}$.

2. *every undecided rooted path contained in π_i is realized in β.* Note that in this case π_i cannot be of the form $xp{\downarrow}$, since xp realized implies that $xp{\downarrow} \in [\beta]$. We split up this case as follows:

   (a) $\pi_i = xpc$. Then xp must be undecided as π_i is an X-joker. Since xp is realized in β, we know that $|xp|_\gamma$ is a variable z with $z \in Z$. If $z \in Z_c$, then we have substituted z by a constant symbol c' different from c. Otherwise, either γ contains an arity constraint zF or we have added an arity constraint zF_z in γ'_G. In both cases we get $\beta' \models \neg\pi_i$.

   (b) $\pi_i = xpF$. Analogous to case (2a)

   (c) $\pi_i = atom(xp)$. Then xp is undecided. Since xp is realized in β, we know that $|xp|_\gamma$ is a variable z with $z \in Z$. Now $\pi_i \notin [\beta]$ implies that $atom(z) \notin \gamma_G$ and therefore $z \notin Z_C$. Hence, we have added a constraint zF_z in γ'_G.

   (d) $\pi_i = xp{\downarrow}yq$ *or* $\pi_i = yq{\downarrow}xp$ *where xp is undecided and not determined in β.* By the above cases we can assume that xp is realized in β. Again we get $|xp|_\gamma = z \in Z$. There are two cases, namely $z \in Z_C$ and $z \in (Z \backslash Z_C)$.
   If $z \in Z_C$, then we have substituted z by c_z in γ'_G. If yq is undecided, we know that yq is also realized in β. This implies that we have either added a constraint $z'F_{z'}$ in γ'_G or we have substituted z' by $c_{z'}$ in γ'_G, where $z' = |yq|_\gamma$. Since c_z and $c_{z'}$ are different, this shows $\beta' \models \neg\pi_i$.
   If yq is decided, then $\mathfrak{A}, \alpha \models \forall X(\beta \rightarrow \neg yqc_z)$ by the definition of c_z. As $\beta' \models \beta$, we get $\mathfrak{A}, \alpha \models \forall X(\beta' \rightarrow \neg xp{\downarrow}yq)$.
   The other case $z \in (Z \backslash Z_C)$ is handled analogously.

□

Lemma 4.12 *If β, $\beta_1, \ldots, \beta_n$ are prime formulae, then*

$$\exists X(\beta \wedge \bigwedge_{i=1}^{n} \neg\beta_i) \quad \models_{\mathrm{CFT}'} \quad \bigwedge_{i=1}^{n} \exists X(\beta \wedge \neg\beta_i).$$

Proof. Let $\beta, \beta_1, \ldots, \beta_n$ be prime formulae. Then $\exists X(\beta \wedge \bigwedge_{i=1}^n \neg\beta_i) \models \bigwedge_{i=1}^n \exists X(\beta \wedge \neg\beta_i)$ is trivial. To see the other direction, suppose that $\mathfrak{A}$ is a model of CFT′ and $\mathfrak{A}, \alpha \models \bigwedge_{i=1}^n \exists X(\beta \wedge \neg\beta_i)$. We must exhibit some X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta$ and $\mathfrak{A}, \alpha' \models \neg\beta_i$ for $i = 1, \ldots, n$.

Without loss of generality we can assume that $\mathfrak{A}, \alpha' \models \exists X(\beta \wedge \beta_i)$ for $i = 1, \ldots, m$ and $\mathfrak{A}, \alpha' \models \neg\exists X(\beta \wedge \beta_i)$ for $i = m+1, \ldots, n$. For every $i = 1, \ldots, m$ let λ_i be a projection of β_i.

Since for every $i = 1, \ldots, m$

$$\lambda_i \models\!\!\!\models_{\mathrm{CFT}'} \beta_i,$$

we know that there is a proper path constraint π with

$$\mathfrak{A}, \alpha \models \exists X(\beta \wedge \pi) \quad \text{and} \quad \mathfrak{A}, \alpha \models \exists X(\beta \wedge \neg\pi),$$

This implies by corollary 4.2 that we can calculate, for every $i = 1, \ldots, m$, an X-joker π_i' for β with $\beta \wedge \pi_i \models_{\mathrm{CFT}'} \pi_i'$. By Lemma 4.11 we have

$$\exists X\beta \models \exists X(\beta \wedge \bigwedge_{i=1}^{m} \neg\pi_i').$$

from which

$$\exists X\beta \models \exists X(\beta \wedge \bigwedge_{i=1}^{m} \neg\pi_i).$$

follows.

Since $\neg\pi_i \models \neg\beta_i$ by Proposition 4.4, we have

$$\exists X\beta \models \exists X(\beta \wedge \bigwedge_{i=1}^{m} \neg\beta_i).$$

Hence we know that there exists an X-update α' of α such that $\mathfrak{A}, \alpha' \models \beta$ and $\mathfrak{A}, \alpha' \models \neg\beta_i$ for $i = 1, \ldots, m$. Since we know that $\mathfrak{A}, \alpha \models \neg\exists X(\beta \wedge \beta_i)$ for $i = m+1, \ldots, n$, we have $\mathfrak{A}, \alpha' \models \neg\beta_i$ for $i = m+1, \ldots, n$. □

Lemma 4.13 *For every two prime formulae β, β' and every set of variables X one can compute a Boolean combination δ of prime formulae such that*

$$\exists X(\beta \wedge \neg\beta') \models\!\!\!\models_{\mathrm{CFT}'} \delta \quad \textit{and} \quad \mathcal{V}(\delta) \subseteq \mathcal{V}(\exists X(\beta \wedge \neg\beta'))$$

Proof. Let λ be a projection of β' and $\mathfrak{A}$ be model of CFT′. We distinguish the following cases:

1. *There exists an $\pi \in \lambda$ such that we can derive an X-joker π' with $\beta \wedge \pi \models_{\mathrm{CFT}'} \pi'$ using lemma 4.10.* Then $\exists X\beta \models_{\mathrm{CFT}'} \exists X(\beta \wedge \neg\pi')$ by lemma 4.11. Since $\beta \wedge \neg\pi' \models_{\mathrm{CFT}'} \neg\pi$, we get

   $$\exists X\beta \models_{\mathrm{CFT}'} \exists X(\beta \wedge \neg\pi).$$

   Since $\beta' \models_{\mathrm{CFT}'} \lambda \models \pi$, we know that $\neg\pi \models_{\mathrm{CFT}'} \neg\beta'$ and hence $\exists X\beta \models_{\mathrm{CFT}'} \exists X(\beta \wedge \neg\beta')$. Thus

   $$\exists X(\beta \wedge \neg\beta') \models\!\!\!\models_{\mathrm{CFT}'} \exists X\beta$$

   The rest follows from proposition 4.30.

2. *For every $\pi \in \lambda$ lemma 4.10 does not produce an X-joker π' with $\beta \wedge \pi \models_{\mathrm{CFT}'} \pi'$.* Then for every valuation α into $\mathfrak{A}$ and every $\pi \in \lambda$ either $\mathfrak{A}, \alpha \models \forall X(\beta \to \pi)$ or $\mathfrak{A}, \alpha \models \forall X(\beta \to \neg\pi)$. This implies that either

$$\mathfrak{A}, \alpha \models \forall X(\beta \to \bigwedge_{\pi \in \lambda} \pi)$$

or

$$\mathfrak{A}, \alpha \models \forall X(\beta \to \neg(\bigwedge_{\pi \in \lambda} \pi)).$$

Since $\bigwedge_{\pi \in \lambda} \pi \models\!\models \lambda \models\!\models_{\mathrm{CFT}'} \beta'$, this implies that there is no valuation α with

$$\mathfrak{A}, \alpha \models \exists X(\beta \wedge \beta') \quad \text{and} \quad \mathfrak{A}, \alpha \models \exists X(\beta \wedge \neg\beta').$$

Hence

$$\exists X(\beta \wedge \neg\beta') \models\!\models_{\mathrm{CFT}'} \exists X\beta \wedge \neg\exists X(\beta \wedge \beta').$$

The rest follows from propositions 4.30 and 4.31.

□

Theorem 4.5 *For every formula ϕ one can compute a Boolean combination δ of prime formulae such that $\phi \models\!\models_{\mathrm{CFT}'} \delta$ and $\mathcal{V}(\delta) \subseteq \mathcal{V}(\beta)$.*

Proof. Follows from Lemma 4.1, Propositions 4.31 and 4.30, and Lemmas 4.12 and 4.13. □

Theorem 4.6 CFT′ *is a complete and decidable theory.*

Proof. The completeness of CFT′ follows from the preceding theorem and the fact that $\top$ is the only closed prime formula. The decidability follows from the completeness and the fact that CFT′ is given by a recursive set of sentences. □

Chapter 5

Decidability of the Positive Existential Fragment of RFT

In this chapter, we will show that the positive existential fragment of the language RFT is decidable. This is done by presenting a quasi-terminating rule system that transforms each quantifier-free conjunction of atomic constraints into an equivalent set of solved formulae (seen as a disjunction).

Section 5.1 gives an informal description of the method for checking satisfiability. Section 5.2 introduces the two-sorted logic RF, in which a regular path expression xLy is expressed by two constraints $x[\mu]y$ and $\mu \doteq L$. μ is a variable that is interpreted as a feature path. Furthermore, this section presents a validity preserving translation of RFT-clauses into RF-clauses. Section 5.3 characterise the clauses that are obtained by translating clauses of the original syntax into RF and defines two normal forms, namely pre-solved clauses and solved clauses. Pre-solved clauses are satisfiable under the condition that there exists a valuation for the path variables, whereas solved clauses are always satisfiable. We show the satisfiability of solved clauses by constructing a valuation in the feature tree interpretation. Since we will prove in the following sections that we can transform an initial clause into an equivalent set of solved clauses, this implies that the feature tree interpretation is canonical. In Section 5.4 we define a set of deterministic and non-deterministic simplification rules that transforms an initial clause into an equivalent set of pre-solved clauses. This intermediate step is necessary since otherwise the algorithm would not terminate. Section 5.5 finally shows how pre-solved clauses can be transformed into solved clauses.

5.1 The Method

We will first present a slightly modified method for testing satisfiability of quantifier free formulae in the FT′-language, and then turn to the systems as extended by regular path expressions. For the sake of convenience, we allow expressions of the form xpy where p is a path. Although these constraints extend the language of FT′, they can easily be defined within FT′. Every constraint $xf_1 \ldots f_n y$ is equivalent to the FT′-formula $\exists x_1, \ldots x_n (xf_1x_1 \wedge x_1f_2x_2 \wedge \ldots x_{n-1}f_nx_n \wedge x_n \doteq y)$.

Now consider a clause $\phi = xp_1y_1 \wedge xp_2y_2$ (in the rest of the chapter we will call formulae like this clauses). Although only subtree relations for x, y_1 and x, y_2 are contained in this clause, an additional subtree or equality relation can be implied depending on the paths p_1 and p_2. If p_1 equals p_2, we know that y_1 and y_2 must be equal, which implies that ϕ is equivalent to $xp_1y_1 \wedge y_1 \doteq y_2$. If p_1 is a prefix of p_2 and hence $p_2 = p_1p'$, we can transform ϕ into the equivalent formula $xp_1y_1 \wedge y_1p'y_2$, thus additionally stating that y_2 is a subtree of y_1. The reverse case is handled similarly. If neither prefix nor equality holds between the paths, there is nothing to do. By and large, clauses where the last condition holds for every x and every pair of different constraints $xp_1y_1 \in \phi$ and $xp_2y_2 \in \phi$ are the solved graphs as defined in the last chapter, which are satisfiable.

If we consider a clause of the form $\phi = xL_1y_1 \wedge xL_2y_2$, then we have again to check the relation between y_1 and y_2. But now there is in general no unique relation determined by ϕ, since this depends on which paths p_1 and p_2 are used out of L_1 and L_2. Hence, we have to select non-deterministically a relation between p_1 and p_2 before we can calculate the relation between y_1 and y_2. In the following, we will often just say "guess" instead of "select non-deterministically".

But there is a problem with the original syntax, namely that it does not allow one to express any relation between the chosen paths[1]. Therefore, we extend the syntax by introducing so-called path variables (written $\mu, \nu, \mu', \ldots$), which are interpreted as feature paths. We will henceforth refer to the variables of the original syntax as tree variables. If we use in addition the modified subtree relation $x[\mu]y$ and a restriction constraint $\mu \,\dot{\in}\, L$, a path expression xLy can be expressed by the equivalent clause $x[\mu]y \wedge \mu \,\dot{\in}\, L$ (μ new).

[1] Maxwell and Kaplan solved this problem by using operations on regular languages such as intersection and calculating prefix languages directly. The use of this method forced them to introduce a new variable each time a transformation rule was applied. For a feature description that contains a cycle of the form $xL_1y_1 \wedge \ldots y_{n-1}L_nx$ this resulted in the introduction of an infinite number of variables.

Using this extended (two-sorted) syntax we are now able to reason about the relations between different path variables. To do this we introduce additional constraints $\mu \doteq \nu$ (equality), $\mu \dot{\prec} \nu$ (prefix) and $\mu \dot{\amalg} \nu$ (divergence). Divergence holds if neither equality nor prefix does. Now we can describe a normal form equivalent to the solved graphs of the last chapter, which we will call pre-solved clauses. A clause ϕ is *pre-solved* if for each pair of different constraints $x[\mu]y_1$ and $x[\nu]y_2$ in ϕ there is a constraint $\mu \dot{\amalg} \nu$ in ϕ. Additionally, we require pre-solved clauses to contain at most one constraint $\mu \dot{\in} L$ for each path variable μ. We call these clauses pre-solved, since these clauses are not necessarily satisfiable: it may happen that the divergence constraints together with the restrictions of the form $\mu \dot{\in} L$ are inconsistent (think of the clause $\mu \dot{\in} f^+ \wedge \nu \dot{\in} ff^+ \wedge \mu \dot{\amalg} \nu$, e.g.). But pre-solved clauses have the property that if we find a valuation for the path variables, then the clause is satisfiable.

Our algorithm first transforms a clause into a set of pre-solved clauses, which is (when viewed as a disjunction) equivalent to the initial clause. In a second phase the pre-solved clauses are checked for satisfiability with respect to the path variables. In both phases we use a set of deterministic and non-deterministic transformation rules.

Before starting with the technical part we will illustrate the first phase, since it is the more difficult one. For the rest of the chapter we will write clauses as sets of atomic constraints. Consider the clause $\gamma = \{x[\mu]y,\ \mu \dot{\in} L_1,\ x[\nu]z,\ \nu \dot{\in} L_2\}$. Initially, one guesses the relation between the path variables μ and ν. In our example there are four different possibilities. Therefore, γ can be expressed equivalently by the set of clauses

$$\begin{aligned}
\gamma_1 &= \{\mu \dot{\amalg} \nu,\ x[\mu]y,\ \mu \dot{\in} L_1,\ x[\nu]z,\ \nu \dot{\in} L_2\} \\
\gamma_2 &= \{\mu \doteq \nu,\ x[\mu]y,\ \mu \dot{\in} L_1,\ x[\nu]z,\ \nu \dot{\in} L_2\} \\
\gamma_3 &= \{\mu \dot{\prec} \nu,\ x[\mu]y,\ \mu \dot{\in} L_1,\ x[\nu]z,\ \nu \dot{\in} L_2\} \\
\gamma_4 &= \{\nu \dot{\prec} \mu,\ x[\mu]y,\ \mu \dot{\in} L_1,\ x[\nu]z,\ \nu \dot{\in} L_2\}.
\end{aligned}$$

The clause γ_1 is pre-solved. For the others we must evaluate the relation between μ and ν as follows. In γ_2 we substitute μ for ν and y for z, which yields

$$\{y \doteq z,\ x[\mu]y,\ \mu \dot{\in} L_1,\ \mu \dot{\in} L_2\}.$$

We keep only the equality constraint for the tree variables since we are interested only in their valuation. Combining $\{\mu \dot{\in} L_1, \mu \dot{\in} L_2\}$ into $\{\mu \dot{\in} (L_1 \cap L_2)\}$ will then give us an equivalent pre-solved clause. For γ_3 we know that the variable ν can be split up into two parts, one of them covered by μ. We can use concatenation of path variables to express this, that means we can replace ν by the term $\mu \circ \nu'$ with ν' new. This would lead to the clause

$$\{\mu \dot{\prec} \mu\circ\nu',\ x[\mu]y,\ \mu \dot{\in} L_1,\ x[\mu\circ\nu']z,\ \mu\circ\nu' \dot{\in} L_2\}.$$

But this could easily be expressed more simply. First, the constraint $\mu \dot{\prec} \mu\circ\nu'$ is superfluous. Second, the constraint $x[\mu\circ\nu']z$ in combination with $x[\mu]y$ can also be expressed by $\{x[\mu]y,\ y[\nu']z\}$. We now obtain the clause

$$\gamma_3' = \{x[\mu]y,\ \mu \,\dot{\in}\, L_1,\ y[\nu']z,\ \mu\circ\nu' \,\dot{\in}\, L_2\}.$$

This shows that we do not need concatenation of path variables within subtree agreements, and we will avoid them for simplicity.

The only thing that remains in order to achieve a pre-solved clause is to resolve the constraint $\mu\circ\nu' \,\dot{\in}\, L_2$. To do this we have to guess a *decomposition* P, S of L_2 with $P\circ S = \{ps \mid p \in P, s \in S\} \subseteq L_2$ such that $\mu \,\dot{\in}\, P$ and $\nu' \,\dot{\in}\, S$ holds. In general, there can be an infinite number of decompositions (think of the possible decompositions of the language f^*g). But as we use regular languages, there is a finite set of regular decompositions which covers all possibilities. Finally, reducing $\{\mu \,\dot{\in}\, L_1,\ \mu \,\dot{\in}\, P\}$ to $\{\mu \,\dot{\in}\, (L_1 \cap P)\}$ will yield a pre-solved clause.

Note that the evaluation of the prefix relation in γ_3 has the additional effect of introducing a new constraint $y[\nu']z$. In general this implies that after the evaluation of prefix constraints there may be some path variables whose relation is unknown. Hence, after reducing the terms of form $\mu \doteq \nu$ or $\mu \dot{\prec} \nu$, we may have to repeat the non-deterministic choice of relation between path variables. In the end, the only remaining constraints between path variables are of form $\mu \,\ddot{\amalg}\, \nu$.

Now let's turn to an additional point we have to consider, namely that the rules we present will (naturally) loop in some cases. Roughly speaking, one can say that this occurs if a cycle in the graph coincides with a cycle in the regular language. To see this let us vary the above example and let γ be the clause

$$\{x[\mu]x,\ \mu \,\dot{\in}\, f,\ x[\nu]z,\ \nu \,\dot{\in}\, f^*g\}$$

Then a possibly looping derivation could be

$\{\mu \dot{\prec} \nu,\ x[\mu]x,\ \mu \,\dot{\in}\, f,\ x[\nu]z,\ \nu \,\dot{\in}\, f^*g\}$	adding relation $\mu \dot{\prec} \nu$
$\{x[\mu]x,\ \mu \,\dot{\in}\, f,\ x[\nu']z,\ \mu\circ\nu' \,\dot{\in}\, f^*g\}$	splitting ν into $\mu\circ\nu'$
$\{x[\mu]x,\ \mu \,\dot{\in}\, f,\ x[\nu']z,\ \mu \,\dot{\in}\, f^*,\ \nu' \,\dot{\in}\, f^*g\}$	decomposing $\mu\circ\nu' \,\dot{\in}\, f^*g$
$\{x[\mu]x,\ \mu \,\dot{\in}\, f,\ x[\nu']z,\ \nu' \,\dot{\in}\, f^*g\}$	joining μ-restrictions

But we will prove that we get a quasi-terminating rule system, which means that the rule system may cycle, but produces only finitely many different clauses (see [Der87]). This is achieved by the following measures: first, we will guarantee that the rules do

not introduce additional variables; second, we restrict concatenation to length 2; and third, we will show that the rule system produces only finitely many regular languages. In order to show that our rewrite system is complete, we must additionally show that every solution can be found in a pre-solved clause.

Finally, we want to mention a related work in the context of terminological logics, which have their roots in the knowledge representation formalism KL-ONE (see [BS85]; for a comparison of feature logic and terminological logics see [NS91]). Baader [Baa90, Baa91] has considered an extension of the terminological language $\mathcal{ALC}$ of [SSS91], which uses regular languages of roles (binary relations) instead of single roles to build up concept terms. He has shown that testing satisfiability of this extension is decidable. In contrast with our problem, there are cycles in the regular languages but no cycles in the formulae, since these cycles (i.e., cyclic concept definitions) can be compiled out using a technique called "internalisation". As we have pointed out in the example on page 112, the combination of both types of cycle makes our problem hard. Thus, while Baader can split the problem into independent subproblems, we cannot use a similar technique because of the structure of our problem (to be more concrete, the technique of "internalisation" has been used in Baader et al. [BBN+93] to prove undecidability of functional uncertainty with negation).

5.2 The language RF

Before defining the language RF, we first introduce some relation on paths. We say that a path u is a **prefix** of a path v (written $u \prec v$) if there is a non-empty path w such that $v = uw$. Note that $\prec$ is neither symmetric nor reflexive. We say that two paths u, v **diverge** (written $u \amalg v$) if there are features f, g with $f \neq g$, and possibly empty paths w, w_1, w_2, such that $u = wfw_1 \wedge v = wgw_2$. It is clear that $\amalg$ is a symmetric relation.

Proposition 5.1 *Given two paths u and v, then exactly one of the relations $u = v$, $u \prec v$, $u \succ v$ or $u \amalg v$ holds.*

The language RF has two sorts, *tree* and *path*, and an infinite supply of variables of both sorts. The set of tree variables is denoted by $\mathcal{X}$, and the set of path variables is denoted by $\mathcal{P}$. We use the letters $x, y, \ldots$ for tree variables and $\mu, \nu, \ldots$ for path variables. Besides the equality symbol $\doteq$, the signature of RF consists of

- an infinite set $\mathcal{L}$ of constant symbols of the sort *tree*,

- a ternary relation symbol $\cdot[\cdot]\cdot$ of sort $\mathit{tree} \times \mathit{path} \times \mathit{tree}$,
- a binary function $\circ$ of sort $\mathit{path} \times \mathit{path} \to \mathit{path}$ called **concatenation**,
- a binary relation $\dot{\prec}$ of sort $\mathit{path} \times \mathit{path}$,
- a binary relation $\dot{\amalg}$ of sort $\mathit{path} \times \mathit{path}$, and
- an infinite set of unary predicate symbols of sort path of the form $\doteq L$, where L is a regular expression with $[\![L]\!] \subseteq \mathcal{L}^+$.

We use mixfix notation $t[p]t'$ for the so-called **subtree constraints**, infix notation $p \dot{\prec} q$ and $p \dot{\amalg} q$ for the so-called **prefix and divergence constraints**, and postfix notation $p \doteq L$ for the so-called **path restriction constraints**. In RF, a tree term is either a constant symbol (denoted by $c, c', \ldots$), or a tree variable. Tree terms will be denoted by the letters $t, t', \ldots$. A path term (denoted by $p, q, \ldots$) is either a path variable or the concatenation of to path terms $p \circ q$. Given a RF-formula ϕ, we use $\mathcal{V}_{\mathcal{X}}(\phi)$ to denote the set of tree variables in ϕ, and $\mathcal{V}_{\mathcal{P}}(\phi)$ to denote the set of path variables ϕ.

In the following, we consider only those interpretations of RF which have $\mathcal{L}^+$ as the domain for the sort *path*, and which interpret $\circ$, $\dot{\prec}$ and $\dot{\amalg}$ as concatenation, prefix and divergence, respectively. Hence, we assume for simplicity that an RF-interpretation $\mathfrak{A}$ just consists of a domain $\mathbf{U}_{\mathit{tree}}(\mathfrak{A})$ for the sort *tree*. Clearly, an RF-interpretation is uniquely determined by its interpretation of the constants and the predicate symbol $\cdot[\cdot]\cdot$. A **valuation** into some RF-interpretation $\mathfrak{A}$ is a pair $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$, where $\alpha_{\mathcal{X}}$ is a valuation of the tree terms into $\mathbf{U}_{\mathit{tree}}(\mathfrak{A})$ and $\alpha_{\mathcal{P}}$ is a function $\alpha_{\mathcal{P}} : \mathcal{P} \to \mathcal{L}^+$. In the following, we use $(\alpha_{\mathcal{P}}, \alpha_{\mathcal{X}}) \models_{\mathfrak{A}} \phi$ instead $\mathfrak{A}, (\alpha_{\mathcal{P}}, \alpha_{\mathcal{X}}) \models \phi$ for better readability.

The feature tree structure $\mathfrak{T}_{\mathrm{RF}}$ is the following RF-structure:

- the universe $\mathbf{U}_{\mathit{tree}}(\mathfrak{T}_{\mathrm{RF}})$ of the sort *tree* is the set of all feature trees,
- $c^{\mathfrak{T}_{\mathrm{RF}}} = (\{\epsilon\}, \{(\epsilon, c)\})$ for every constant symbol $c \in \mathcal{L}$,
- $(\sigma, p, \tau) \in \cdot[\cdot]\cdot^{\mathfrak{T}_{\mathrm{RF}}}$ iff τ is the subtree of σ at the path p.

$\mathfrak{R}_{\mathrm{RF}}$ is the substructure of $\mathfrak{T}_{\mathrm{RF}}$ consisting only of the rational feature trees.

As we have argued previously, the language RF is more appropriate for testing satisfiability of formulae in the positive existential fragment of RFT. Hence, we have to translate the Σ_1^+-fragment of RFT into RF such that validity is preserved. Now we could restrict ourself to the two RFT-interpretations $\mathfrak{T}_{\mathrm{RFT}}$ and $\mathfrak{R}_{\mathrm{RFT}}$. But we want

to solve the more general problem as was stated by [KM88] and [BBN+93], namely whether satisfiability of conjunction of "functional uncertainty" constraints is decidable. Roughly speaking, these authors considered a language which is similar to our language RFT. Furthermore, they considered only those RFT-interpretations which satisfy the following system RFT of axioms:

(Uniq)	$\neg(c_1 \doteq c_2)$	if c_1 and c_2 are different constants
(FCCl)	$\tilde{\forall}(cfx \to \bot)$	for all constants c and features f
(Fea)	$\tilde{\forall}(xfy \wedge xfz \to y \doteq z)$	for every feature f
(Conc)	$\tilde{\forall}(xfpy \leftrightarrow \exists z(xfz \wedge zpy)$	for all features f and non-empty paths p.
(Reg)	$xLy \leftrightarrow \bigvee_{p \in L} xpy$	for every regular expression L.

Clearly, this is not a first-order axiomatisation since it uses a possibly infinite disjunction in the axiom scheme (Reg).

Proposition 5.2 *Both* $\mathfrak{T}_{\mathrm{RFT}}$ *and* $\mathfrak{R}_{\mathrm{RFT}}$ *are models of* RFT.

Now we define a translation of RFT-formulae into RF-formulae, and a corresponding translation of RFT-models into RF-interpretations. Given a RFT-formula ϕ, we define $\overline{\phi}$ to be the RF-formula where every occurrence xLt in ϕ is replace by $x[\mu]t \wedge \mu \dot{\in} L$, where μ is a new path variable. Given a RFT-model $\mathfrak{A}$, we define the associated RF-interpretation $\overline{\mathfrak{A}}$ by

$$\begin{aligned} \mathbf{U}_{tree}(\overline{\mathfrak{A}}) &:= \mathbf{U}(\mathfrak{A}) \\ c^{\overline{\mathfrak{A}}} &:= c^{\mathfrak{A}} \quad \text{for every } c \in \mathcal{L} \\ \cdot[\cdot]\cdot^{\overline{\mathfrak{A}}} &:= \{(\sigma, p, \tau) \mid (\sigma, \tau) \in \{p\}^{\mathfrak{A}}\} \end{aligned}$$

Using this translation of RFT-models, we get the following system RF of axioms which is equivalent to the above stated RFT-axioms:

(Uniq)	$\neg(c_1 \doteq c_2)$	if c_1 and c_2 are different constants
(FCCl')	$\tilde{\forall}(c[\mu]x \to \bot)$	for all constants c
(Fea')	$\tilde{\forall}(x[\mu]y \wedge x[\mu]z \to y \doteq z)$	
(Conc')	$\tilde{\forall}(x[\mu \circ \mu']y \leftrightarrow \exists z(x[\mu]z \wedge z[\mu']y)$	

Note that we do not have to translate the last RFT-axiom scheme (Reg), since we have fixed the interpretation of the sort *path* in RF.

Proposition 5.3 *Both* $\mathfrak{T}_{\mathrm{RF}}$ *and* $\mathfrak{R}_{\mathrm{RF}}$ *are models of* RF. *Furthermore,* $\overline{\mathfrak{T}_{\mathrm{RFT}}} = \mathfrak{T}_{\mathrm{RF}}$ *and* $\overline{\mathfrak{R}_{\mathrm{RFT}}} = \mathfrak{R}_{\mathrm{RF}}$.

Proposition 5.4 $\overline{\mathfrak{A}}$ *is an* RF*-model for every* RFT*-model* $\mathfrak{A}$. *Conversely, there exists for every* RF*-model* $\mathfrak{B}$ *an* RFT*-model* $\mathfrak{A}$ *such that* $\mathfrak{B} = \overline{\mathfrak{A}}$.

Proposition 5.5 *For every* RFT*-model* $\mathfrak{A}$, *every valuation* $\alpha_{\mathcal{X}}$ *and every* RFT*-clause* ϕ,

$$\mathfrak{A}, \alpha_{\mathcal{X}} \models \phi \iff \textit{exists } \alpha_{\mathcal{P}}\colon\ \overline{\mathfrak{A}}, (\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models \overline{\phi}.$$

5.3 Prime, Pre-Solved and Solved Clauses

In this section, we will define the input and output clauses for both phases of the algorithm. A **clause** is either the special symbol $\bot$ ("false") or a finite set of atomic constraints denoting their conjunction. We will say that a path term $\mu \circ \nu$ is **contained** (or **used**) **in** some clause ϕ if ϕ contains either a constraint $\mu \circ \nu \mathrel{\dot\in} L$ or a constraint $\mu \circ \nu \mathrel{\dot{\amalg}} q$.[2] Constraints of the form $p \mathrel{\dot\in} L$, $p \mathrel{\dot{\amalg}} q$, $\mu \mathrel{\dot\prec} \nu$ and $\mu \doteq \nu$ will be called **path term constraints**. Note that the validity of path term constraints depend only on the valuation of the path variables. Hence, we write $\alpha_{\mathcal{P}} \models \phi$ if ϕ is a set of path term constraints that are valid under $\alpha_{\mathcal{P}}$.

Let ϕ be some clause and x, y be distinct variables. We say that ϕ **binds** y **to** x **(resp. c)** if $x \doteq y \in \phi$ (resp. $c \doteq y \in \phi$) and y occurs only once in ϕ. Here it is important that we consider equations as directed, that is, we assume that $x \doteq y$ is different from $y \doteq x$. Note that we diverge from the standard practise in treating equality as binding its right argument (as defined in Section 4.3, page 68). This is for uniformity with constraints involving the $\mathrel{\dot\prec}$ relation, since they will always share a left argument which we wish to avoid renaming.[3] We say that ϕ **eliminates** y if ϕ binds y to some variable x. A clause is called **basic** if it is either $\bot$ or:

1. all path terms in ϕ are either path variables or the concatenation of two path variables (i.e., the length of concatenation is restricted to 2),
2. concatenation is not used in prefix or equality constraints in ϕ (i.e., ϕ does not contain a constraint of the form $\mu \circ \mu' \doteq \nu$, $\mu \circ \mu' \doteq \nu \circ \nu'$ etc.),

[2] We will not distinguish between $p \mathrel{\dot{\amalg}} q$ and $q \mathrel{\dot{\amalg}} p$.

[3] We find the commuted notation with $\mathrel{\dot\succ}$ in place of the prefix relation $\mathrel{\dot\prec}$ even less natural.

3. ϕ does not contain a constraint of the form $t \doteq c$,
4. an equation $t \doteq x$ appears in ϕ if and only if ϕ eliminates x, and
5. for every path variable μ used in ϕ there is *at most* one constraint $t[\mu]t' \in \phi$.

A clause ϕ is called **prime** if it is basic, does not contain a path term of the form $\mu \circ \nu$ and does not contain an atomic constraint of form $p \mathrel{\dot{\amalg}} q$, $\mu \mathrel{\dot{\prec}} \nu$ or $\mu \doteq \nu$.

The following two clauses are not basic. The first clause does not satisfy condition 4, and the second clause does not satisfy condition 5:

$$\{y \doteq z,\ \ x[\mu]y,\ x[\mu']z,\ \mu \mathrel{\dot{\in}} f,\ \mu' \mathrel{\dot{\in}} (f \cup g)\}$$
$$\{\mu \mathrel{\dot{\amalg}} \mu',\ x[\mu]y,\ y[\mu]z,\ x[\mu']z,\ \mu \mathrel{\dot{\in}} f,\ \mu' \mathrel{\dot{\in}} (f \cup g)\}$$

On the other hand, the clause

$$\{y \doteq z,\ \mu \mathrel{\dot{\amalg}} \mu',\ x[\mu]y,\ x[\mu']y',\ \mu \mathrel{\dot{\in}} f,\ \mu' \mathrel{\dot{\in}} (f \cup g)\}$$

is an example of a basic clause which is not prime. The prime clauses are the input clauses of our algorithm.

Proposition 5.6 *For every* RFT*-clause ϕ there is a prime clause ψ such that ψ is equivalent to $\overline{\phi}$. Conversely, for every prime clause ϕ there is a* RFT*-clause ψ such that ϕ is equivalent to $\overline{\psi}$.*

Taking as an example the RFT-formula $s\ \mathit{topic}\ x\ \wedge\ s\ \mathit{comp}^*\ \mathit{obj}\ x$, the equivalent prime clause

$$s[\mu_1]x \wedge s[\mu_2]x \wedge \mu_1 \mathrel{\dot{\in}} \mathit{topic} \wedge \mu_2 \mathrel{\dot{\in}} \mathit{comp}^*\ \mathit{obj}.$$

Now we turn to the output clauses of the first phase. A basic clause is said to be **pre-solved** if it is either $\bot$ or the following holds:

1. ϕ contains no atomic constraint of the form $c[\mu]t$,
2. $\mu \mathrel{\dot{\in}} L \in \phi$ and $\mu \mathrel{\dot{\in}} L' \in \phi$ implies $L = L'$,
3. $\mu \mathrel{\dot{\in}} \emptyset$ is not in ϕ,
4. ϕ contains no term of form $\mu \circ \nu$,
5. ϕ contains no constraint of form $\mu \doteq \nu$ or $\mu \mathrel{\dot{\prec}} \nu$, and

6. $\mu \dot{\amalg} \nu \in \phi$ if and only if $\mu \neq \nu$, $x[\mu]t \in \phi$ and $x[\nu]t' \in \phi$.

For example the clause

$$\left\{ \begin{array}{c} \mu_1 \dot{\amalg} \mu_2,\ \mu_1 \dot{\amalg} \mu_3,\ \mu_2 \dot{\amalg} \mu_3,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

is pre-solved. The following clauses are not pre-solved. The first violates condition 4, the second violates condition 5, and the third and fourth do not satisfy condition 6:

$$\{\mu_1 \dot{\amalg} \mu_2, x[\mu_1]y_1,\ x[\mu_2]y_2,\ \mu_1 \dot{\in} (f \cup g),\ \mu_1 \circ \mu_2 \dot{\in} (f^+ \cup h^+)\}$$

$$\{\mu_1 \dot{\prec} \mu_2,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+)\}$$

$$\{x[\mu_1]y_1,\ x[\mu_2]y_2,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+)\}$$

$$\{\mu_1 \dot{\amalg} \mu_2,\ x[\mu_1]y_1,\ y_1[\mu_2]y_2,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+)\}$$

Lemma 5.1 *Let ϕ be a pre-solved clause different from $\bot$. Then ϕ is satisfiable in $\mathfrak{T}_{\mathrm{RF}}$ and $\mathfrak{R}_{\mathrm{RF}}$ if there is a path valuation $\alpha_{\mathcal{P}}$ with $\alpha_{\mathcal{P}} \models \phi_p$, where ϕ_p is the set of path term constraints in ϕ.*

Proof. Let ϕ be a pre-solved clause, and let $\alpha_{\mathcal{P}}$ be a path valuation such that $\alpha_{\mathcal{P}} \models \phi$. Let ψ be the following RFT-clause:

$$\{x\alpha_{\mathcal{P}}(\mu)t \mid x[\mu]t \in \phi\}.$$

It is easy to check that for every tree valuation $\alpha_{\mathcal{X}}$, $\mathfrak{T}_{\mathrm{RFT}}, \alpha_{\mathcal{X}} \models \psi$ if and only if $\mathfrak{T}_{\mathrm{RF}}, (\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models \phi$. The same holds for $\mathfrak{R}_{\mathrm{RF}}$ and $\mathfrak{R}_{\mathrm{RFT}}$.

That ψ is satisfiable in $\mathfrak{T}_{\mathrm{RFT}}$ and $\mathfrak{R}_{\mathrm{RFT}}$ can easily be shown using a similar technique as applied in Theorem 4.1, where we have shown that the solved FT$'$-clauses are satisfiable in $\mathfrak{T}_{\mathrm{FT}'}$ and $\mathfrak{R}_{\mathrm{FT}'}$. □

Since in the first phase we transform each prime clause into an equivalent set of pre-solved clauses, this implies that the structure $\mathfrak{T}_{\mathrm{RF}}$ (resp. $\mathfrak{R}_{\mathrm{RF}}$) is **canonical** for prime clauses; i.e., a prime clause is satisfiable if it is satisfiable in $\mathfrak{T}_{\mathrm{RF}}$ (resp. $\mathfrak{R}_{\mathrm{RF}}$).

In the second phase we will check satisfiability of a pre-solved clause by transforming it into an equivalent set of solved clauses. A clause ϕ is called **solved** if it is either $\bot$ or

1. ϕ contains no atomic constraint of the form $c[\mu]t$,
2. $\mu \,\dot{\in}\, L \in \phi$ and $\mu \,\dot{\in}\, L' \in \phi$ implies $L = L'$,
3. $\mu \,\dot{\in}\, \emptyset$ is not in ϕ,
4. ϕ contains no term of form $\mu \circ \nu$,
5. ϕ contains no constraint of form $\mu \doteq \nu$, $\mu \,\dot{\prec}\, \nu$ or $\mu \,\dot{\amalg}\, \nu$, and
6. for every pair of variables μ, ν such that $\mu \neq \nu$, $x[\mu]t \in \phi$ and $x[\nu]t' \in \phi$, we have $\phi \models \mu \,\dot{\amalg}\, \nu$.

Here $\phi \models \gamma$ means that for every $\mathfrak{A}$ and every $(\alpha_\mathcal{X}, \alpha_\mathcal{P})$ in $\mathfrak{A}$, $(\alpha_\mathcal{X}, \alpha_\mathcal{P}) \models_\mathfrak{A} \phi$ implies $(\alpha_\mathcal{X}, \alpha_\mathcal{P}) \models_\mathfrak{A} \gamma$. Note that the definitions of pre-solvedness and solvedness differ in the last two conditions and that every solved clause is also a prime clause. The clause

$$\{x[\mu]y,\ x[\mu']z, \mu \,\dot{\in}\, (f \cup g),\ \mu' \,\dot{\in}\, h\}$$

is a solved clause, whereas

$$\phi = \{x[\mu]y,\ x[\mu']z, \mu \,\dot{\in}\, (f \cup g),\ \mu' \,\dot{\in}\, gh\}$$

is not solved since $\phi \not\models \mu \,\dot{\amalg}\, \mu'$.

Lemma 5.2 *Every solved clause different from $\bot$ is satisfiable in $\mathfrak{T}_{\mathrm{RF}}$ and $\mathfrak{R}_{\mathrm{RF}}$*

Proof. For every solved clause ϕ there is a $\mathcal{X} \cup \mathcal{P}$-equivalent clause γ such that γ is pre-solved, namely

$$\gamma = \{\mu \,\dot{\amalg}\, \nu \mid \mu \neq \nu \wedge x[\mu]y \in \phi \wedge x[\nu]z \in \phi\} \cup \phi.$$

Thus, a solved clause ϕ is (by lemma 5.1) satisfiable in $\mathfrak{T}_{\mathrm{RF}}$ and $\mathfrak{R}_{\mathrm{RF}}$ if there is a path valuation $\alpha_\mathcal{P}$ with $\alpha_\mathcal{P} \models \phi$. Now conditions 2–5 in the definition of solvedness guarantee that ϕ contains only path term constraints of the form $\mu \,\dot{\in}\, L_\mu$ with $L_\mu \neq \emptyset$ (but no path term constraints of the form $\mu \circ \mu' \,\dot{\in}\, L$, $\mu \doteq \mu'$, $\mu \,\dot{\prec}\, \mu'$ or $p \,\dot{\amalg}\, q$). Hence, every path valuation $\alpha_\mathcal{P}$ with $\alpha_\mathcal{P}(\mu) \in L_\mu$ for every $\mu \in \mathcal{V}_\mathcal{P}(\phi)$ satisfies $\alpha_\mathcal{P} \models \phi$. Since $L_\mu \neq \emptyset$ for every $\mu \in \mathcal{V}_\mathcal{P}(\phi)$, we know that there is at least one path valuation $\alpha_\mathcal{P}$ with $\alpha_\mathcal{P} \models \phi$. □

5.4 The First Phase

5.4.1 A Set of Rules

For checking satisfiability of prime clauses we will use a set of deterministic and non-deterministic transformation rules. Which set of rules is used will depend on the initial clause.

The first rule is the non-deterministic addition of relational constraints between path variables. In one step we will add the relations between one fixed variable μ and all other path variables ν which are used under the same node x as μ. We will consider only the constraints $\mu \doteq \nu$, $\mu \ddot{\sqsubset} \nu$ and $\mu \dot{\prec} \nu$ but not $\mu \dot{\succ} \nu$. Thus the rule can be described by the following pseudo code:

Choose $x \in \mathcal{V}_{\mathcal{X}}(\phi)$ (don't care)
  Choose $x[\mu]t \in \phi$ (don't know)
    For each $x[\nu]t' \in \phi$ **with** μ different from ν and $\mu \ddot{\sqsubset} \nu \notin \phi$
    add $\mu \dot{\diamond}_\nu \nu$ with $\dot{\diamond}_\nu \in \{\doteq, \dot{\prec}, \ddot{\sqsubset}\}$ (don't know)

Formally, this rule is written as

$$\text{(PathRel)} \quad \frac{\{x[\mu]t\} \cup \psi}{\{\mu \dot{\diamond}_\nu \nu \mid x[\nu]t' \in \psi \wedge \mu \neq \nu \wedge \mu \ddot{\sqsubset} \nu \notin \psi\} \cup \{x[\mu]t\} \cup \psi}$$

where $\dot{\diamond}_\nu \in \{\doteq, \dot{\prec}, \ddot{\sqsubset}\}$.

This rule will only be applied if

- ψ contains no prefix and path equality constraint,
- ψ contains no path concatenation, and
- the rule adds at least one constraint.

Although we have restricted the relations $\dot{\diamond}_\nu$ to $\{\doteq, \dot{\prec}, \ddot{\sqsubset}\}$, this rule is globally preserving since we have non-deterministically chosen $x[\mu]y$. To see this let ϕ be a clause, $\mathfrak{A}$ be an interpretation and $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$ be a valuation in $\mathfrak{A}$ with $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathfrak{A}} \phi$. To find an instance of (PathRel) such that $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathfrak{A}} \gamma$ where γ is the result of applying this instance, we choose $x[\mu]y \in \phi$ with $\alpha_{\mathcal{P}}(\mu)$ $\prec$-minimal in

$$\{\alpha_{\mathcal{P}}(\nu) \mid x[\nu]z \in \phi\}.$$

Then for each $x[\nu]z \in \phi$ with $\mu \neq \nu$ and $\mu \,\dot{\amalg}\, \nu \notin \phi$ we add $\mu \,\dot{\diamond}_\nu\, \nu$ where $\alpha_{\mathcal{P}}(\mu) \diamond_\nu \alpha_{\mathcal{P}}(\nu)$ holds. Note that $\dot{\diamond}_\nu$ equals $\dot{\succ}$ will not occur since we have chosen a path variable μ the interpretation of which is $\prec$-minimal. Therefore, the restriction $\dot{\diamond}_\nu \in \{\dot{=}, \dot{\prec}, \dot{\amalg}\}$ is satisfied.

The definition of (PathRel) is more complex than the naive one in the introduction. The reason for this is that only by using this special definition can we maintain the condition that concatenation of path variables is restricted to binary concatenation. To see this suppose that we had added both $\nu_1 \,\dot{\prec}\, \mu$ and $\mu \,\dot{\prec}\, \nu_2$ to a clause γ. Then first splitting up the variable ν_2 into $\mu \circ \nu_2'$ and then μ into $\nu_1 \circ \mu'$ will result in a substitution of ν_2 in γ by $\nu_1 \circ \mu' \circ \nu_2'$. By the definition of (PathRel) we have ensured that this does not happen.

The second non-deterministic rule is used in the decomposition of regular languages. For decomposition we have the following rules:

(DecClash) $$\frac{\{\mu \circ \nu \,\dot{\in}\, L\} \cup \psi}{\bot} \qquad \text{if } \{w \in L \mid |w| > 1\} = \emptyset$$

(LangDec$_\Lambda$) $$\frac{\{\mu \circ \nu \,\dot{\in}\, L\} \cup \psi}{\{\mu \,\dot{\in}\, P\} \cup \{\nu \,\dot{\in}\, S\} \cup \psi} \qquad P \circ S \subseteq L$$

where $L, P, S \subseteq F^+$, $L, P, S \in \Lambda$, and L contains a path w with $|w| > 1$.

For a specific instance of the rule family LangDec$_\Lambda$, Λ must be a *finite* set of regular languages. The clash rule is needed since we require that regular languages do not contain the empty path.

We use Λ in (LangDec$_\Lambda$) as a global restriction, which means that for every Λ we get a different rule (LangDec$_\Lambda$) (and hence a different rule system $\mathcal{R}_\Lambda$). This is done as the rule system is quasi-terminating. By restricting (LangDec$_\Lambda$) we can guarantee that only finitely many regular languages are produced.

For (LangDec$_\Lambda$) to be globally preserving we need to find, for every possible valuation of μ and ν, a suitable pair P, S in Λ. Therefore, we require Λ to satisfy

$$\forall L \in \Lambda,\ \forall w_1, w_2 \neq \epsilon : \\ [w_1 w_2 \in L \Rightarrow \exists P, S \in \Lambda : (P \circ S \subseteq L \wedge w_1 \in P \wedge w_2 \in S)].$$

We will call Λ **closed under decomposition** if it satisfies this condition. Additionally, we have to ensure that $L \in \Lambda$ for every L that is contained in some clause ϕ. We will call such a set Λ **ϕ-closed**.

$$\text{(Eq1)}\quad \frac{\{\mu \doteq \nu,\ x[\mu]t,\ x[\nu]y\} \cup \psi}{\{t \doteq y,\ x[\mu]t\} \cup \psi[\nu \leftarrow \mu, y \leftarrow t]} \qquad \text{(Eq2)}\quad \frac{\{\mu \doteq \nu,\ x[\mu]c,\ x[\nu]c\} \cup \psi}{\{x[\mu]c\} \cup \psi[\nu \leftarrow \mu]}$$

$$\text{(EqClash)}\ \frac{\{\mu \doteq \nu,\ x[\mu]c,\ x[\nu]c'\} \cup \psi}{\bot}\quad c \neq c' \qquad \text{(CClash)}\quad \frac{\{c[\mu]t\} \cup \psi}{\bot}$$

$$\text{(Join)}\quad \frac{\{\mu \mathrel{\dot\in} L,\ \mu \mathrel{\dot\in} L'\} \cup \psi}{\{\mu \mathrel{\dot\in} (L \cap L')\} \cup \psi}\quad L \neq L' \qquad \text{(Empty)}\quad \frac{\{\mu \mathrel{\dot\in} \emptyset\} \cup \psi}{\bot}$$

$$\text{(Div1)}\quad \frac{\{\mu \mathrel{\ddot\amalg} \nu'\} \cup \{\mu \circ \nu \mathrel{\ddot\amalg} \nu'\} \cup \psi}{\{\mu \mathrel{\ddot\amalg} \nu'\} \cup \psi} \qquad \text{(Div2)}\quad \frac{\{\mu \circ \nu \mathrel{\ddot\amalg} \mu \circ \nu'\} \cup \psi}{\{\nu \mathrel{\ddot\amalg} \nu'\} \cup \psi}$$

$$\text{(DClash1)}\quad \frac{\{\mu \circ \nu \mathrel{\ddot\amalg} \mu\} \cup \psi}{\bot} \qquad \text{(DClash2)}\quad \frac{\{\mu \mathrel{\ddot\amalg} \mu\} \cup \psi}{\bot}$$

$$\text{(Pre)}\quad \frac{\{\mu \mathrel{\dot\prec} \nu,\ x[\mu]t,\ x[\nu]t'\} \cup \psi}{\{x[\mu]t,\ t[\nu]t'\} \cup \psi[\nu \leftarrow \mu \circ \nu]}\quad \mu \neq \nu$$

Figure 5.1: Simplification rules

The remaining rules are listed in figure 5.1.

The (Pre) rule needs some additional explanation. One might expect (Pre) to be of the form

$$\text{(Pre')}\quad \frac{\{\mu \mathrel{\dot\prec} \nu,\ x[\mu]t,\ x[\nu]t'\} \cup \psi}{\{x[\mu]t,\ t[\nu']t'\} \cup \psi[\nu \leftarrow \mu \circ \nu']}\quad \nu' \text{ new.}$$

But as we have mentioned, we have to define our rules in a way such that no additional variables are introduced. This is not satisfied by the rule (Pre'). For solving this problem note that ν is not used in the result of applying (Pre'). Hence, we can substitute ν' by ν, which has the effect that no new variable is needed. This leads to the definition of (Pre) as presented in figure 5.1.

The following proposition and lemma will show that the definition of (LangDec_Λ) is

meaningful.

Proposition 5.7 *If Λ is ϕ-closed and closed under intersection, then Λ is γ-closed for all $\mathcal{R}_\Lambda$-derivatives γ of ϕ.*

Proof. We will prove this lemma by induction over the length of derivations. We use the term $\mathrm{reg}(\gamma)$ to denote the set of regular languages used in γ. Then $\mathcal{R}_\Lambda$ is γ-closed if $\mathrm{reg}(\gamma) \subseteq \Lambda$.

Let γ be some $\mathcal{R}_\Lambda$-derivative of ϕ. For the base step $\gamma = \phi$ the lemma holds trivially. For the induction step let γ satisfy the induction hypotheses $\mathrm{reg}(\gamma) \subseteq \Lambda$ and let $r \in \mathcal{R}_\Lambda$ be a rule such that $\gamma \to_r \gamma'$.

If r is some clash rule, then $\mathrm{reg}(\gamma') = \emptyset$.

If r is not a clash rule and not in ($\mathrm{LangDec}_\Lambda$) or (Join), then $\mathrm{reg}(\gamma') = \mathrm{reg}(\gamma)$ and therefore $\mathrm{reg}(\gamma') \subseteq \Lambda$ by induction hypotheses. If $r \in (\mathrm{LangDec}_\Lambda)$, then r adds only regular languages $P, S \in \Lambda$.

Now let

$$r' = \frac{\{\mu \mathrel{\dot\in} L,\ \mu \mathrel{\dot\in} L'\} \cup \psi}{\{\mu \mathrel{\dot\in} (L \cap L')\} \cup \psi} \in (\mathrm{Join}).$$

By induction hypotheses we know that $L, L' \in \Lambda$. But then $(L \cap L') \in \Lambda$ since Λ is closed under intersection. □

We define a **finite non-deterministic automaton** $\mathcal{A}$ over the infinite set $\mathcal{L}$ to be a tuple $(Q_\mathcal{A}, i_\mathcal{A}, \delta_\mathcal{A}, Fin_\mathcal{A})$, where

1. $Q_\mathcal{A}$ is a finite set of states,
2. $i_\mathcal{A} \in Q_\mathcal{A}$ is the initial state,
3. $\delta_\mathcal{A} : Q_\mathcal{A} \times \mathcal{L} \to \wp(Q_\mathcal{A})$ is a transition function such that

$$\forall p \in Q_\mathcal{A}, Q \subseteq Q_\mathcal{A} : \{f \in \mathcal{L} \mid \delta_\mathcal{A}(p, f) = Q\} \text{ is finite or cofinite,} \tag{5.1}$$

4. and $Fin_\mathcal{A} \subseteq Q_\mathcal{A}$ are the final states.

With $\delta^*_\mathcal{A}$ we mean the unique extension of $\delta_\mathcal{A}$ to $\mathcal{L}^*$. The regular language that is accepted by an automaton $\mathcal{A}$ is defined as

$$L(\mathcal{A}) = \{w \mid \delta^*_\mathcal{A}(i_\mathcal{A}, w) \cap Fin_\mathcal{A} \neq \emptyset\}.$$

A finite deterministic automaton is a tuple $(Q_{\mathcal{A}}, i_{\mathcal{A}}, \delta_{\mathcal{A}}, Fin_{\mathcal{A}})$ where $Q_{\mathcal{A}}, i_{\mathcal{A}}$ and $Fin_{\mathcal{A}}$ have the same meaning as in the non-deterministic case, and $\delta_{\mathcal{A}}$ is a function $Q_{\mathcal{A}} \times \mathcal{L} \to Q_{\mathcal{A}}$ with the property that for all states p, q the set $\{f \in \mathcal{L} \mid \delta_{\mathcal{A}}(p, f) = q\}$ is finite or cofinite. $\delta^*_{\mathcal{A}}$ and $L(\mathcal{A})$ are defined analogous.

Note that although the standard constructions for transforming regular expression into a finite deterministic automaton and vice versa assume a finite alphabet, the proofs do not use this assumption at all. Hence, the constructions can be generalised to the infinite case.[4] A survey of the standard constructions can, e.g., be found in [HU79]. As an example, we show how to extend the construction of a deterministic automaton from a non-deterministic one. This construction is used in [HU79] in the proof of the equivalence of regular expressions and finite deterministic automatons.

Proposition 5.8 *For every non-deterministic automaton $\mathcal{A}$ there is a deterministic automaton $\mathcal{A}'$ with $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proof. Let $\mathcal{A} = (Q_{\mathcal{A}}, i_{\mathcal{A}}, \delta_{\mathcal{A}}, Fin_{\mathcal{A}})$ be a non-deterministic automaton. The standard construction yields a deterministic automaton $\mathcal{A}' = (\wp(Q_{\mathcal{A}}), \{i_{Q_{\mathcal{A}}}\}, \delta_{\mathcal{A}'}, Fin_{\mathcal{A}'})$ with

$$\begin{aligned} \delta_{\mathcal{A}'}(\{p_1, \ldots, p_n\}, f) &:= \bigcup_{i=1\ldots n} \delta_{\mathcal{A}}(p_i, f) \\ Fin_{\mathcal{A}'} &:= \{Q \subseteq Q_{\mathcal{A}} \mid Q \cap Fin_{\mathcal{A}} \neq \emptyset\} \end{aligned}$$

In order to show that $\mathcal{A}'$ is a finite deterministic automaton we have to show that for all $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ in $Q_{\mathcal{A}'} = \wp(Q_{\mathcal{A}})$ the set

$$F^Q_P = \{f \in \mathcal{L} \mid \delta_{\mathcal{A}'}(P, f) = Q\}$$

is either finite or cofinite. Now we can define F^Q_P also inductively as

$$\begin{aligned} F^{\emptyset}_{\{p_1,\ldots,p_n\}} &:= \bigcup_{i=1\ldots n} \{f \mid \delta_{\mathcal{A}}(p_i, f) = \emptyset\} \\ F^{Q\cup\{q\}}_{\{p_1,\ldots,p_n\}} &:= \bigcup_{i=1\ldots n} \{f \mid \delta_{\mathcal{A}}(p_i, f) \subseteq (Q \cup \{q\})\} \cap \bigcap_{Q' \subseteq Q} \overline{F^{Q'}_{\{p_1,\ldots,p_n\}}} \end{aligned}$$

Under the assumption that $\delta_{\mathcal{A}}$ satisfies 5.1, an easy induction over the cardinality of the sets shows that F^Q_P is finite or cofinite. □

The other constructions can be extended similar. Thus we get the following proposition.

[4] The same observation for tree automata was made in [Pod92].

Proposition 5.9 *For every regular expression L there is a finite deterministic automaton $\mathcal{A}$ with $L = L(\mathcal{A})$ and vice versa.*

Lemma 5.3 *For every prime clause ϕ there is a* finite Λ *such that Λ is ϕ-closed, closed under intersection and decomposition.*

Proof. Let $\mathrm{reg}(\phi) = \{L_1, \ldots, L_n\} \subseteq P(\mathcal{L}^+)$ be the set of regular languages used in ϕ and let $\mathcal{A}_i = (Q_{\mathcal{A}_i}, i_{\mathcal{A}_i}, \delta_{\mathcal{A}_i}, Fin_{\mathcal{A}_i})$ be finite, deterministic automatons such that $\mathcal{A}_i$ accepts L_i. For each $\mathcal{A}_i$ we define $\mathrm{dec}(\mathcal{A}_i)$ to be the set

$$\mathrm{dec}(\mathcal{A}_i) = \{\widehat{L_p^q} \mid p, q \in Q_{\mathcal{A}_i}\},$$

where $\widehat{L_p^q} = \{w \in \mathcal{L}^+ \mid \delta^*_{\mathcal{A}_i}(p, w) = q\}$.

Of course, each $\mathrm{dec}(\mathcal{A}_i)$ is finite and contains L_i. Furthermore, it is also closed under decomposition. The complete set of decompositions for a language $\widehat{L_p^q} \in \mathrm{dec}(\mathcal{A}_i)$ consists of the languages

$$P = \widehat{L_p^s} \text{ and } S = \widehat{L_s^q} \quad \text{for } s \in Q_{\mathcal{A}_i}.$$

We define Λ_0 to be $\bigcup_{i=1}^n \mathrm{dec}(\mathcal{A}_i)$. Λ_0 contains each $L_i \in \mathrm{reg}(\phi)$ and is closed under decomposition. Now let

$$\Lambda = \mathrm{fi}\,(\Lambda_0)$$

be the least set that contains Λ_0 and is closed under intersection. Then Λ is finite and ϕ-closed, since it contains each $L_i \in \mathrm{reg}(\phi)$.

We will prove that Λ is also closed under decomposition. Given some $L \in \Lambda$ and a path $w = w_1 w_2 \in L$, we have to find an appropriate decomposition P, S in Λ. Since each L in Λ can be written as a finite intersection

$$L = \bigcap_{k=1}^{m} L_k$$

with L_k in Λ_0, we know that $w = w_1 w_2$ is in L_k for $1..m$. As Λ_0 is closed under decomposition, there are languages P_k and S_k for $k = 1..m$ with $w_1 \in P_k$, $w_2 \in S_k$ and $P_k \circ S_k \subseteq L_k$. Let $P = \bigcap_{k=1}^m P_k$ and $S = \bigcap_{k=1}^m S_k$. Clearly, $w_1 \in P$, $w_2 \in S$ and $P \circ S \subseteq L$. Furthermore, $P, S \in \Lambda$ as Λ is closed under intersection. This implies that P, S is an appropriate decomposition for $w_1 w_2$. □

Before proceeding to some properties of the rule system, we present some sample derivations. We will start with the prime clause

$$\phi = \{x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^* \cup h^*),\ \mu_3 \dot{\in} h(f \cup g)\}\,.$$

To find an appropriate set of rules $\mathcal{R}_\Lambda$, we must generate a ϕ-closed set of languages Λ, which is closed under decomposition. By the construction used in the proof of the last lemma we obtain the following set of languages:

$$\Lambda = \{(f \cup g),\ (f^+ \cup h^+),\ h(f \cup g),\ h\}.$$

For the sake of clarity, we will say that we apply (PathRel) *on* some variable μ if we apply an instance of (PathRel) of the form

$$\frac{\{x[\mu]y\} \cup \psi}{\{\ldots\}}.$$

Figure 5.2 is an example of an $\mathcal{R}_\Lambda$-derivation which transforms ϕ into a pre-solved clause. We use the frames to highlight the corresponding parts of a clause which have been modified by the last rule application. The empty frame $\square$ denotes the deletion of a constraint. Note that we have removed the constraint $\mu_2 \dot{\prec} \mu_3$ in the fourth clause using the (Pre) rule, $\mu_1 \dot{\amalg} \mu_2 \circ \mu_3$ in the fifth clause using the (Div1) rule, and the constraint $\mu_2 \mathrel{\dot{\in}} (f^+ \cup h^+)$ in the last clause.

Next we want to examine two clashing $\mathcal{R}_\Lambda$-derivations. The first one (see figure 5.3) shows that $\mu_2 \dot{\prec} \mu_1$ cannot hold. The second clashing derivation (see figure 5.4) shows that μ_3 cannot be the prefix of μ_2.

Finally, here is the complete list of pre-solved clauses different from $\bot$ that are derivable from ϕ using $\mathcal{R}_\Lambda$:

$$\left\{ \begin{array}{c} \mu_1 \dot{\amalg} \mu_2,\ \mu_1 \dot{\amalg} \mu_3,\ \mu_2 \dot{\amalg} \mu_3,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \mathrel{\dot{\in}} (f \cup g),\ \mu_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \mu_3 \mathrel{\dot{\in}} h(f \cup g) \end{array} \right\}$$

$$\left\{ \mu_1 \dot{\amalg} \mu_2,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ y_2[\mu_3]y_3,\ \mu_1 \mathrel{\dot{\in}} (f \cup g),\ \mu_2 \mathrel{\dot{\in}} h,\ \mu_3 \mathrel{\dot{\in}} (f \cup g) \right\}$$

$$\left\{ \mu_1 \dot{\amalg} \mu_3,\ x[\mu_1]y_1,\ y_1[\mu_2]y_2,\ x[\mu_3]y_3,\ \mu_1 \mathrel{\dot{\in}} f,\ \mu_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \mu_3 \mathrel{\dot{\in}} (f \cup g) \right\}$$

$$\left\{ \mu_1 \dot{\amalg} \mu_3,\ y_1 \doteq y_2,\ x[\mu_1]y_1,\ x[\mu_3]y_3,\ \mu_1 \mathrel{\dot{\in}} f,\ \mu_3 \mathrel{\dot{\in}} (f \cup g) \right\}$$

5.4.2 Some Properties of the Rule System

For the rest of the paper we will call clauses that are derivable from prime clauses **admissible**.

Lemma 5.4

1. *Every admissible clause is basic.*

$$\{x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g)\}$$

$$\downarrow \quad \text{(PathRel) on } \mu_1$$

$$\left\{ \begin{array}{c} \boxed{\mu_1 \ \dot{\amalg}\ \mu_2,\ \mu_1 \ \dot{\amalg}\ \mu_3},\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(PathRel) on } \mu_2$$

$$\left\{ \begin{array}{c} \boxed{\mu_2 \ \dot{\prec}\ \mu_3},\ \mu_1 \ \dot{\amalg}\ \mu_2,\ \mu_1 \ \dot{\amalg}\ \mu_3,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(Pre)}$$

$$\left\{ \begin{array}{c} \boxed{\ },\ \mu_1 \ \dot{\amalg}\ \mu_2,\ \boxed{\mu_1 \ \dot{\amalg}\ \mu_2 \circ \mu_3},\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ \boxed{y_2[\mu_3]y_3}, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \boxed{\mu_2 \circ \mu_3 \dot{\in} h(f \cup g)} \end{array} \right\}$$

$$\downarrow \quad \text{(Div1)}$$

$$\left\{ \begin{array}{c} \mu_1 \ \dot{\amalg}\ \mu_2,\ \boxed{\ },\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ y_2[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_2 \circ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(LangDec}_\Lambda\text{)}$$

$$\left\{ \begin{array}{c} \mu_1 \ \dot{\amalg}\ \mu_2,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ y_2[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \boxed{\mu_2 \dot{\in} h,\ \mu_3 \dot{\in} (f \cup g)} \end{array} \right\}$$

$$\downarrow \quad \text{(Join)}$$

$$\{\mu_1 \ \dot{\amalg}\ \mu_2,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ y_2[\mu_3]y_3,\ \mu_1 \dot{\in} (f \cup g),\ \boxed{\ },\ \mu_2 \dot{\in} h,\ \mu_3 \dot{\in} (f \cup g)\}$$

Figure 5.2: A successful $\mathcal{R}_\Lambda$-derivation

$$\{x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g)\}$$

$$\downarrow \quad \text{(PathRel) on } \mu_2$$

$$\left\{ \begin{array}{c} \boxed{\mu_2 \dot{\prec} \mu_1,\ \mu_2 \dot{\amalg} \mu_3},\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(Pre)}$$

$$\left\{ \begin{array}{c} \boxed{\ },\ \mu_2 \dot{\amalg} \mu_3,\ \boxed{y_2[\mu_1]y_1},\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \boxed{\mu_2 \circ \mu_1 \dot{\in} (f \cup g)},\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(DecClash)}$$

$$\bot$$

Figure 5.3: A clashing $\mathcal{R}_\Lambda$-derivation

2. *If $\mu \dot{\prec} \nu$, $\mu \doteq \nu$ or $\mu \dot{\amalg} \nu$ is contained in some admissible clause ϕ, then there is a term t such that $t[\mu]t'$ and $t[\nu]t''$ is in ϕ.*

Proof. The proof of the first claim is left to the reader. The second claim will be proved by induction over the length of derivations. For prime clauses the claim holds trivially. For the induction hypotheses assume that we have proven the claim for every admissible clause ϕ that is derivable from a prime clause in n steps and let $\phi \to_r \phi'$. If r is different from (Pre), (PathRel), (Eq1,2) or (Div2), there is nothing to prove. Thus we have the following cases:

$r \in$ (PathRel): the claim holds by definition of (PathRel).

$r \in$ (Eq1,2): the claim is invariant under substitution of one variable ν by another variable μ if both $t[\mu]t'$ and $t[\nu]t''$ are contained in ϕ.

$r \in$ (Pre): then $\phi = \{\mu \dot{\prec} \nu,\ x[\mu]t,\ x[\nu]t'\} \cup \psi$ and $\phi' = \{x[\mu]t,\ t[\nu]t'\} \cup \psi[\nu \leftarrow \mu \circ \nu]$. The only subtree constraint that is changed is $x[\nu]t'$. But as ν is substituted by $\mu \circ \nu$, ϕ' does not contain any path equality or prefix constraints involving ν.

$r \in$ (Div2): then $\phi = \{\mu \circ \nu \ \dot{\amalg}\ \mu \circ \nu'\} \cup \psi$ and $\phi' = \{\nu \ \dot{\amalg}\ \nu'\} \cup \psi$. We will prove below that if $\mu \circ \nu$ is contained in some admissible clause γ, then there are terms t, t', t'' such that $t[\mu]t'$ and $t'[\nu]t''$ are contained in γ. This will complete the

$$\{x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3,\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g)\}$$

$$\downarrow \quad \text{(PathRel) on } \mu_3$$

$$\left\{ \begin{array}{c} \boxed{\mu_3 \ \dot{\amalg}\ \mu_1,\ \mu_3 \dot{\prec} \mu_2},\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(Pre)}$$

$$\left\{ \begin{array}{c} \mu_3 \ \dot{\amalg}\ \mu_1,\ \boxed{\square},\ x[\mu_1]y_1,\ \boxed{y_3[\mu_2]y_2},\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \boxed{\mu_3 \circ \mu_2 \dot{\in} (f^+ \cup h^+)},\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad (\text{LangDec}_\Lambda)$$

$$\left\{ \begin{array}{c} \mu_3 \ \dot{\amalg}\ \mu_1,\ x[\mu_1]y_1,\ y_3[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \boxed{\mu_2 \dot{\in} (f^+ \cup h^+),\ \mu_3 \dot{\in} (f^+ \cup h^+)},\ \mu_3 \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow \quad \text{(Join)}$$

$$\left\{ \begin{array}{c} \mu_3 \ \dot{\amalg}\ \mu_1,\ x[\mu_1]y_1,\ y_3[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \dot{\in} (f \cup g),\ \mu_2 \dot{\in} (f^+ \cup h^+),\ \boxed{\mu_3 \dot{\in} \emptyset} \end{array} \right\}$$

$$\downarrow \quad \text{(Empty)}$$

$$\bot$$

Figure 5.4: Another clashing $\mathcal{R}_\Lambda$-derivation

proof, since then $\mu \circ \nu \ \dot{\amalg} \ \mu \circ \nu'$ in ϕ implies that there are terms t_1, t_2, t_3 and t_1', t_2', t_3' with $\{t_1[\mu]t_2,\ t_2[\nu]t_3,\ t_1'[\mu]t_2',\ t_2'[\nu']t_3'\} \subseteq \phi$. But as ϕ is admissible, it is also basic by the first claim. Hence, t_1 equals t_1' and t_2 equals t_2'. Therefore, both $t_2[\nu]t_3$ and $t_2[\nu']t_3'$ are in ϕ and in ϕ'.

Thus it remains to show that if $\mu \circ \nu$ is used in some admissible clause γ, then there are terms t, t', t'' such that $t[\mu]t'$ and $t'[\nu]t''$ are contained in γ. Let ϕ be an admissible clause for which this holds, and let $\gamma \rightarrow_r \gamma'$. The only rules we have to consider are (Eq1,2) and (Pre). For (Eq1,2) note that the claim is invariant under consistent variable renaming. If $r \in$ (Pre), then we have to check the path term $\mu \circ \nu$ that is introduced by r. But by definition of (Pre) the clause γ' must contain both $x[\mu]t$ and $t[\nu]t'$. □

This lemma implies that one of (Eq1,2), (EqClash) or (CClash) can always be applied if a constraint $\mu \doteq \nu$ is contained in some admissible clause. The next lemma will show that different applications of (Pre) or (Eq1,2) will not interact. This means the application of one of these rules to some prefix or path equality constraint will not change any other prefix or path equality constraint contained in the same clause.

Lemma 5.5 *Given some prime clause ϕ and a derivation*

$$\phi = \phi_0 \rightarrow_{r_0} \phi_1 \cdots \phi_{n-1} \rightarrow_{r_{n-1}} \phi_n = \gamma$$

that contains an application of (PathRel). *Then $\mu \doteq \nu \in \gamma$ (resp. $\mu \dot{\prec} \nu \in \gamma$) implies $\mu \doteq \nu \in \phi_i$ (resp. $\mu \dot{\prec} \nu \in \phi_i$) for $i > k$, where k is the number of the last application of* (PathRel). *Furthermore, if $\mu \circ \nu$ is contained in γ, then either $\mu \circ \nu$ or $\mu \dot{\prec} \nu$ is contained in ϕ_i for $i > k$.*

Proof. We will use induction over length of derivations. Assume that we have proven the lemma for admissible clauses γ that are derivable in n steps and let $\gamma \rightarrow_r \gamma'$ with $r \notin$ (PathRel). If r is different from (Eq1,2) or (Pre), then there is nothing to prove. If $r \in$ (Eq1,2), then a constraint $\mu \dot{\prec} \nu$ or $\mu \doteq \nu$ in γ' can be missing in γ if and only if γ contains a constraint $\mu \doteq \nu'$ or $\mu \dot{\prec} \nu'$ (resp. $\nu' \doteq \nu$ or $\nu' \dot{\prec} \nu$) and r is of the form

$$\frac{\{\nu \doteq \nu', \ldots\} \cup \psi}{\cdots} \text{ with } \nu' \neq \nu \quad \left(\text{resp. } \frac{\{\mu \doteq \nu', \ldots\} \cup \psi}{\cdots} \text{ with } \nu' \neq \mu\right).$$

Hence, γ must contain at least two prefix or path equality constraints, the left sides of which are different. By induction hypotheses these path equality or prefix constraints

must have been introduced by the last application of (PathRel). But this contradicts to the definition of (PathRel). A similar argument can be given for the part of the lemma concerning path terms of form $\mu \circ \nu$.

If r is in (Pre), then we have to check only the second claim of the lemma, namely that $\mu \circ \nu$ contained in γ' implies that either $\mu \dot{\prec} \nu$ is in γ or $\mu \circ \nu$ is used in γ. For all path terms in γ' that are not introduced by this application of (Pre) this holds trivially. For the path term $\mu \circ \nu$ that is introduced, this is guaranteed by the application condition of (Pre), namely that γ must contain $\mu \dot{\prec} \nu$. □

We can derive from this lemma certain syntactic properties of admissible clauses which are needed for proving completeness and quasi-termination.

Corollary 5.1 *If $\mu \dot{\prec} \nu$ is contained in an admissible clause ϕ, then μ is different from ν. Furthermore, there is no other prefix or equality constraint in ϕ involving ν and neither $\nu \circ \nu'$ nor $\nu' \circ \nu$ is in ϕ.*

Note that by lemma 5.4 together with this corollary, either (CClash) or (Pre) is applicable if a constraint $\mu \dot{\prec} \nu$ is contained in an admissible clause. Furthermore, an application of (Pre) causes no violation of the restrictions that we have imposed on the syntax. This means that concatenation does not occur in prefix or path equality constraints; and concatenation of path variables is restricted to binary concatenation.

Lemma 5.6 *If $\mu \circ \nu \mathrel{\dot{\amalg}} \nu'$ is contained in an admissible clause ϕ with μ different from ν', then ϕ contains a constraint of form $\mu \mathrel{\dot{\amalg}} \nu'$, $\mu \doteq \nu'$ or $\mu \dot{\prec} \nu'$.*

Proof. We will prove a stronger result, namely that if $\{\mu \dot{\prec} \nu, \nu \mathrel{\dot{\amalg}} \nu'\} \subseteq \phi$ or $\{\mu \circ \nu \mathrel{\dot{\amalg}} \nu'\} \subseteq \phi$, then ϕ contains a constraint of form $\mu \mathrel{\dot{\amalg}} \nu'$, $\mu \doteq \nu'$ or $\mu \dot{\prec} \nu'$. We will prove this by induction over length of derivations. Assume that we have proven the claim for every admissible clause ϕ that is derivable in n steps from a prime clause and let $\phi \rightarrow_r \phi'$. Again we have to check only the rules (Pre), (PathRel), (Eq1,2) or (Div2):

$r \in$ (PathRel): we have to check only constraints $\nu \mathrel{\dot{\amalg}} \nu'$ that are already in ϕ. By lemma 5.4 we know that if $\nu \mathrel{\dot{\amalg}} \nu'$ is in ϕ, then there is a variable x with both $x[\nu]y$ and $x[\nu']z$ in ϕ. Hence, if (PathRel) adds the constraint $\mu \dot{\prec} \nu$, it must by definition also add a constraint $\mu \mathrel{\dot{\amalg}} \nu'$, $\mu \doteq \nu'$ or $\mu \dot{\prec} \nu'$.

$r \in$ (Eq1,2): the claim is invariant under consistent variable renaming.

$r \in$ (Pre): then $\phi = \{\mu \dot{\prec} \nu,\ x[\mu]t,\ x[\nu]t'\} \cup \psi$ and $\phi' = \{x[\mu]t,\ t[\nu]t'\} \cup \psi[\nu \leftarrow \mu \circ \nu]$. The only case that we have to check is that ϕ contains a constraint $\nu \dot{\amalg} \nu'$. Then ϕ' contains $\mu \circ \nu \dot{\amalg} \nu'$. By induction hypotheses ϕ must contain a constraint c of form $\mu \dot{\amalg} \nu'$, $\mu \doteq \nu'$ or $\mu \dot{\prec} \nu'$. Since (Pre) does not change c, this must hold also for ϕ'.

$r \in$ (Div2): then $\phi = \{\mu \circ \nu \dot{\amalg} \mu \circ \nu'\} \cup \psi$ and $\phi' = \{\nu \dot{\amalg} \nu'\} \cup \psi$. The only new divergence constraint that comes in is $\nu \dot{\amalg} \nu'$. But as ϕ contains both $\mu \circ \nu$ and $\mu \circ \nu'$, it may not contain $\mu \dot{\prec} \nu$ or $\mu \dot{\prec} \nu'$ by corollary 5.1. Hence, ϕ' does not contain such a constraint.

□

This lemma ensures that a constraint $\mu \circ \nu \dot{\amalg} \nu'$ is always reducible. If ν' equals μ, then we can apply (DClash1). If $\mu \dot{\amalg} \nu'$ is in ϕ, we can apply (Div1). If $\mu \doteq \nu'$ is in ϕ we can either apply (CClash) or (EqClash), or we can apply (Eq1,2) followed by (DClash1). If $\phi = \{\mu \dot{\prec} \nu', \mu \circ \nu \dot{\amalg} \nu'\} \cup \psi$, then we can either apply (CClash) or we can apply (Pre) yielding $\{\mu \circ \nu \dot{\amalg} \mu \circ \nu'\} \cup \psi'$, where we can apply (Div2).

5.4.3 Soundness and Completeness

Since we have a two-sorted logic, we have to redefine the notions of soundness and preservingness. For a set $\xi \subseteq \mathcal{X}$ we define $=_\xi$ to be the following relation on valuations of tree variables:

$$\alpha_\mathcal{X} =_\xi \alpha'_\mathcal{X} \quad \text{iff} \quad \text{for all } x \in \xi \text{ the equation } \alpha_\mathcal{X}(x) = \alpha'_\mathcal{X}(x) \text{ holds.}$$

Similarly, we define $=_\pi$ with $\pi \subseteq \mathcal{P}$ for path valuations. Let $\vartheta \subseteq \mathcal{X} \cup \mathcal{P}$ be a set of variables. For a given interpretation $\mathfrak{A}$ we say that a valuation $(\alpha_\mathcal{X}, \alpha_\mathcal{P})$ is a **ϑ-solution** of a clause ϕ in $\mathfrak{A}$ if there is a valuation $(\alpha'_\mathcal{X}, \alpha'_\mathcal{P})$ in $\mathfrak{A}$ such that

$$\alpha_\mathcal{X} =_{\mathcal{X} \cap \vartheta} \alpha'_\mathcal{X}, \quad \alpha_\mathcal{P} =_{\mathcal{P} \cap \vartheta} \alpha'_\mathcal{P} \quad \text{and} \quad (\alpha'_\mathcal{X}, \alpha'_\mathcal{P}) \models_\mathfrak{A} \phi.$$

The set of all ϑ-solutions of ϕ in $\mathfrak{A}$ is denoted by $[\![\phi]\!]^\mathfrak{A}_\vartheta$. We call $\mathcal{X}$-solutions just solutions and write $[\![\phi]\!]^\mathfrak{A}$ instead of $[\![\phi]\!]^\mathfrak{A}_\mathcal{X}$.[5] A clause ϕ is **ϑ-equivalent** to a clause

[5]By calling $\mathcal{X}$-solutions just solutions we intend to suggest that $\mathcal{X}$-solutions are the interesting one. The are two reasons for concentrating on $\mathcal{X}$-solutions rather than $\mathcal{X} \cup \mathcal{P}$-solutions. First, there are only tree variables in the original Kaplan/Maxwell syntax, and we have added path variables as an additional data structure. Our $\mathcal{X}$-solutions will be solutions for the corresponding clauses in the original syntax. And second, all of the rules we will present will preserve the solutions (i.e., $\mathcal{X}$-solutions) of a clause, but not necessarily the $\mathcal{X} \cup \mathcal{P}$-solutions.

γ (resp. a set of clauses Γ) if for every interpretation $\mathfrak{A}$, $[\![\phi]\!]^{\mathfrak{A}}_{\vartheta} = [\![\gamma]\!]^{\mathfrak{A}}_{\vartheta}$ (resp. $[\![\phi]\!]^{\mathfrak{A}}_{\vartheta} = \bigcup_{\gamma\in\Gamma}[\![\gamma]\!]^{\mathfrak{A}}_{\vartheta}$). Again we use equivalent as short for $\mathcal{X}$-equivalent.

A rule R is **ϑ-sound** if $\phi \to_R \gamma$ implies $[\![\phi]\!]^{\mathfrak{A}}_{\vartheta} \supseteq [\![\gamma]\!]^{\mathfrak{A}}_{\vartheta}$ for every interpretation $\mathfrak{A}$. R is called **ϑ-preserving** if $\phi \to_R \gamma$ implies $[\![\phi]\!]^{\mathfrak{A}}_{\vartheta} \subseteq [\![\gamma]\!]^{\mathfrak{A}}_{\vartheta}$ for every $\mathfrak{A}$. And R is **globally ϑ-preserving** if

$$\forall\mathfrak{A}: \quad [\![\phi]\!]^{\mathfrak{A}}_{\vartheta} \subseteq \bigcup_{\phi\to_R\gamma} [\![\gamma]\!]^{\mathfrak{A}}_{\vartheta}.$$

Proposition 5.10 *The rules* (Eq1,2), (EqClash), (Div1,2), (CClash), (Join), (Empty) and (DClash1,2) *are $\mathcal{X}\cup\mathcal{P}$-sound and $\mathcal{X}\cup\mathcal{P}$-preserving.*

Proposition 5.11 *The rule* (Pre) *is $\mathcal{X}$-sound and $\mathcal{X}$-preserving.*

For (Pre) we can even characterise pairs of path valuations which preserve the $\mathcal{X}$-solutions.

Proposition 5.12 *Let $\phi = \{\mu \doteq \nu, x[\mu]t, x[\nu]t'\} \cup \psi$ and γ be the result of applying* (Pre) *to ϕ. Given a pair of path valuations $\alpha_{\mathcal{P}}, \alpha'_{\mathcal{P}}$ with*

$$\alpha_{\mathcal{P}} =_{\mathcal{P}-\{\nu\}} \alpha'_{\mathcal{P}} \quad \text{and} \quad \alpha_{\mathcal{P}}(\nu) = \alpha_{\mathcal{P}}(\mu)\alpha'_{\mathcal{P}}(\nu) = \alpha'_{\mathcal{P}}(\mu)\alpha'_{\mathcal{P}}(\nu),$$

then for each interpretation $\mathfrak{A}$ and for each first order valuation $\alpha_{\mathcal{X}}$

$$(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathfrak{A}} \phi \leftrightarrow (\alpha_{\mathcal{X}}, \alpha'_{\mathcal{P}}) \models_{\mathfrak{A}} \gamma.$$

Proposition 5.13 *If Λ is closed under decomposition, then* ($\mathrm{LangDec}_{\Lambda}$) *is $\mathcal{X}\cup\mathcal{P}$-sound and globally $\mathcal{X}\cup\mathcal{P}$-preserving. Furthermore,* (PathRel) *is $\mathcal{X}\cup\mathcal{P}$-sound and globally $\mathcal{X}\cup\mathcal{P}$-preserving.*

Finally, we have to prove that the rules are complete. This means that given an input clause ϕ, for every solution $\alpha_{\mathcal{X}}$ of ϕ in some interpretation $\mathfrak{A}$ there is a pre-solved clause γ derivable from ϕ such that $\alpha_{\mathcal{X}}$ is a solution of γ. If the rule system is terminating, then for completeness one has to prove that the pre-solved clauses are just the irreducible clauses.

In our case this is not enough since the rule system can loop. Therefore, we have to prove explicitly that each solution of a given prime clause ϕ can be found in some pre-solved ϕ-derivative. We define $\mathrm{Irred}(\phi, \mathcal{R}_{\Lambda})$ to be the set all $\mathcal{R}_{\Lambda}$-derivatives of ϕ which are $\mathcal{R}_{\Lambda}$-irreducible, and $\mathrm{Pre\text{-}Solved}(\phi, \mathcal{R}_{\Lambda})$ to be the set of all pre-solved clauses which are derivable from ϕ. A set of rules $\mathcal{R}_{\Lambda}$ is said to be **ϕ-complete** wrt. to a set of variables ϑ if

1. $\text{Irred}(\phi, R_\Lambda) = \text{Pre-Solved}(\phi, R_\Lambda)$,
2. for every interpretation $\mathfrak{A}$

$$[\![\phi]\!]^{\mathfrak{A}}_{\vartheta} \subseteq \bigcup_{\gamma \in \text{Pre-Solved}(\phi, R_\Lambda)} [\![\gamma]\!]^{\mathfrak{A}}_{\vartheta}.$$

We will show that for every prime clause ϕ there is a set of regular languages Λ such that $\mathcal{R}_\Lambda$ is ϕ-complete wrt. the tree variables $\mathcal{X}$.

Theorem 5.1 (Completeness I) *Given a prime clause ϕ. If Λ is a set of regular languages that is ϕ-closed, closed under intersection and closed under decomposition, then every $\mathcal{R}_\Lambda$-derivative γ of ϕ that is not pre-solved is $\mathcal{R}_\Lambda$-reducible.*

Proof. Let γ be a $\mathcal{R}_\Lambda$-derivative of ϕ that is not pre-solved. We will check all conditions that are stated in the definition on page 117.

If one of the conditions 1–3 is not satisfied by γ, then one of the rules (CClash), (Join) or (Empty) will apply.

Now let's check the conditions 4 and 5:

γ **contains a constraint** $\mu \circ \nu \mathrel{\dot\in} L$**.** As Λ is ϕ-closed, we know that Λ is also γ-closed by lemma 5.7. Therefore we can apply (LangDec_Λ) or (DecClash).

γ **contains a constraint** $\mu \circ \nu \mathrel{\dot{\amalg}} \mu' \circ \nu'$**.** By lemma 5.5 we know that in every $\mathcal{R}_\Lambda$-derivation for γ the last application of (PathRel) must have introduced the constraints $\mu \mathrel{\dot\prec} \nu$ and $\mu' \mathrel{\dot\prec} \nu'$. By the definition of (PathRel) this implies that μ equals μ'. Hence, we can apply (Div2).

γ **contains a constraint** $\mu \circ \nu \mathrel{\dot{\amalg}} \nu'$**.** If ν' equals μ, then we can directly apply (DClash1). Otherwise, there is by lemma 5.6 a constraint $\mu \doteq \nu'$, $\mu \mathrel{\dot\prec} \nu'$ or $\mu \mathrel{\dot{\amalg}} \nu'$ in γ. If $\mu \doteq \nu'$ is in γ, we can apply one of (CClash), (EqClash) or (Eq1,2) by lemma 5.4. Applying (Eq1,2) results in the substitution of ν' by μ. The remaining constraint $\mu \circ \nu \mathrel{\dot{\amalg}} \mu$ can be reduced using (DClash1). If $\mu \mathrel{\dot\prec} \nu'$ is in γ, then we can apply either (CClash) or (Pre) by lemma 5.4 and corollary 5.1. In the later case we obtain the constraint $\mu \circ \nu \mathrel{\dot{\amalg}} \mu \circ \nu'$, which can be reduced using (Div2). The last case is that $\mu \mathrel{\dot{\amalg}} \nu'$ is in γ, where we can apply (Div1).

γ **contains a constraint** $\mu \doteq \nu$**.** Then one of (CClash), (EqClash) or (Eq1,2) is applicable by lemma 5.4.

γ contains a constraint $\mu \dot{\prec} \nu$. Then either (CClash) or (Pre) is applicable by lemma 5.4 and corollary 5.1.

The remaining case is that γ does not satisfy the last condition of a pre-solved clause, namely that $\mu \dot{\amalg} \nu$ with $\mu \neq \nu$ in γ if and only if $x[\mu]t$ and $x[\nu]t'$ in γ. Given the above, we can now assume that γ does not contain a path concatenation or a prefix or path equality constraint.

There are three possibilities for γ to violate the last condition. The first is that γ contains a constraint of the form $\mu \dot{\amalg} \mu$. Then (DClash2) is applicable. The second is that there is a constraint $\mu \dot{\amalg} \nu$ with $x[\mu]t \in \gamma$ and $x'[\nu]t' \in \gamma$ such that x is different from x'. But this is excluded by lemma 5.4.

The last case is that there are different path variables μ and ν such that $x[\mu]t$ and $x[\nu]t'$ are in γ but $\mu \dot{\amalg} \nu$ is not. As γ contains no concatenation and no path equality or prefix constraints, the rule (PathRel) is applicable. □

Next we have to establish the second condition for ϕ-completeness, namely that for every interpretation $\mathfrak{A}$ and for every solution $\alpha_{\mathcal{X}}$ of ϕ there is a pre-solved ϕ-derivative γ with $\alpha_{\mathcal{X}} \in [\![\gamma]\!]^{\mathfrak{A}}$. This property is needed since our rule system can loop. Let us recall an example of a looping derivation in order to explain the main idea involved in the second part of the completeness proof. In contrast with our first example of a looping derivation (see page 112), we will omit the path restrictions, since they are not needed for what we want to demonstrate. Let ϕ be the clause

$$\phi = \{x[\mu]x, x[\nu]y\}.$$

A looping derivation can consist of an application of (PathRel) yielding the clause $\phi_1 = \{\mu \dot{\prec} \nu, x[\mu]x, x[\nu]y\}$, followed by an application of (Pre) on γ yielding $\phi_2 = \phi$.[6] Clearly, the cause of the looping derivation is the rule (Pre). We will later prove that, indeed, every infinite derivation must use the (Pre) rule infinitely often.

To prove the second completeness condition we restrict the set of allowed derivations of a prime clause ϕ to those depending on some arbitrary but fixed valuation $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$ with $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathfrak{A}} \phi$. This control will guarantee that

1. $\alpha_{\mathcal{X}}$ is a solution of every clause in the derivation,
2. under this control, all derivations are finite.

[6] The first example of a looping derivation on page 112 shows that the situation is no different if we add path restrictions.

We will additionally show that even under this control the irreducible clauses are just the pre-solved clauses. Hence this control will give us, for every clause ϕ and every initial solution $\alpha_{\mathcal{X}}$, a pre-solved ϕ-derivative that has $\alpha_{\mathcal{X}}$ as an solution.

We will add this further control only on the non-deterministic rules (PathRel) and (LangDec$_\Lambda$), thus restricting the set of instances of these rules that may be applied. We allow only those instances which preserve the valuation $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$. Using our above example, if $\alpha_{\mathcal{P}}$ satisfies

$$\alpha_{\mathcal{P}}(\mu) = f \quad \text{and} \quad \alpha_{\mathcal{P}}(\nu) = g$$

we may apply only that instance of (PathRel) which transforms ϕ into $\phi_1 = \{\mu \mathbin{\dot{\amalg}} \nu, x[\mu]x, x[\nu]y\}$. Since the choice of the instances depends only on the path valuation, we will call such restricted derivations $\alpha_{\mathcal{P}}$-strict.

It is easy to see that the above restriction will always enforce finiteness of derivations if the initial path valuation $\alpha_{\mathcal{P}}$ satisfies

$$\alpha_{\mathcal{P}}(\mu) \not\prec \alpha_{\mathcal{P}}(\nu) \quad \text{where} \quad \mu \neq \nu \wedge x[\mu]y \in \phi \wedge x[\nu]z \in \phi.$$

One might say that in this case $\alpha_{\mathcal{P}}$ is prefix-free with respect to ϕ.

For initial path valuations which are not prefix-free we must have a closer look at the (Pre) rule, since this rule is the cause of looping derivations. Since the (Pre) rule is not $\mathcal{P}$-preserving, it may happen that the clause γ resulting of an application of (Pre) is not valid under the initial valuation $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$. But as (Pre) is $\mathcal{X}$-preserving, we know that there is a $\alpha'_{\mathcal{P}}$ such that $(\alpha_{\mathcal{X}}, \alpha'_{\mathcal{P}}) \models_{\mathfrak{A}} \gamma$.

Hence, in a $\alpha_{\mathcal{P}}$-strict derivation we can keep the initial valuation $\alpha_{\mathcal{X}}$ of the tree variables, but we must change the path valuation every time the (Pre) rule is applied. Since application of (Pre) is the cause of looping derivations, this implies that we can obtain finiteness of $\alpha_{\mathcal{P}}$-strict derivations if we guarantee that after a finite number of (Pre) applications the initial path valuation has been transformed into a prefix-free path valuation.

We will again turn to our example to clarify this. If the initial path valuation $\alpha_{\mathcal{P}}$ for ϕ is of the form

$$\alpha_{\mathcal{P}}(\mu) = f \quad \text{and} \quad \alpha_{\mathcal{P}}(\nu) = fffg,$$

the first rule in a $\alpha_{\mathcal{P}}$-strict ϕ-derivation could be an application of (PathRel) transforming $\phi = \phi_0$ into $\phi_1 = \{\mu \mathbin{\dot{\prec}} \nu, x[\mu]x, x[\nu]y\}$. Now we are able to apply (Pre), which implies that we have to change $\alpha_{\mathcal{P}}$. Using proposition 5.12 we can use the following $\alpha'_{\mathcal{P}}$:

$$\alpha'_{\mathcal{P}}(\mu) = f \quad \text{and} \quad \alpha'_{\mathcal{P}}(\nu) = ffg.$$

Proposition 5.12 guarantees that this can be done without loosing $\mathcal{X}$-preservingness. Note that we have shortened $\alpha_{\mathcal{P}}(\nu)$ by f. Now we could iterate this twice more before ending up with a prefix-free path valuation.

After these remarks we can turn to the technical part.

Theorem 5.2 (Completeness-II) *Let ϕ be a prime clause, let Λ be a set of regular languages which is ϕ-closed, closed under intersection and decomposition. Then $\mathcal{R}_\Lambda$ is ϕ-complete wrt. the tree variables $\mathcal{X}$.*

First we need an additional lemma.

Lemma 5.7 *There are no infinite derivations using only finitely many instances of* (Pre).

Proof. Assume there is such a derivation. Then there exists an infinite sub-derivation not using any instance of (Pre). Let ϕ be the starting point of such a derivation. Let γ be some clause. Then we define the following functions on γ:

$$\begin{array}{rcl}
\Theta_1(\gamma) & = & \text{number of concatenations in } \gamma \\
\Theta_2(\gamma) & = & \text{number of different path variables in } \gamma \\
\Delta^\phi(\gamma) & = & \text{number of constraints } \mu \mathrel{\dot\diamond} \nu \text{ with } \mathrel{\dot\diamond} \in \{\doteq, \dot\prec, \dot\Pi\}, \\
 & & \mu, \nu \in \mathcal{V}_{\mathcal{P}}(\phi) \text{ and } \mu \mathrel{\dot\diamond} \nu \text{ \underline{not in} } \gamma \\
\Pi(\gamma) & = & \text{total number of constraints in } \gamma
\end{array}$$

We define $\Theta(\gamma)$ to be the tuple $\langle \Theta_1(\gamma), \Theta_2(\gamma) \rangle$. Using the functions Θ, Δ^ϕ and Π we can construct a partial order on clauses by defining $\gamma <_\phi \gamma'$ iff

$$(\Theta(\gamma) < \Theta(\gamma'))$$

or

$$(\Theta(\gamma) = \Theta(\gamma')) \quad \wedge \quad (\Delta^\phi(\gamma) < \Delta^\phi(\gamma'))$$

or

$$(\Theta(\gamma) = \Theta(\gamma')) \quad \wedge \quad (\Delta^\phi(\gamma) = \Delta^\phi(\gamma')) \quad \wedge \quad (\Pi(\gamma) = \Pi(\gamma')).$$

Here $<$ is the lexicographic ordering on tuples for $\Theta(\gamma)$ and elsewhere the usual numeric comparison. It is easy to check, that $<_\phi$ defines a well-founded, partial ordering on clauses.

	Θ_1	Θ_2	Δ^ϕ	Π
(PathRel)	$=$	$=$	$<$	
(Eq1,2)	$=$	$<$		
(LangDec_Λ)	$<$	$=$		
(Join)	$=$	$=$	$=$	$<$
(Div1)	$<$	$=$		
(Div2)	$<$	$=$		

Table 5.1: Monotonicity of the rules wrt. the measure functions.

Let γ be some derivation of ϕ. Now $\mathcal{V}_\mathcal{P}(\gamma) \subseteq \mathcal{V}_\mathcal{P}(\phi)$ holds, which is important for the value of Δ^ϕ. In table 5.1 we have summarised for every non-clash rule other than (Pre) the variation of $\Theta(\gamma)$, $\Delta^\phi(\gamma)$ and $\Pi(\gamma)$[7]. The clash rules are not considered because they automaticly terminate every derivation. The table shows that for every rule $r\ \gamma \to_r \gamma'$ implies $\gamma' <_\phi \gamma$. Because $<_\phi$ is a well-founded ordering and therefore cannot have infinite descending chains, this contradicts our assumption that there is a infinite derivation not using (Pre). □

Corollary 5.2 *There are no infinite derivations using only finitely many instances of* (PathRel).

Proof. By the above lemma we know that there are no infinite derivations without infinite use of (Pre). But (Pre) removes the constraints $\mu \dot{\prec} \nu$, the existence of which is an application condition for (Pre). But additional constraints of form $\mu \dot{\prec} \nu$ are only introduced by (PathRel). □

Proof of theorem 5.2 (Completeness II). The first condition for ϕ-completeness was proved in theorem 5.1 (Completeness I). For the second, let $\mathfrak{A}$ be some interpretation and $(\alpha_\mathcal{X}, \alpha_\mathcal{P})$ be a valuation with $(\alpha_\mathcal{X}, \alpha_\mathcal{P}) \models_\mathfrak{A} \phi$. We have to show that there is a $\mathcal{R}_\Lambda$-derivative γ of ϕ which is pre-solved and satisfies $\exists \alpha'_\mathcal{P} : (\alpha_\mathcal{X}, \alpha'_\mathcal{P}) \models_\mathfrak{A} \gamma$. This will be done by defining $\alpha_\mathcal{P}$-strict derivations, which will always end up in a pre-solved clause. As we have mentioned, we have to redefine the path valuation every time (Pre) is applied. This leads to the following definition: a derivation

$$\phi = \phi_0 \to_{r_0} \phi_1 \cdots \phi_n \to_{r_n} \phi_{n+1} \cdots$$

[7]If a rule decreases the Θ-value, the clause resulting from applying this rule is smaller than the input clause wrt. $<_\phi$ independently of the effects of the rule on the Δ^ϕ-part. Therefore, we omit the corresponding Δ^ϕ-entries in this case; and similarly for the Π-part.

is called $\alpha_{\mathcal{P}}$-strict if there is a family of path valuations $(\alpha^i_{\mathcal{P}})$ such that

1. $\alpha^0_{\mathcal{P}} = \alpha_{\mathcal{P}}$;
2. for each i the proposition $(\alpha_{\mathcal{X}}, \alpha^i_{\mathcal{P}}) \models_{\mathfrak{A}} \phi_i$ holds; and
3. for each i
   - $r_i \notin$ (Pre) implies $\alpha^i_{\mathcal{P}} = \alpha^{i+1}_{\mathcal{P}}$ and
   - $r_i = \frac{\{\mu \dot{\prec} \nu, \ \cdots\} \cup \psi}{\cdots} \in$ (Pre) implies
   $$\alpha^i_{\mathcal{P}} =_{\mathcal{P}-\{\nu\}} \alpha^{i+1}_{\mathcal{P}} \qquad \text{and} \qquad \alpha^i_{\mathcal{P}}(\nu) = \alpha^{i+1}_{\mathcal{P}}(\mu)\alpha^{i+1}_{\mathcal{P}}(\nu).$$

Now for every $\alpha_{\mathcal{P}}$-strict $(\phi, \mathcal{R}_\Lambda)$-derivation

$$\phi = \phi_0 \to_{r_0} \phi_1 \cdots \phi_{n-1} \to_{r_{n-1}} \phi_n$$

where ϕ_n is not pre-solved, there is a $\alpha_{\mathcal{P}}$-strict continuation, as the following argumentation shows. If ϕ_n is not pre-solved, then there is (by theorem 5.1) a rule which is applicable. We have to show that there is an applicable rule instance such that a corresponding $\alpha^{n+1}_{\mathcal{P}}$ can be found.

If the applicable rule is different from (Pre), then we know that there is an appropriate path valuation $\alpha^{n+1}_{\mathcal{P}}$, as all rules different from (Pre) are either $\mathcal{X} \cup \mathcal{P}$-preserving or globally $\mathcal{X} \cup \mathcal{P}$-preserving. If (Pre) is applicable, then proposition 5.12 shows that we can find an appropriate $\alpha^{n+1}_{\mathcal{P}}$.

Next we must show that there is no infinite $\alpha_{\mathcal{P}}$-strict $(\phi, \mathcal{R}_\Lambda)$-derivation, which finally proves the lemma. This is done by introducing a norm on path valuations. For a path valuation $\alpha_{\mathcal{P}}$ we define $|\alpha_{\mathcal{P}}|_\phi$ to be:

$$|\alpha_{\mathcal{P}}|_\phi = \sum_{\mu \in \mathcal{V}_{\mathcal{P}}(\phi)} |\alpha_{\mathcal{P}}(\mu)|.$$

Now let

$$\phi_i \to_{r_i} \phi_{i+1}$$

be a step in some $\alpha_{\mathcal{P}}$-strict $(\phi, \mathcal{R}_\Lambda)$-derivation and let $\alpha^i_{\mathcal{P}}, \alpha^{i+1}_{\mathcal{P}}$ be the corresponding path valuations. If $r_i \notin$ (Pre) we know that $\alpha^i_{\mathcal{P}} = \alpha^{i+1}_{\mathcal{P}}$ and hence $|\alpha^i_{\mathcal{P}}|_\phi = |\alpha^{i+1}_{\mathcal{P}}|_\phi$. If $r_i \in$ (Pre) we know by the third condition of $\alpha_{\mathcal{P}}$-strictness that there are μ and ν such that

$$\alpha^i_{\mathcal{P}} =_{\mathcal{P}-\{\nu\}} \alpha^{i+1}_{\mathcal{P}} \qquad \text{and} \qquad \alpha^i_{\mathcal{P}}(\nu) = \alpha^{i+1}_{\mathcal{P}}(\mu)\alpha^{i+1}_{\mathcal{P}}(\nu).$$

As $\mathcal{V}_{\mathcal{P}}(\phi_{i+1}) \subseteq \mathcal{V}_{\mathcal{P}}(\phi_i) \subseteq \mathcal{V}_{\mathcal{P}}(\phi)$ this implies $|\alpha^{i+1}_{\mathcal{P}}|_\phi < |\alpha^i_{\mathcal{P}}|_\phi$.

As there are no infinite derivations without infinite use of (Pre) this proves that there are no infinite $\alpha_{\mathcal{P}}$-strict derivations. □

5.4.4 Quasi-Termination

Lemma 5.8 *Let ϕ be a prime clause and Λ be a finite ϕ-closed set of regular languages. Then the set of all $\mathcal{R}_\Lambda$-derivatives of ϕ is finite.*

Proof. We will first consider the sets $\mathcal{C}$ which contains every atomic constraint that occur in at least one $\mathcal{R}_\Lambda$-derivative of ϕ. $\mathcal{C}$ could be seen as the union of all $\mathcal{R}_\Lambda$-derivatives of ϕ. We will show that $\mathcal{C}$ is finite. As every $\mathcal{R}_\Lambda$-derivative of ϕ is a subset of $\mathcal{C}$ this will prove the lemma.

First we know that no rule adds new variables. This implies that there are at most $n_1 = |\mathcal{V}_\mathcal{P}(\phi)| + |\mathcal{V}_\mathcal{P}(\phi)|^2$ many different path terms. By lemma 5.7 we know that Λ is γ-closed for every $\mathcal{R}_\Lambda$-derivative γ of ϕ, which implies that at most $|\Lambda|$ different regular languages are used in the $\mathcal{R}_\Lambda$-derivatives of ϕ.

Let n_{con} be the number of constants used in ϕ. Note that no rule adds new constants. Therefore $\mathcal{C}$ contains at most $|n_{\text{con}} + \mathcal{V}_\mathcal{X}(\phi)|^2$ equality constraints, $|n_{\text{con}} + \mathcal{V}_\mathcal{X}(\phi)| * |\mathcal{V}_\mathcal{P}(\phi)| * |n_{\text{con}} + \mathcal{V}_\mathcal{X}(\phi)|$ subtree constraints, n_1^2 path divergence constraints, $|\mathcal{V}_\mathcal{P}(\phi)|^2$ prefix and path equality constraints and $n_1 * |\Lambda|$ path restriction constraints. □

Theorem 5.3 *For every prime clause ϕ there exists a set of regular languages Λ such that $\mathcal{R}_\Lambda$ is ϕ-complete wrt. $\mathcal{X}$ and the set* Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ *is finite and computable.*

Proof. Let reg(ϕ) be the set of regular languages used in ϕ. By lemma 5.3 there must be a finite Λ such that Λ is ϕ-closed, closed under intersection and decomposition. Note that the construction of the set Λ given in lemma 5.3 is effective. Then $\mathcal{R}_\Lambda$ is ϕ-complete wrt. $\mathcal{X}$ by theorem 5.2. By lemma 5.8 we know that Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ must be finite. Hence, it suffices to prove that the set Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ is computable.

To do this we will consider loop-free derivations. A derivation is called loop-free if it is not of the form

$$\phi_0 \rightarrow_{r_1} \ldots \rightarrow_{r_i} \phi_i \ldots \rightarrow_{r_k} \phi_k \ldots,$$

where $\phi_i = \phi_k$. In order to generate the set of derivatives (or a subset of them) it is enough to consider loop-free derivations. This is because for every pair γ, γ' every γ-derivation which yields γ' and is not loop-free can be replaced with a shorter derivation by removing some loop. Iterating this step finally yields a loop-free γ-derivation for γ'.

Furthermore, the set of all loop-free $(\phi, \mathcal{R}_\Lambda)$-derivations must be finite since $\mathcal{R}_\Lambda$ can only generate finitely many $\mathcal{R}_\Lambda$-derivatives of ϕ by lemma 5.8, and there are only finitely many rules of $\mathcal{R}_\Lambda$ applicable on every $\mathcal{R}_\Lambda$-derivative of ϕ. But as we

have mentioned we need to consider only the loop-free derivations, which shows that Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ is computable. □

Corollary 5.3 *For every prime clause ϕ there exists a finite and computable set of pre-solved clauses Γ such that Γ is equivalent to ϕ.*

Proof. Follows from the last theorem and the fact, that every rule is at least $\alpha_\mathcal{X}$-sound. □

5.5 The Second Phase: Satisfiability of Pre-Solved Clauses

In this section we present a rule system that transforms each pre-solved clause into an equivalent set of solved clauses (interpreted as a disjunction), each of which is different from $\bot$ and is thus satisfiable by lemma 5.2. A pre-solved clause is satisfiable if and only if the corresponding set is non-empty.

We will first make a minor redefinition of divergence. We say that two paths u, v are **directly diverging** (written $u \amalg_0 v$) if there are features $f \neq g$ such that $u \in f\mathcal{L}^*$ and $v \in g\mathcal{L}^*$. Then $u \amalg v$ holds if there are a possible empty prefix w and paths u', v' such that $u = wu'$ and $v = wv'$ and $u' \amalg_0 v'$. Using this definition of divergence and the additional atomic constraint

$$\mu \mathrel{\dot{\amalg}_0} \nu \qquad \textbf{direct divergence},$$

we can (non-deterministically) transform a clause $\phi = \{\mu_1 \mathrel{\dot{\amalg}} \mu_2\} \cup \psi$ into either $\{\mu_1 \mathrel{\dot{\amalg}_0} \mu_2\} \cup \psi$ or $\{\mu_1 \doteq \nu \circ \mu_1',\ \mu_2 \doteq \nu \circ \mu_2',\ \mu_1' \mathrel{\dot{\amalg}_0} \mu_2'\} \cup \psi$.[8] By the definition of $\amalg_0$ we can reduce (non-deterministically) the constraints of form $\mu_1 \mathrel{\dot{\amalg}_0} \mu_2$ into $\{\mu_1 \mathrel{\dot{\in}} f\mathcal{L}^*, \mu_2 \mathrel{\dot{\in}} g\mathcal{L}^*\}$ with $f \neq g$. The aim is to process all divergence constraints this way in order to achieve a solved clause.

Before we can present the rule system for solving clause, we have to do two things. First, we have to redefine the notion of a solved clause, since we have extended the syntax by the constraint $\mathrel{\dot{\amalg}_0}$. Without loss of generality we can in the following assume that every clause ϕ contains for every path variable $\mu \in \mathcal{V}_\mathcal{P}(\phi)$ a path restriction $\mu \mathrel{\dot{\in}} L$. We say that a clause over the extended syntax is **solved** if it is either $\bot$ or satisfies the conditions of a solved clause as stated in Section 5.3, page 118 plus additionally

[8] The first case is needed because we do not allow values of path variables to be empty paths.

7. for every pair of variables μ, ν such that $\mu \neq \nu$, $\mu \; \dot{\amalg}_0 \; \nu \in \phi$ if and only if ϕ contains a path restriction $\mu \mathrel{\dot{\in}} L$ or $\nu \mathrel{\dot{\in}} L$ with $L = F \circ L'$ and $F \subseteq \mathcal{L}$ is co-finite.

Proposition 5.14 *Every solved clause different from $\bot$ is satisfiable.*

And second, we have to reformulate the reduction of divergence constraints. The reason is that we have to evaluate constraints of the form $\mu_1 \doteq \nu \circ \mu_1'$. This can produce constraints of the forms $\mu \circ \nu \mathrel{\dot{\in}} L$ and $\mu \circ \nu \; \dot{\amalg} \; \nu'$. The second is problematic as we must guess the relation between μ and ν'. This complicates the termination proof.

We will avoid this problem by using a special property of pre-solved clauses, namely that $\mu \; \dot{\amalg} \; \nu$ is in a pre-solved clause ϕ iff $x[\mu]y$ and $x[\nu]z$ are in ϕ. Hence, if $\mu \; \dot{\amalg} \; \nu$ and $\nu \; \dot{\amalg} \; \nu'$ are in ϕ, then $\mu \; \dot{\amalg} \; \nu'$ is also in ϕ. This implies that we can write ϕ as $\dot{\amalg}(A_1) \uplus \ldots \uplus \dot{\amalg}(A_n) \uplus \psi$, where $\dot{\amalg}(A)$ abbreviates

$$\{\mu \; \dot{\amalg} \; \mu' \mid \mu \neq \mu' \; \wedge \; \mu, \mu' \in A\},$$

$A_1, \ldots, A_n$ are disjoint sets of path variables and ψ contains no divergence constraints. Now given such a constraint $\dot{\amalg}(A)$, and some valuation $\alpha_{\mathcal{P}}$ with $\alpha_{\mathcal{P}} \models \dot{\amalg}(A)$, suppose that a whole set of path variables $A_1 \subseteq A$ diverge under $\alpha_{\mathcal{P}}$ with the same prefix, i.e., there is a path p such that

$$\forall \mu \in A_1 : \alpha_{\mathcal{P}}(\mu) = p p_\mu \;\; \text{and} \;\; \forall \mu, \mu' \in A_1 : \mu \neq \mu' \rightarrow p_\mu \amalg_0 p_{\mu'}.$$

Then we can replace the constraints set $\dot{\amalg}(A_1) \subseteq \dot{\amalg}(A)$ by

$$A_1 \doteq \nu \circ A_1' \cup \; \dot{\amalg}_0(A_1')$$

under this valuation, since every valuation $\alpha'_{\mathcal{P}}$ with $\alpha'_{\mathcal{P}} \models (A_1 \doteq \nu \circ A_1' \cup \; \dot{\amalg}_0(A_1'))$ satisfies $\alpha_{\mathcal{P}} =_{A_1} \alpha'_{\mathcal{P}}$. Here ν is new, $A_1' = \{\mu_1', \ldots, \mu_n'\}$ is a fresh copy of $A_1 = \{\mu_1, \ldots, \mu_n\}$ and $A \doteq \nu \circ A_1'$ abbreviates the clause $\{\mu_1 \doteq \nu \circ \mu_1', \ldots, \mu_n \doteq \nu \circ \mu_n'\}$. $\dot{\amalg}_0(A)$ is defined similarly to $\dot{\amalg}(A)$. If $\alpha_{\mathcal{P}}$ additionally satisfies

$$\forall \mu \in (A - A_1) : p \amalg \alpha_{\mathcal{P}}(\mu),$$

then we can replace the set of constraints $\dot{\amalg}(A)$ by

$$\dot{\amalg}(\{\nu\} \cup (A - A_1)) \; \cup \; A_1 \doteq \nu \circ A_1' \cup \; \dot{\amalg}_0(A_1')$$

under the valuation $\alpha_{\mathcal{P}}$.

Now for every path valuation $\alpha_{\mathcal{P}}$ with $\alpha_{\mathcal{P}} \models \dot{\amalg}(A)$ there must be a set A_1 with the properties stated above. We can find an appropriate A_1 by taking a path p which is maximal in

$$\{p \mid \exists \mu, \mu' \in A : \mu \neq \mu' \wedge p \prec \alpha_{\mathcal{P}}(\mu) \wedge p \prec \alpha_{\mathcal{P}}(\mu')\},$$

and defining A_1 as $\{\mu \in A \mid p \prec \alpha_{\mathcal{P}}(\mu)\}$.

This finally leads to the following non-deterministic rule, where we also consider the effects of $A_1 \doteq \nu \circ A_1'$ on the subtree constraints in ϕ:

$$\text{(Reduce}_1\text{)} \quad \frac{x A_1 Y_1 \cup \dot{\amalg}(A) \cup \psi}{\{x[\nu]z\} \cup z A_1' Y_1 \cup \dot{\amalg}_0(A_1') \cup \dot{\amalg}(\{\nu\} \cup A_2) \cup \psi'}$$

where $\psi' = \psi[\mu_1 \leftarrow \nu \circ \mu_1', \ldots, \mu_n \leftarrow \nu \circ \mu_n']$, $A_1 \uplus A_2 = A$, $|A_1| > 1$ and z, ν new. A_1' is a disjoint copy of A_1. $x A_1 Y_1$ is short for $\{x[\mu_1]y_1, \ldots, x[\mu_n]y_n\}$. ψ may not contain constraints of form $\delta \circ \delta' \mathrel{\dot{\in}} L$ in ψ.

Note that we have avoided constraints of the form $\mu \circ \nu \mathrel{\dot{\amalg}} \nu'$. We also employ the non-deterministic rules

$$\text{(Reduce}_2\text{)} \quad \frac{\dot{\amalg}(A) \cup \psi}{\dot{\amalg}_0(A) \cup \psi}$$

$$\text{(Solv)} \quad \frac{\{\mu \mathrel{\dot{\amalg}_0} \nu\} \cup \psi}{\{\mu \mathrel{\dot{\in}} f \circ F^*,\ \mu \mathrel{\dot{\in}} g \circ F^*\} \cup \psi} \quad f \neq g$$

if ϕ contains constraints $\mu \mathrel{\dot{\in}} L_1$ and $\nu \mathrel{\dot{\in}} L_2$ such that $L_1 = F_1 L_1'$ and $L_2 = F_2 L_2'$ for two finite sets $F_1, F_2 \subseteq \mathcal{L}$.

(Reduce$_2$) is needed because path variables always denote non-empty paths. We will view (Reduce$_1$) and (Reduce$_2$) as a single rule (Reduce).

To complete our rule system, we need the rules (LangDec$_\Lambda$), (DecClash), (Join) and (Empty). Since we will show that the rule system is terminating, we can replace (LangDec$_\Lambda$) by a simpler version, namely

$$\text{(LangDec}_{\text{dfun}}\text{)} \quad \frac{\{\mu \circ \nu \mathrel{\dot{\in}} L\} \cup \psi}{\{\mu \mathrel{\dot{\in}} P\} \cup \{\nu \mathrel{\dot{\in}} S\} \cup \psi} \qquad P \circ S \subseteq L, \quad (P, S) \in \text{dfun}(L)$$

L must contain a path w with $|w| > 1$.

Here dfun : $\wp(\mathcal{L}^+) \to \wp(\mathcal{L}^+) \times \wp(\mathcal{L}^+)$ is a decomposition function that assigns to each regular language L a finite set of decompositions. dfun is called **decomposition complete** if for every regular language L and every path $w = w_1 w_2 \in L$ there is a pair (P, S) in dfun(L) with $w_1 \in P$ and $w_2 \in S$. The complete set of rules is denoted $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$.

We illustrate the rule system $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$ using one of the pre-solved clauses listed in section 5.4.1 on page 126, namely

$$\phi = \left\{ \begin{array}{c} \mu_1 \mathrel{\dot{\amalg}} \mu_2,\ \mu_1 \mathrel{\dot{\amalg}} \mu_3,\ \mu_2 \mathrel{\dot{\amalg}} \mu_3,\ x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3, \\ \mu_1 \mathrel{\dot{\in}} (f \cup g),\ \mu_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \mu_3 \mathrel{\dot{\in}} h(f \cup g) \end{array} \right\}$$

Using the notation introduced in this section, we can write ϕ as

$$\dot{\amalg}(\{\mu_1, \mu_2, \mu_3\}) \cup \left\{ \begin{array}{c} x[\mu_1]y_1,\ x[\mu_2]y_2,\ x[\mu_3]y_3,\ \mu_1 \mathrel{\dot{\in}} (f \cup g), \\ \mu_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \mu_3 \mathrel{\dot{\in}} h(f \cup g) \end{array} \right\}$$

A successful $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$-derivation transforming ϕ into a solved clause appears in figure 5.5. In the derivation we assume a function dfun with

$$\text{dfun}(f^+ \cup h^+) = \{(f^+ \cup h^+,\ f^+ \cup h^+)\}$$

and

$$\text{dfun}(h(f \cup g)) = \{(h,\ f \cup g)\}.$$

After the explanation of the rule system we can commence the technical part. A clause ϕ is called **partitioned** if the set of divergence constraints of ϕ is of the form $\dot{\amalg}(A_1) \uplus \ldots \uplus \dot{\amalg}(A_n)$, where the A_i are disjoint.

Proposition 5.15 *There exists a decomposition function* dfun *that is decomposition complete.*

Proof. See proof of lemma 5.3 for the construction of such a function. □

Proposition 5.16 *Let ϕ be a pre-solved clause and let γ be a $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$-derivative of ϕ. Then γ is partitioned. Furthermore, for every pair of variables μ, ν such that $\mu \neq \nu$, $x[\mu]y \in \gamma$ and $x[\nu]z \in \gamma$ we have $\gamma \models \mu \mathrel{\dot{\amalg}} \nu$.*

Proposition 5.17 *For every partitioned clause ϕ the rule* (Reduce) = (Reduce$_1$) + (Reduce$_2$) *is $\mathcal{V}_{\mathcal{X}}(\phi)$-sound and globally $\mathcal{V}_{\mathcal{X}}(\phi)$-preserving. The rule* (Solv) *is $\alpha_{\mathcal{X}} \cup \alpha_{\mathcal{P}}$-sound and $\alpha_{\mathcal{X}} \cup \alpha_{\mathcal{P}}$-preserving. If* dfun *is decomposition complete, then* (LangDec$_{\text{dfun}}$) *is $\alpha_{\mathcal{X}} \cup \alpha_{\mathcal{P}}$-sound and $\alpha_{\mathcal{X}} \cup \alpha_{\mathcal{P}}$-preserving.*

$$\phi$$

$$\downarrow (\text{Reduce}_1) \text{ with } A_1 = \{\mu_2, \mu_3\}$$

$$\left\{ \begin{array}{c} \mu_1 \,\dot{\amalg}\, \nu,\ \mu_2' \,\dot{\amalg}_0\, \mu_3',\ x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3,\ \mu_1 \dot{\in} (f \cup g), \\ \nu \circ \mu_2' \dot{\in} (f^+ \cup h^+),\ \nu \circ \mu_3' \dot{\in} h(f \cup g) \end{array} \right\}$$

$$\downarrow 2\times (\text{LangDec}_{\text{dfun}})$$

$$\left\{ \begin{array}{c} \mu_1 \,\dot{\amalg}\, \nu,\ \mu_2' \,\dot{\amalg}_0\, \mu_3',\ x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3,\ \mu_1 \dot{\in} (f \cup g), \\ \nu \dot{\in} (f^+ \cup h^+),\ \mu_2' \dot{\in} (f^+ \cup h^+),\ \nu \dot{\in} h,\ \mu_3' \dot{\in} (f \cup g) \end{array} \right\}$$

$$\downarrow (\text{Solv})$$

$$\left\{ \begin{array}{c} \mu_1 \,\dot{\amalg}\, \nu,\ x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3,\ \mu_1 \dot{\in} (f \cup g), \\ \mu_2' \dot{\in} h\mathcal{L}^*,\ \mu_3' \dot{\in} g\mathcal{L}^*,\ \nu \dot{\in} (f^+ \cup h^+),\ \mu_2' \dot{\in} (f^+ \cup h^+),\ \nu \dot{\in} h,\ \mu_3' \dot{\in} (f \cup g) \end{array} \right\}$$

$$\downarrow 3\times (\text{Join})$$

$$\left\{ \begin{array}{c} \mu_1 \,\dot{\amalg}\, \nu,\ x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3,\ \mu_1 \dot{\in} (f \cup g), \\ \nu \dot{\in} h,\ \mu_2' \dot{\in} h^+,\ \mu_3' \dot{\in} g \end{array} \right\}$$

$$\downarrow (\text{Reduce}_2)$$

$$\left\{ \begin{array}{c} \mu_1 \,\dot{\amalg}_0\, \nu,\ x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3,\ \mu_1 \dot{\in} (f \cup g), \\ \nu \dot{\in} h,\ \mu_2' \dot{\in} f^+,\ \mu_3' \dot{\in} g \end{array} \right\}$$

$$\downarrow (\text{Solv})$$

$$\left\{ \begin{array}{c} x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3,\ \mu_1 \dot{\in} (f \cup g), \\ \mu_1 \dot{\in} f\mathcal{L}^*,\ \nu \dot{\in} h\mathcal{L}^*,\ \nu \dot{\in} h,\ \mu_2' \dot{\in} f^+,\ \mu_3' \dot{\in} g \end{array} \right\}$$

$$\downarrow 2\times (\text{Join})$$

$$\left\{ \begin{array}{c} x[\mu_1]y_1,\ x[\nu]z,\ z[\mu_2']y_2,\ z[\mu_3']y_3, \\ \mu_1 \dot{\in} f,\ \nu \dot{\in} h,\ \mu_2' \dot{\in} f^+,\ \mu_3' \dot{\in} g \end{array} \right\}$$

Figure 5.5: A successful $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$-derivation

Lemma 5.9 *$\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$ is terminating.*

Proof. For (Solv), (Join), (LangDec), (DecClash) and (Empty) it is trivial to see that there are no infinite derivations using only these rules. Furthermore, there are no derivations which use (Reduce) infinitely often, since during every application of (Reduce) at least one divergence constraint is removed (note that $|A_1| > 1$ is an application condition of (Reduce_1)). Hence, there are no infinite $\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$-derivations. □

Lemma 5.10 *Let ϕ be a pre-solved clause. If* dfun *is decomposition complete, then a $\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$-derivative of ϕ is $\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$-irreducible if and only if it is solved.*

Proof. Let γ be a $\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$-derivative of ϕ. We have to show that if γ is not solved, then one of the rules applies.

Condition 1 is satisfied by every $\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$-derivative of ϕ since ϕ is pre-solved and we do not add or change any sort restriction constraint. If one of the conditions 2 or 3 is not satisfied, then one of the rules (Join) or (Empty) will apply. Condition 6 is satisfied by every $\mathcal{R}^{\mathrm{Solv}}_{\mathrm{dfun}}$-derivative of ϕ by proposition 5.16. Now let's check the conditions 4, 5 and 7:

γ **contains a constraint** $\mu \circ \nu \mathrel{\dot\in} L$. ($\mathrm{LangDec}_{\mathrm{dfun}}$) or (DecClash) is applicable.

γ **contains a constraint** $\mu \mathrel{\dot{\amalg}} \nu$. By proposition 5.16 we know that in this case γ is of the form $\dot{\amalg}\,(A) \cup \psi$. Given the above we can assume that (Reduce) is applicable.

γ **contains a constraint** $\mu \mathrel{\dot{\amalg}_0} \nu$. Then either 7 is satisfied or (Solv) is applicable.

□

Lemma 5.11 *For every pre-solved clause ϕ there is a finite and effectively computable set of solved clauses Γ such that for every $\mathfrak{A}$*

$$[\![\phi]\!]^{\mathfrak{A}}_{\mathcal{V}_{\mathcal{X}(\phi)}} = \bigcup_{\gamma \in \Gamma} [\![\gamma]\!]^{\mathfrak{A}}_{\mathcal{V}_{\mathcal{X}(\phi)}}.$$

Proof. Follows from propositions 5.15, 5.16 and 5.17 and lemmas 5.9 and 5.10. □

Corollary 5.4 *Satisfiability of pre-solved clauses is decidable.*

Finally, we are able to combine both phases.

Theorem 5.4 *Satisfiability of prime clauses is decidable.*

Proof. Follows from the corollaries 5.3 and 5.4. □

Theorem 5.5 *The positive existential fragment of* $\mathrm{Th}(\mathfrak{T}_{\mathrm{RFT}})$ *and* $\mathrm{Th}(\mathfrak{R}_{\mathrm{RFT}})$ *is decidable.*

Proof. Follows from Propositions 5.3 and 5.5, which show that every RFT-clause can be translated into an RF-clause such that validity in the corresponding feature tree structures is preserved, Lemma 5.11, which shows that one can effectively transform a prime clause into an equivalent finite set of solved clauses, and Lemma 5.2, which shows that solved clauses are satisfiable in $\mathfrak{T}_{\mathrm{RF}}$ and $\mathfrak{R}_{\mathrm{RF}}$. □

Appendix A

Mathematical Preliminaries

We assume the reader to be familiar with first-order predicate logic. The signature of our first-order languages always contains the binary relation symbol $\doteq$, which is interpreted as equality, and the special symbols $\bot$ ("false") and $\top$ ("true"). $\bot$ (resp. $\top$) is the same as the empty disjunction (resp. conjunction). We assume an infinite alphabet of variables and adopt the conventions that x, y, z always denote variables, and X, Y always denote finite, possibly empty sets of variables.

Compound formulae are obtained as usual with the connectives $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$ and the quantifiers $\exists$ and $\forall$. We call atomic formulae also **constraints**.

We identify $\exists x \exists y \phi$ with $\exists y \exists x \phi$. If $X = \{x_1, \ldots, x_n\}$, we write $\exists X \phi$ for $\exists x_1 \ldots \exists x_n \phi$. If $X = \emptyset$, then $\exists X \phi$ stands for ϕ. Moreover, we use $\tilde{\exists}\phi$ [$\tilde{\forall}\phi$] to denote the existential [universal] closure of a formula ϕ, and $\exists! x \phi$ as an abbreviation for

$$\exists x \phi \wedge \forall x, y (\phi \wedge \phi[x \leftarrow y] \rightarrow x \doteq y).$$

For a set of variables $X = \{x_1, \ldots, x_n\}$ the quantifier $\exists! X \phi$ is defined as $\exists x_1 \ldots \exists x_n \phi$. Moreover, $\mathcal{V}(\phi)$ is taken to denote the set of all variables that occur free in a formula ϕ. The letters ϕ and ψ will always denote formulae. We use the conventions that to write $\phi(x_1, \ldots, x_n)$ if $x_1, \ldots, x_n$ is the set of free variables of ϕ.

We assume that the conjunction of formulae is an associative and commutative operation that has $\top$ as identity element. This means that we identify $\phi \wedge (\psi \wedge \theta)$ with $\theta \wedge (\psi \wedge \phi)$, and $\phi \wedge \top$ with ϕ (but not, for example, $xfy \wedge xfy$ with xfy). A conjunction of atomic formulae can thus be seen as the finite multiset of these formulae, where conjunction is multiset union, and $\top$ (the "empty conjunction") is the empty multiset. We will write $\psi \subseteq \phi$ (or $\psi \in \phi$, if ψ is an atomic formula) if there exists a formula ψ' such that $\psi \wedge \psi' = \phi$.

Structures and satisfaction of formulae are defined as usual. A valuation into a structure $\mathfrak{A}$ is a total function from the set of all variables into the universe $\mathbf{U}(\mathfrak{A})$ of $\mathfrak{A}$. A valuation α' into $\mathfrak{A}$ is called an **x-update** [**X-update**] of a valuation α into $\mathfrak{A}$ if α' and α agree everywhere except possibly on x [X]. We use $[\![\phi]\!]^{\mathfrak{A}}$ to denote the set of all valuations α such that $\mathfrak{A}, \alpha \models \phi$. We write $\phi \models \psi$ ("ϕ entails ψ") if $[\![\phi]\!]^{\mathfrak{A}} \subseteq [\![\psi]\!]^{\mathfrak{A}}$ for all structures $\mathfrak{A}$, and $\phi \models\!\models \psi$ ("ϕ is equivalent to ψ") if $[\![\phi]\!]^{\mathfrak{A}} = [\![\psi]\!]^{\mathfrak{A}}$ for all structures $\mathfrak{A}$.

A **theory** is a set of closed formulae. A **model** of a theory is a structure that satisfies every formulae of the theory. A formula ϕ is a **consequence of a theory** T ($T \models \phi$) if $\tilde{\forall}\phi$ is valid in every model of T. A formula ϕ **entails** a formula ψ in a theory T ($\phi \models_T \psi$) if $[\![\phi]\!]^{\mathfrak{A}} \subseteq [\![\psi]\!]^{\mathfrak{A}}$ for every model $\mathfrak{A}$ of T. Two formulae ϕ, ψ are **equivalent** in a theory T ($\phi \models\!\models_T \psi$) if $[\![\phi]\!]^{\mathfrak{A}} = [\![\psi]\!]^{\mathfrak{A}}$ for every model $\mathfrak{A}$ of T.

A theory T is **complete** if for every closed formula ϕ either ϕ or $\neg\phi$ is a consequence of T. The set of all sentences valid in some specific first-order structure $\mathfrak{A}$ is called the **theory of** $\mathfrak{A}$ (abbreviated by $\mathrm{Th}(\mathfrak{A})$). The theory of a single structure is always a complete theory. A theory is **decidable** if the set of its consequences is decidable. Since the consequences of a recursively enumerable theory are recursively enumerable (completeness of first-order deduction), a complete theory is decidable if and only if it is recursively enumerable.

Two first-order structures $\mathfrak{A}$, $\mathfrak{B}$ are **elementarily equivalent** if, for every first-order formula ϕ, ϕ is valid in $\mathfrak{A}$ if and only if ϕ is valid in $\mathfrak{B}$. Note that all models of a complete theory are elementarily equivalent.

A **transformation rule** is an ordered pair $\frac{\phi}{\psi}$ plus optional application conditions. Rule instances are defined as usual. With $\phi[x \leftarrow y]$ we denote the formula that is obtained from ϕ by replacing every occurrence of x with y. We say that $r = \frac{\phi}{\psi}$ is **applicable** to ϕ' if there is an instance of $\frac{\phi'}{\psi'}$ of r and the application conditions noted in the definition of r are satisfied. We write $\phi \rightarrow_r \gamma$ if r is applicable on ϕ and the result of the application is γ. For a set of transformation rules $\mathcal{R}$ we say $\phi \rightarrow_{\mathcal{R}} \gamma$ if there is an $r \in \mathcal{R}$ with $\phi \rightarrow_r \gamma$. ϕ is called **$\mathcal{R}$-irreducible** if no rule instance $r \in \mathcal{R}$ applies to ϕ. We say that a formula ϕ is $\mathcal{R}$-reducible if ϕ is not $\mathcal{R}$-irreducible. A sequence

$$\phi_0 \rightarrow_{r_0} \phi_1 \cdots \phi_i \rightarrow_{r_i} \phi_{i+1} \cdots$$

is called a **derivation**. A formula γ is called a **$\mathcal{R}$-derivative of** ϕ if there is a derivation from ϕ to γ that uses only rule instances of $\mathcal{R}$. Note that we ommit $\mathcal{R}$ if the set of rules is clear from the context.

A rule system $\mathcal{R}$ is called **terminating** if there are no inifinite derivations, and

quasi-terminating if for every formula ϕ, the set of derivable formulae is finite.

In the following, we recall some standard definitions for lattices and fixpoints of continous functions. They can be found in a standard text book on lattice theory (e.g., [DP90]).

Let P be a set partially ordered by $\preccurlyeq$. An element $n \in P$ is a **lower bound** for a subset N of P if

$$\forall m \in N : n \preccurlyeq m,$$

and an **upper bound** for N if $\forall m \in N : m \preccurlyeq n$. n is the **greatest lower bound of** N (denoted by $\sqcap N$) if n is a lower bound of N and in addition

$$\forall m \in M : \ m \text{ lower bound for } N \ \Rightarrow \ m \preccurlyeq n.$$

The **least upper bound** $\bigsqcup N$ of N is defined analogously. A partially ordered set P is called **complete lattice** if for all subsets N of P, both $\bigsqcup N$ and $\sqcap N$ is defined.

A non-empty subset D of a partially ordered set P is called **directed** if for every finite subset F of D there is an upper bound in D. A function $T : P \to P$ on an ordered set P is **continuous** if for every directed set D in P

$$T(\bigsqcup D) = \bigsqcup T(D) \qquad (:= \bigsqcup\{T(x) \mid x \in D\}).$$

An element $a \in M$ is called **fixpoint of** T if $T(a) = a$. A fixpoint a is a **least fixpoint of** T if $a \preccurlyeq b$ for all other fixpoints b of T. Similarly, we define **greatest fixpoint**. The least fixpoint is denoted by $\mathrm{lfp}(T)$, and the greatest fixpoint by $\mathrm{gfp}(T)$.

In order to describe fixpoints we need to define **ordinal powers** of the function T. We define

$$\begin{aligned} T \uparrow 0 &:= \bot \\ T \uparrow \alpha &:= T(T \uparrow (\alpha - 1)), \quad \text{if } \alpha \text{ is a successor ordinal} \\ T \uparrow \alpha &:= \bigsqcup\{T \uparrow (\beta) \mid \beta < \alpha\}, \quad \text{if } \alpha \text{ is a limes ordinal} \end{aligned}$$

and

$$\begin{aligned} T \downarrow 0 &:= \top \\ T \downarrow \alpha &:= T(T \downarrow (\alpha - 1)), \quad \text{if } \alpha \text{ is a successor ordinal} \\ T \downarrow \alpha &:= \bigsqcap\{T \downarrow (\beta) \mid \beta < \alpha\}, \quad \text{if } \alpha \text{ is a limes ordinal} \end{aligned}$$

The next two theorems states the well-known results for the fixpoints of continuous function over a complete lattice.

Theorem A.1 (Kleene) *Let P be a complete lattice and $T : P \to P$ be a continuous function. Then*

$$\mathrm{lfp}(T) \;=\; T \uparrow \omega,$$

and there is some ordinal β with

$$\mathrm{gfp}(T) \;=\; T \downarrow \beta' \;\; \textit{for all } \beta' \geq \beta.$$

Theorem A.2 (Tarksi) *Let P be a complete lattice and $T : P \to P$ be a continuous function. Then*

$$\mathrm{lfp}(T) \;=\; \bigsqcap\{m \mid T(m) \leq m\}$$

and

$$\mathrm{lfp}(T) \;=\; \bigsqcup\{m \mid m \leq T(m)\}.$$

The next definitions will be concerned with special properties of algebraic lattices. An element k of an CPO P is called **finite** if for every directed set D in P

$$k \preccurlyeq \bigsqcup D \Rightarrow k \preccurlyeq d \text{ for some } d \in D.$$

The set of finite elements is denoted by $F(P)$. A complete lattice P is called **algebraic**[1] if for every $a \in p$

$$a = \bigsqcup\{k \in F(P) \mid k \preccurlyeq a\}$$

If we consider an algebraic lattice, then continuity of a function has a special role, namely that the the function is complete determined by its finite approximations. The next proposition is a reformulation of Proposition 3.31 in [DP90, page 62].

Proposition A.1 *If P is an algebraic lattice, then $T : P \to P$ is a continuous function if and only if*

$$\textit{for all } e \in P\textit{: } T(e) = \bigsqcup\{T(k) \mid k \in F(P) \wedge k \preccurlyeq e\}.$$

[1]Normally, a lattice is called algebraic if every element $a \in P$ is the least upper bound of the compact elements that are smaller than a. But in complete lattices, the compact elements are just the finite ones.

Proof. For the first direction we know that $e = \bigsqcup\{k \mid k \in F(P) \wedge k \preccurlyeq e\}$, since P is algebraic. As $\{k \mid k \in F(P) \wedge k \preccurlyeq e\}$ is directed and T is continuous we know that

$$T(e) = T(\bigsqcup\{k \mid k \in F(P) \wedge k \preccurlyeq e\}) = \bigsqcup\{T(k) \mid k \in F(P) \wedge k \preccurlyeq e\}$$

For the other direction assume that $T(e) = \bigsqcup\{T(k) \mid k \in F(P) \wedge k \preccurlyeq e\}$ for every $e \in P$. We have to show that

$$\bigsqcup T(D) = T(\bigsqcup D)$$

for every directed set D. We know that $T(d) = \bigsqcup\{T(k) \mid k \in F(P) \wedge k \preccurlyeq d\}$ for every $d \in D$, which implies $\bigsqcup T(D) = \bigsqcup\{T(k) \mid k \in F(P) \wedge \exists d \in D : k \preccurlyeq d\}$. Now k finite implies that $k \preccurlyeq \bigsqcup D$ iff $k \preccurlyeq d$ for some $d \in D$. This shows that

$$\bigsqcup T(D) = \bigsqcup\{T(k) \mid k \in F(P) \wedge k \preccurlyeq \bigsqcup D\}.$$

Using our assumption we get $\bigsqcup\{T(k) \mid k \in F(P) \wedge k \preccurlyeq \bigsqcup D\} = T(\bigsqcup D)$. □

Bibliography

[AK86] Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. **Theoretical Computer Science**, 45:293–351, 1986.

[AK93] Hassan Aït-Kaci. An introduction to LIFE — programming with logic, inheritance, functions and equations. In Dale Miller, editor, **Proc. of the International Symposium on Logic Programming**, pages 52–68, Vancouver, October 1993. MIT Press.

[AKLN87] Hassan Aït-Kaci, Patrick Lincoln, and Roger Nasr. Le Fun: Logic, equations, and functions. In **Proceedings of the 1987 Symposium on Logic Programming**, pages 17–23. IEEE Computer Society, 1987.

[AKN86] Hassan Aït-Kaci and Roger Nasr. Login: A logic programming language with built-in inheritance. **The Journal of Logic Programming**, 3:185–215, 1986.

[AKN89] Hassan Aït-Kaci and Roger Nasr. Integrating logic and functional programming. **Lisp and Symbolic Computation**, 2:51–89, 1989.

[AKP93] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. **The Journal of Logic Programming**, 16:195–234, 1993.

[AKPG93] Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. Order-sorted feature theory unification. In Dale Miller, editor, **Proc. of the International Symposium on Logic Programming**, pages 506–524, Vancouver, October 1993. MIT Press.

[AKPS94] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. **Theoretical Computer Science**, 122(1–2):263–283, January 1994.

[Baa90] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Research Report RR-90-13, DFKI, Postfach 2080, 6750 Kaiserslautern, Germany, 1990.

[Baa91] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In **Proc. of the 12^{th} International Joint Conference on Artificial Intelligence**, pages 446–451, Sidney, 1991.

[Bac89] Rolf Backofen. Integration von Funktionen, Relationen und Typen beim Sprachentwurf. Teil II: Attributterme und Relationen. Master's thesis, Universität Erlangen-Nürnberg, 1989.

[Bac94] Rolf Backofen. Regular path expressions in feature logic. **Journal of Symbolic Computation**, 17:412–455, 1994.

[Bac95a] Rolf Backofen. A complete axiomatization of a theory with feature and arity constraints. **The Journal of Logic Programming**, 1995. To appear.

[Bac95b] Rolf Backofen. Type systems and definite equivalences. Forthcoming, 1995.

[BBN+93] Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, and Gert Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. **Journal of Logic, Language and Information**, 2:1–18, 1993.

[BEG90] Rolf Backofen, Lutz Euler, and Günther Görz. Towards the integration of functions, relations and types in an AI programming language. In Heinz Marburger, editor, **Proc. of the 14^{th} German Workshop on Artificial Intelligence**, volume 251 of **Informatik Fachberichte**, pages 297–306. Springer, Berlin, 1990.

[BK93] Rolf Backofen and Hans-Ulrich Krieger. The *TDL/UDiNe* system. In Backofen et al. [BKSU93], pages 67–74. DFKI Document D-93-27.

[BKSU93] Rolf Backofen, Hans-Ulrich Krieger, Stephen P. Spackman, and Hans Uszkoreit, editors. **Report of the EAGLES Workshop on Implemented Formalisms at DFKI, Saarbrücken**, 1993. DFKI Document D-93-27.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. **Cognitive Science**, 9(2):171–216, April 1985.

[BS93a] Rolf Backofen and Gert Smolka. A complete and recursive feature theory. In **Proc. of the 31th Annual Meeting of the Association for Computational Linguistics**, pages 193–200, Columbus, Ohio, 1993. Full version has appeared as Research Report RR-92-30, DFKI, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany, and will appear in Theoretical Computer Science.

[BS93b] Patrick Blackburn and Edith Spaan. A modal perspective on the computational complexity of attribute value grammar. **Journal of Logic, Language and Information**, 2:129–169, 1993.

[BT94] Rolf Backofen and Ralf Treinen. How to win a game with features. In Jean-Pierre Jouannaud, editor, **1st International Conference on Constraints in Computational Logics**, Lecture Notes in Computer Science, vol. 845, pages 320–335, München, Germany, 7–9 September 1994. Springer-Verlag.

[Car92] Bob Carpenter. **The Logic of Typed Feature Structures**, volume 32 of **Cambridge Tracts in Theoretical Computer Science**. Cambridge University Press, Cambridge, UK, 1992.

[Car94] Bob Carpenter. Ale: The attribute logic engine. user's guide, version 2.0. Technical report, Carnegie–Mellon University, Pittsburgh, PA 15213, 1994.

[CL89] Hubert Comon and Pierre Lescanne. Equational problems and disunification. **Journal of Symbolic Computation**, 7:371–425, 1989.

[Cla78] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, **Logic and Databases**, pages 293–322. Plenum Press, New York, NY, 1978.

[DD93] Jochen Dörre and Michael Dorna. CUF – a formalism for linguistic knowledge representation. In Dörre [Dör93a], pages 1–22. DYANA Deliverable R1.2.A.

[DE89] Jochen Dörre and Andreas Eisele. Determining consistency of feature terms with distributed disjunctions. In Dieter Metzinger, editor, **Proc.**

of the 13th German Workshop on Artificial Intelligence, volume 216 of **Informatik Fachberichte**, pages 270–279. Springer, Berlin, 1989.

[DE90] Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In **Proc. of the 13th International Conference on Computational Linguistics**, pages 100–105, Helsinki, Finland, 1990.

[DE91] Jochen Dörre and Andreas Eisele. A comprehensive unification-based grammar formalism. Deliverable R3.1.B, DYANA — ESPRIT Basic Research Action BR3175, Centre for Cognitive Science, University of Edinburgh, January 1991.

[Der87] Nachum Dershowitz. Termination of rewriting. **Journal of Symbolic Computation**, 3:69–116, 1987.

[Dör93a] Jochen Dörre, editor. **Computational Aspects of Constraint-Based Linguistic Descriptions**, 1993. DYANA Deliverable R1.2.A.

[Dör93b] Jochen Dörre. **Feature-Logik und Semiunifcation**. PhD thesis, Universität Stuttgart, 1993. In German.

[DP90] P. A. Davey and H. A. Priestley. **Introduction to Lattices and Order**. Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, England, 1990.

[DR89] Jochen Dörre and William C. Rounds. On subsumption and semiunification in feature algebras. IWBS report, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, 1989.

[DR92] Jochen Dörre and William C. Rounds. On subsumption and semiunification in feature algebras. **Journal of Symbolic Computation**, 13(4):441–461, April 1992.

[Eur94] European Community. **EAGLES Interim Report**, 1994. draft.

[EZ90a] Martin Emele and Rémi Zajac. A fixed-point semantics for feature type systems. In Stéphane Kaplan and Mitsuhiro Okada, editors, **Conditional and Typed Rewriting Systems, 2nd International Workshop**, LNCS 516, pages 383–388, Montreal, Canada, June 11–14, 1990. Springer-Verlag.

[EZ90b] Martin Emele and Rémi Zajac. Typed unification grammars. In **Proc. of the 13th International Conference on Computational Linguistics**, pages 293–298, Helsinki, Finland, 1990.

[HS88] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG-Report 53, IBM Deutschland GmbH, Stuttgart, October 1988.

[HSW93] Martin Henz, Gert Smolka, and Jörg Würtz. Oz—a programming language for multi-agent systems. In Ruzena Bajcsy, editor, **13th International Joint Conference on Artificial Intelligence**, volume 1, pages 404–409, Chambéry, France, 1993. Morgan Kaufmann Publishers.

[HSW95] Martin Henz, Gert Smolka, and Jörg Würtz. Object-oriented concurrent constraint programming in Oz. In V. Saraswat and P. Van Hentenryck, editors, **Principles and Practice of Constraint Programming**, chapter 2, pages 27–48. The MIT Press, Cambridge, MA, 1995. To appear.

[HU79] J.E. Hopcroft and J.D. Ullman. **Introduction to Automata Theory, Languages, and Computation**. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, Reading, MA, 1979.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In **Principles of Programming Languages**, pages 111–119, 1987.

[Joh88] Mark Johnson. **Attribute-Value Logic and the Theory of Grammar**, volume 16 of **CSLI Lecture Notes**. CSLI, 1988.

[Joh91] Mark Johnson. Features and formulae. **Computational Linguistics**, 17(2):131–151, 1991.

[Joh94] Mark Johnson. Computing with features as formulae. **Computational Linguistics**, 20(1):1–25, 1994.

[Kay79] Martin Kay. Functional grammar. In C. Chiarello et al., editor, **Proceedings of the 5th Annual Meeting of the Berkeley Linguistics Society**, pages 142–158, Berkeley, CA, 1979.

[KB82] Ronald M. Kaplan and Joan Bresnan. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, **The Mental Representation of Grammatical Relations**, pages 173–381. MIT Press, Cambridge (MA), 1982.

[KM88] R. M. Kaplan and J. T. Maxwell III. An algorithm for functional uncertainty. In **Proceedings of the 12th International Conference on Computational Linguistics**, pages 297–302, Budapest, Hungary, 1988.

[KR86] Robert T. Kasper and William C. Rounds. A logical semantics for feature structures. In **Proceedings of the 24th Annual Meeting of the ACL, Columbia University**, pages 257–265, New York, N.Y., 1986.

[KR90] Ropert T. Kasper and William C. Rounds. The logic of unification grammar. **Lingusitic and Philosophy**, 13:35–58, 1990.

[Kri95] Hans-Ulrich Krieger. *TDL*—**A Type Description Language for Constraint-Based Grammars. Foundations, Implementation, and Applications.** PhD thesis, Universität des Saarlandes, 1995. Forthcoming.

[KS94] Hans-Ulrich Krieger and Ulrich Schäfer. *TDL*—a type description language for constraint-based grammars. In **Proceedings of the 15th International Conference on Computational Linguistics, COLING-94**, pages 893–899, Kyoto, Japan, 1994.

[KZ88] Ronald M. Kaplan and Annie Zaenen. Long-distance dependencies, constituent structure, and functional uncertainty. In M. Baltin and A. Kroch, editors, **Alternative Conceptions of Phrase Structure**. University of Chicago Press, Chicago, 1988.

[Mah88] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In **Proceedings of the 3rd Annual Symposium on Logic in Computer Science**, pages 348–457, Edinburgh, Scotland, July 1988.

[MAK90] Richard Meyer and Hassan Ait-Kaci. Wild-life, a user manual. Technical Note 1, Digital Equipment Corporation, November 1990.

[Man93a] Suresh Manandhar. Cuf in context. In Dörre [Dör93a], pages 45–53. DYANA Deliverable R1.2.A.

[Man93b] Suresh K. Manandhar. **Relational Extensions to Feature Logic: Applications to Constraint Based Grammars**. PhD thesis, University of Edinburgh, 1993.

[MK89] John Maxwell and Roland Kaplan. An overview over disjunctive constraint satisfaction. In **Proceedings of the International Parsing Workshop 1989**, pages 18–27, 1989.

[Mos92] Lawrence S. Moss. Completeness theorems for logics of feature structures. In Y. N. Moschovakis, editor, **Logic from Computer Science**, pages 387–403, Berlin, Heidelberg, New York, 1992. Springer-Verlag.

[MP93] M. Andrew Moshier and Carl J. Pollard. The domain of set-valued feature structures. CLAUS Report 35, Universität des Saarlandes - Computerlinguistik, 1993.

[NP93] Joachim Niehren and Andreas Podelski. Feature automata and recognizable sets of feature trees. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, **TAPSOFT 93: Theory and Practice of Software Development**, Lecture Notes in Computer Science, vol. 668, pages 356–375, Orsay, France, 13–16 April 1993. Springer-Verlag.

[NS91] Bernhard Nebel and Gert Smolka. Attributive description formalisms ... and the rest of the world. In O. Herzog and C.-R. Rollinger, editors, **Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence**, volume 546 of **Lectures Notes in Artificial Intelligence**, pages 439–452. Springer-Verlag, Berlin, Germany, 1991.

[Per83] Fernando C.N. Pereira. Parsing as deduction. In **Proceedings of the 21th Annual Meeting of the Association for Computational Linguistics**, pages 137–144, Cambridge, MA, 1983.

[PM90] Carl J. Pollard and M. Drew Moshier. Unifying partial descriptions of sets. In P. Hanson, editor, **Information, Language, and Cognition. Vol. 1 of Vancouver Studies in Cognitive Science**, pages 285–322. University of British Columbia Press, 1990.

[Pod92] Andreas Podelski. A monoid approach to tree automata. In Maurice Nivat and Andreas Podelski, editors, **Tree Automata and Languages**, pages 41–56. North-Holland, 1992.

[Pol89] Carl Pollard. Sorts in unification-based grammar and what they mean. Unpublished manuscript, 1989.

[PS87] Carl Pollard and Ivan A. Sag. **Information-Based Syntax and Semantics. Vol. 1: Fundamentals**, volume 13 of **CSLI Lecture Notes**. Chicago Univ. Press, Chicago, 1987.

[PS94] Carl Pollard and Ivan A. Sag. **Head-Driven Phrase Structure Grammar**. Studies in Contemporary Linguistics. University of Chicago Press, Chicago, 1994.

[Rea91] Mike Reape. An introduction to the semantics of unification-based grammar formalisms. Deliverable R3.2.A, DYANA, Centre for Cognitive Science, University of Edinburgh, January 1991.

[RJ94] C. J. Rupp and Rod Johnson. On the portability of complex constraint-based grammars. In **Proc. of the 15^{th} International Conference on Computational Linguistics**, pages 900–905, Kyoto, Japan, 1994.

[RK86] William C. Rounds and Robert Kasper. A complete logical calculus for record structures representing linguistic information. In **Proc. of the Symposium on Logic in Computer Sciences**, pages 38–43, Cambridge (MA), 1986. IEEE Computer Society.

[Rou88] William Rounds. Set values for unification-based grammar formalisms and logic programming. Report CLSI-88-129, CSLI, Stanford (CA), June 1988.

[Sar91] Vijay A. Saraswat. Semantic foundation of concurrent constraint programming. In **Principles of Programming Languages**, pages 333–352, 1991.

[Shi86] Stuart M. Shieber. **An Introduction to Unification-Based Approaches to Grammar**, volume 4 of **CSLI Lecture Notes**. Stanford University, Stanford (CA), 1986.

[Shi89] Stuart M. Shieber. Parsing and type inference for natural and computer languages. Technical Note 460, SRI International, Menlo Park, MA, March 1989.

[Shi92] Stuart M. Shieber. **Constraint-Based Grammar Formalisms**. MIT - Press, Cambridge, Massachusetts - London, 1992.

[Smo88] Gert Smolka. A feature logic with subsorts. LILOG-Report 33, IWBS, IBM Deutschland, Stuttgart, May 1988.

[Smo92] Gert Smolka. Feature constraint logics for unification grammars. **Journal of Logic Programming**, 12:51–87, 1992.

[Smo93] Gert Smolka. Logische Programmierung. Lecture Notes (in German), 1993.

[Smo94a] Gert Smolka. A calculus for higher-order concurrent constraint programming with deep guards. Research Report RR-94-03, DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, February 1994.

[Smo94b] Gert Smolka. The definition of Kernel Oz. DFKI Oz documentation series, DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[Smo94c] Gert Smolka. A foundation for higher-order concurrent constraint programming. In Jean-Pierre Jouannaud, editor, **1st International Conference on Constraints in Computational Logics**, Lecture Notes in Computer Science, vol. 845, pages 50–72, München, Germany, 7–9 September 1994. Springer-Verlag.

[Smo94d] Gert Smolka. An Oz primer. DFKI Oz documentation series, DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[SR90] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In **Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages**, pages 232–245, San Francisco, CA, January 1990.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. **Artificial Intelligence**, 48:1–26, 1991.

[ST94] Gert Smolka and Ralf Treinen. Records for logic programming. **The Journal of Logic Programming**, 18(3):229–258, April 1994.

[SUP+83] Stuart Shieber, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and Mabry Tyson. The formalism and implementation of PATR-II. In Barbara J. Grosz and Mark E. Stickel, editors, **Research on Interactive Acquisition and Use of Knowledge**, pages 39–79. AI Center, SRI International, Menlo Park, Cal., 1983.

[Tre93] Ralf Treinen. Feature constraints with first-class features. In Andrzej M. Borzyszkowski and Stefan Sokołowski, editors, **Mathematical Foundations of Computer Science**, Lecture Notes in Computer Science, vol. 711, pages 734–743, Gdańsk, Poland, 30 August–3 September 1993. Springer-Verlag.

[Ven87] K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebra. **Journal of the Association for Computing Machinery**, 34(2):492–510, April 1987.

[VSJ88] K. Vijay-Shanker and A. K. Joshi. Feature structure based tree adjoining grammars. In **Proc. of the 12^{th} International Conference on Computational Linguistics**, pages 714–719, Budapest, Hungary, 1988.

[Zaj92] Rémi Zajac. Inheritance and constraint-based grammar formalisms. **Computational Linguistics**, 18(2):159–182, 1992.

Index to Symbols

List of Theorems, Lemmas etc.

Subject Index