Parallelism Constraints in Underspecified Semantics

Katrin Erk

Dissertation

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

Saarbrücken 2002

Abstract

This thesis studies the *Constraint Language for Lambda Structures* (CLLS), which is interpreted over lambda terms represented as tree-like structures. Our main focus is on the processing of *parallelism constraints*, a construct of CLLS. A parallelism constraint states that two pieces of a tree have the same structure.

We present a sound and complete semi-decision procedure for parallelism constraints, which tests satisfiability and makes structural isomorphism explicit. This procedure is extended to a semi-decision procedure for CLLS.

We discuss two applications of CLLS. First, CLLS has been developed as a formalism for *underspecified natural language semantics*. In this context, parallelism constraints are used for modeling parallelism phenomena. Second, we consider *underspecified beta reduction*, which is beta reduction on partial descriptions of lambda terms. For these application areas, we present extensions both to the language CLLS and to the semi-decision procedure.

Kurzzusammenfassung

Diese Dissertation untersucht die *Constraint Language for Lambda Structures* (CLLS), eine Constraint-Sprache zur Beschreibung von Lambda-Termen in einer baumähnlichen Repräsentation. Der Schwerpunkt der Arbeit liegt auf Verfahren für *Parallelismus-Constraints*, ein Konstrukt der Sprache CLLS. Ein Parallelismus-Constraint besagt, dass zwei Bereiche eines Baumes dieselbe Struktur haben.

Wir stellen ein korrektes und vollständiges Semi-Entscheidungsverfahren für Parallelismus-Constraints vor, das Erfüllbarkeit feststellt und Strukturgleichheit explizit macht. Dies Verfahren wird zu einem Semi-Entscheidungsverfahren für CLLS erweitert.

Wir diskutieren zwei Anwendungen der Sprache CLLS. Zum einen ist CLLS als Beschreibungsformalismus für *unterspezifizierte natürlichsprachliche Semantik* entwickelt worden. In diesem Zusammenhang werden Parallelismus-Constraints zur Modellierung von Parallelismus-Phänomenen verwendet. Zum anderen betrachten wir *unterspezifizierte Beta-Reduktion*, Beta-Reduktion auf partiellen Beschreibungen von Lambda-Termen. Für diese Anwendungsgebiete stellen wir Erweiterungen sowohl der Sprache CLLS als auch des Semi-Entscheidungsverfahrens vor.

Extended abstract

This thesis studies a constraint language that is interpreted over lambda terms represented as tree-like structures. The language has been developed in the context of natural language semantics, where it is used for an underspecified representation of meaning. Our main focus is on determining satisfiability of these constraints, in particular for a construct of this constraint language that can be used to model parallelism phenomena.

The constraint language that we study is the Constraint Language for Lambda Structures (CLLS), and the language construct that we focus on is the *parallelism constraint*. Parallelism constraints are formulas that state that two pieces of a tree have the same structure. The central issue of this thesis is *the processing of parallelism constraints.*

An important characteristic of CLLS is that it allows for statements of parallelism on a description that leaves open the relative position of tree nodes. We consider two related applications of parallelism constraints, which both centrally make use of this property. First, the language CLLS has been developed as a formalism for *underspecified natural language semantics*. In this framework, parallelism constraints have been used to model the *parallelism phenomenon*, which is ubiquitous in linguistics. Prominent examples of parallelism are elliptical constructions like "John sleeps, and Mary does, too". The second application is *underspecified beta reduction*. The idea is to perform beta reduction on the partial descriptions of lambda terms, rather than on the terms themselves.

This thesis consists of two parts. The first part presents the central contribution: a procedure for solving parallelism constraints. The second part of the thesis studies questions of the practical applicability of the formalism as well as the procedure.

Solving parallelism constraints. We present a sound and complete semi-decision procedure for parallelism constraints and extend it to a semi-decision procedure for CLLS. It has the following properties:

- The procedure is stated in terms of high-level transformation rules.
- The procedure computes constraints from which models can be directly read off. In particular, it computes all *minimal* constraints with this property for a given input constraint. During the computation, structural isomorphism imposed by parallelism constraints is made explicit.
- The procedure terminates on the classes of cases relevant for the applications.
- The central concept of the procedure is *correspondence*: In accordance with the node-centered perspective on trees that CLLS adopts, the procedure relates nodes that occupy matching positions in the two parallel tree pieces.

Applicability. In the context of the two applications named above, underspecified natural language semantics and underspecified beta reduction, the thesis focuses on two issues:

Empirical adequacy: Is the formal language adequate for modeling the phenomena arising both in underspecified beta reduction and in underspecified semantics? We present two extensions to the standard CLLS formulation of parallelism constraints, which are of use both for underspecified beta reduction and for modeling ellipsis.

Underspecification: In solving parallelism constraints, the above procedure makes the relative position of nodes explicit. However it may be desirable to maintain underspecification as far as possible while making structural isomorphism explicit. We discuss a procedure which, exploiting knowledge about the relative positions of parallel tree pieces in underspecified beta reduction, can avoid disambiguation in many cases.

For both issues, the notion of correspondence again proves essential.

Ausführliche Zusammenfassung

Diese Dissertation untersucht eine Constraint-Sprache zur Beschreibung von Lambda-Termen in einer baumähnlichen Repräsentation. Die Sprache wurde als Modellierungsformalismus in der natürlichsprachliche Semantik entwickelt und wird für eine unterspezifizierte Beschreibung von Bedeutung verwendet. Unser Schwerpunkt liegt auf Erfüllbarkeitstests für diese Constraints, insbesondere für ein Sprachkonstrukt, das zur Modellierung von Parallelismus-Phänomenen verwendet werden kann.

Die Constraint-Sprache, die wir untersuchen, ist die *Constraint Language for Lambda Structures* (CLLS), und das Sprachkonstrukt, das im Mittelpunkt dieser Arbeit steht, ist der *Parallelismus-Constraint.* Ein Parallelismus-Constraint ist eine Formel, die besagt, dass zwei Bereiche eines Baumes dieselbe Struktur haben. Das Hauptthema dieser Dissertation ist ein *Verfahren für Parallelismus-Constraints.*

Ein wichtiger Punkt an CLLS ist, dass Parallelismus formuliert wird im Rahmen von partiellen Beschreibungen, die die relativen Positionen von Baumknoten offenlassen. Wir betrachten zwei Anwendungen für Parallelismus-Constraints, die sich beide zentral auf diese Eigenschaft von CLLS stützen. Zum einen wurde die Sprache CLLS als Formalismus für *unterspezifizierte natürlichsprachliche Semantik* entwickelt. Parallelismus-Constraints werden hier zur Modellierung des Phänomens Parallelismus verwendet. Typische Beispiele dieses verbreiteten Phänomens sind elliptische Konstruktionen wie z.B. „Hans schläft, und Maria auch." Die andere Anwendung ist die *unterspezifizierte Beta-Reduktion.* Hier geht es darum, Beta-Reduktion auf partielle Beschreibungen von Lambda-Termen anzuwenden statt auf die Terme selbst.

Die Dissertation besteht aus zwei Teilen. Der erste Teil stellt den Hauptbeitrag der Arbeit dar: ein Verfahren zum Lösen von Parallelismus-Constraints. Der zweite Teil der Arbeit beschäftigt sich mit Fragen der praktischen Anwendbarkeit des Formalismus sowie des Verfahrens.

Das Lösen von Parallelismus-Constraints. Wir stellen ein korrektes und vollständiges Semi-Entscheidungsverfahren für Parallelismus-Constraints vor, das wir zu einem Semi-Entscheidungsverfahren für CLLS erweitern. Es hat die folgenden Eigenschaften:

- Das Verfahren ist in Form von Transformationsregeln auf Constraints formuliert.
- Das Verfahren berechnet Constraints, von denen Modelle direkt abgelesen werden können. Für einen gegebenen Eingabe-Constraint berechnet es alle *minimalen* Constraints mit dieser Eigenschaft. Die Berechnung macht die Strukturgleichheit, die ein Parallelismus-Constraint beschreibt, explizit.
- Für die Klasse von Fällen, die für die Anwendungen relevant ist, terminiert das Verfahren.
- Das zentrale Konzept des Verfahrens ist *Korrespondenz*: Es werden Paare von

Knoten in Beziehung gesetzt, die in den beiden parallelen Baum-Bereichen dieselbe Position einnehmen. Das Konzept von Korrespondenz folgt damit der Knoten-zentrierten Perspektive auf Bäume, die CLLS einnimmt.

Anwendbarkeit. Im Zusammenhang mit den zwei oben genannten Anwendungen, unterspezifizierter natürlichsprachlicher Semantik und unterspezifizierter Beta-Reduktion, betrachten wir zwei Fragen:

Empirische Adäquatheit: Werden die Phänomene, die in den Anwendungen auftreten, von dem Formalismus adäquat modelliert? Wir stellen zwei Generalisierungen von Parallelismus-Constraints vor, die sowohl für die unterspezifizierte Beta-Reduktion als auch für eine Modellierung von Ellipsen-Phänomenen von Nutzen sind.

Unterspezifikation: Das oben genannte Verfahren macht beim Lösen von Parallelismus-Constraints relative Positionen von Baumknoten zu einem gewissen Grad explizit. In der Anwendung auf unterspezifizierte Beta-Reduktion ist es aber wünschenswert, Unterspezifikation so weit als möglich aufrechtzuerhalten und gleichzeitig Strukturgleichheit explizit zu machen. Wir stellen ein Verfahren vor, das Wissen über die relativen Positionen von Baum-Bereichen in der unterspezifizierten Beta-Reduktion ausnutzt und so in vielen Fällen Disambiguierung vermeiden kann.

Für beide Fragen erweist sich das Konzept der Korrespondenz als essentiell.

Acknowledgements

I am very grateful to my supervisor Gert Smolka, and also my second supervisor Manfred Pinkal, for opening up a fascinating area of research to me, for all those stimulating discussions and for all they have taught me – not only about tree-description formalisms and underspecified semantics, but also about clarity in writing, about asking the big questions and painting the bigger picture. I am happy that I could work with you.

A very big thank you to my “inofficial supervisor” Joachim Niehren. He very patiently helped me when I was new to the field of my thesis. He is a very generous person, generous both with his vast knowledge on constraints and with his vast supply of wonderful wines. I learned a lot from him, and I highly appreciate his zeal in making thoughts still clearer and definitions still more concise.

I would like to thank some wonderful people I had the luck to work with and who also helped me a lot with their comments on my thesis: Alexander Koller, who infected me with his endless enthusiasm for semantics, Markus Egg, who, I think, knows everything about linguistics (and fine wine), Mateu Villaret, Roy Dyckhoff, and especially Geert-Jan Kruijff, who helped me stay focused by asking me again and again how I was getting along with my “D-thing”, even though he sometimes got a grumpy answer, and who could always make me laugh with stories of the time when he was writing his Ph.D. thesis. For many interesting discussions and thought-provoking questions, many thanks to Ralf Treinen, Sophie Tison, Marc Tommasi and Isabelle Tellier.

Furthermore, many thanks to my colleagues at the programming systems lab, especially my office mate Andreas Rossberg, for the great working environment. And many thanks to the Coli people for being fun and for always being full of great new ideas.

I would like to thank the Deutsche Forschungsgemeinschaft (DFG, German Science Foundation) for funding me by a doctoral scholarship within the Graduiertenkolleg Kognitionswissenschaft, Saarbrücken.

Finally, I would like to thank my parents and, more than all others, Steffen – actually, I cannot thank them enough for all the love and support they have given me.

Thank you all who helped me make it.

Katrin Erk

Contents

1 Introduction **1**

1.1 Constraints . . . 2

1.2 Tree Description Languages . . . 3

1.3 Natural Language Semantics, Underspecification, and Parallelism Phenomena . . . 6

1.4 The Constraint Language for Lambda Structures (CLLS) . . . 10

1.5 A Procedure for CLLS . . . 15

1.6 Contributions . . . 19

1.7 Plan of this Thesis . . . 20

I A Procedure for CLLS Constraints **23**

2 CLLS **25**

2.1 Lambda Structures . . . 25

2.2 The Constraint Language for Lambda Structures . . . 30

2.3 Modeling Scope, Ellipsis, and Anaphora with CLLS . . . 33

2.4 Related Formalisms . . . 43

2.5 Related Modeling Approaches . . . 49

2.6 Summary . . . 54

3 Solving Dominance Constraints **57**

3.1 A Solver for Dominance Constraints: $\mathcal{P}_{\mathrm{d}}$ . . . 57

3.2 Some Properties of the Algorithm: Soundness, Termination, Shape of Saturations . . . 62

3.3 Satisfiability of Saturated Constraints 64
3.4 Completeness 69
3.5 Recapitulation: Properties of the Algorithm $\mathcal{P}_\mathrm{d}$ 70
3.6 Related Approaches 71
3.7 Summary 72

4 Solving Parallelism Constraints 75

4.1 A Semi-Decision Procedure for Parallelism Constraints: $\mathcal{P}_\mathrm{p}$ 75
4.2 Some Properties of the Procedure: Soundness, Nontermination, Control, Saturations 88
4.3 Satisfiability of Saturated Constraints 91
4.4 A Partial Order on $\mathcal{C}_\mathrm{p}$ Constraints 102
4.5 Completeness of Procedure $\mathcal{P}_\mathrm{p}$ 110
4.6 Recapitulation: Properties of the Procedure $\mathcal{P}_\mathrm{p}$ 113
4.7 Related Work 114
4.8 The Search Tree that $\mathcal{P}_\mathrm{p}$ Explores 115
4.9 Summary 116

5 Solving CLLS Constraints 119

5.1 A Semi-Decision Procedure for CLLS: $\mathcal{P}$ 119
5.2 Some Properties of the Procedure: Soundness, Saturations 124
5.3 Satisfiability of Saturated Constraints 125
5.4 Completeness of Procedure $\mathcal{P}$ 129
5.5 Recapitulation: Properties of the Procedure $\mathcal{P}$ 130
5.6 All Rules of $\mathcal{P}$ Collected 131
5.7 Summary 132

II Applying Parallelism Constraints 135

6 Underspecified Beta Reduction 137

6.1 The Problem of Underspecified Beta Reduction 138

6.2 Beta Reduction and Group Parallelism 141

6.3 Extending the Semi-Decision Procedure for CLLS to Group Parallelism and Inverse Lambda Binding Literals 148

6.4 Disambiguating Less: A Second Procedure for a Single Beta Reduction Step 161

6.5 Underspecified Beta Reduction 170

6.6 Discussion: Nonlinear Redexes 171

6.7 Summary 173

7 Modeling Ellipsis with Group Parallelism and Jigsaw Parallelism 175

7.1 The Phenomenon 175

7.2 Modeling Ellipsis with Group Parallelism Constraints 178

7.3 Jigsaw Parallelism 180

7.4 Modeling Ellipsis with Jigsaw Parallelism Constraints 192

7.5 A Look at Other Formalisms 192

7.6 Discussion 194

7.7 Summary 196

8 Modeling Ellipsis: A Comparison of Approaches 197

8.1 What is the Nature of Ellipsis? 197

8.2 What Problems Need to be Solved in Connection with Ellipsis? 198

8.3 At Which Level of Linguistic Structure should an Ellipsis Theory be Situated? 198

8.4 Approaches to Modeling Ellipsis 201

8.5 A Tentative Assessment of the CLLS Approach to Modeling Ellipsis . . . 203

8.6 Summary 205

9 Outlook 207

9.1 A Decidable Fragment of CLLS 208

9.2 Processing CLLS Constraints 209

10 Conclusion **211**

10.1 A Procedure for CLLS Constraints 211

10.2 Applying Parallelism Constraints 212

Index **225**

Chapter 1

Introduction

This thesis studies a constraint language that is interpreted over lambda terms represented as tree-like structures. The language has been developed in the context of natural language semantics, where it is used for an underspecified representation of meaning. Our main focus is on determining satisfiability of these constraints, in particular for a construct of this constraint language that can be used to model parallelism phenomena.

The constraint language that we study is the Constraint Language for Lambda Structures (CLLS) [42], a logical language interpreted over lambda structures, tree-like structures that represent lambda terms. The language construct that we focus on is the *parallelism constraint*. Parallelism constraints are formulas that state that two pieces of a tree, called *segments*, have the same structure. The central issue of this thesis is a *procedure for parallelism constraints*.

The main question that we address is:

> Given a partial description of a tree, including statements of structural isomorphism between some tree segments, how can we test the satisfiability of the description and at the same time make the structural isomorphism explicit?

We examine the formalism as well as the question of processing with respect to two application areas. On the one hand, the language CLLS has been used for an *underspecified account of natural language semantics*. On the other hand, we study the question of *underspecified beta reduction*, i.e. beta reduction on partial descriptions of lambda terms.

The main result that we report in this thesis is a sound and complete semi-decision procedure for CLLS. It is a high-level, rule-based procedure that computes all *minimal* result constraints for a given input constraint. We introduce extensions to the formalism of parallelism constraints that are sufficient for modeling the phenomena occurring in the two application areas, and we present an extension to the CLLS procedure geared at application in underspecified beta reduction.

In the next few sections, we establish the context in which the language CLLS is situated. There are three areas to be mentioned: CLLS is a *constraint language*; CLLS is a *tree description language*; and it was developed to model some phenomena of *natural language*

semantics. Next we sketch the language CLLS and its application to underspecified semantics as well as underspecified beta reduction. Then we discuss the question of processing CLLS: we sketch the main problems and the techniques we will use to solve them. Finally, we summarize the results we present in this thesis, and we give an overview of its organization.

1.1 Constraints

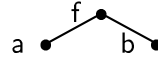
Constraints are formulas that describe sets of data from a specific domain, like finite domains of integers, or (in our case) finite trees (see e.g. the overview article by Comon et al. [24]). A constraint system comprises a constraint language and a class of interpretations, which is typically given either by a theory, or by a structure in which the formulas are interpreted. Some definitions also include a *constraint solver*, an algorithm that tests satisfiability.

Constraint systems have the following interesting properties:

- Constraints give a compact and simple *implicit* description of possibly infinite sets.
- Constraints provide a clean separation between the description (in the constraint language) and the computation (by the constraint solver).
- Constraint solvers can exploit knowledge about the domain, the class of interpretations.
- Constraint solvers compute with data structures that are only partially known and given only implicitly by the constraint formulas.
- Computing with constraints is often seen as simplification: The information about the variables, which is given only implicitly in the formulas, is made as explicit as possible.
- Constraints are well suited to handling *partial* descriptions that are augmented *incrementally*. Adding more constraints means decreasing the size of the set of data described.

The constraint language that we will study in this thesis is interpreted over *lambda structures*, finite constructor trees augmented by a construct for lambda binding. The main problem that we will be dealing with is: How can we test a constraint of the Constraint Language for Lambda Structures for satisfiability, while making the data described more explicit?

1.2 Tree Description Languages



There are two standard tree representations that we will use very often. First, ground terms are (constructor) trees. For example the ground term $f(a, b)$ is the tree drawn to the right. Second, nodes of a tree can be addressed by, and encoded as, their paths from the root down, where a path is a word of edge labels. For example in the tree to the right, ε is the root, its left child is 1, and its right child is 2. The edge labels can be just numbers, as in the example we have just seen, or they can be arbitrary symbols, as long as different outgoing edges of a node are labeled by different symbols. With this encoding of nodes as paths, a tree can be described simply by the set of its nodes together with a node labeling function.

There is a large body of research on tree description languages, both in computer science and in computational linguistics. One way of structuring it is by the languages' perspective on trees. In the terminology of Blackburn, Meyer-Viol and De Rijke [10], a language that takes an *external* perspective describes relations between *trees*, while taking the *internal* perspective means talking about relations between *nodes* of a single tree. The difference in perspective usually implicates a difference in the notion of identity: In external perspective languages, identity usually means identity of *structure*, while in internal perspective languages identity usually is identity of *occurrence*.

As an example for the external perspective, suppose that we have variables x, y standing for *trees*, then a statement like $x = f(y, y)$ could be used to say that the root of x is labeled f, and that x has two identical subtrees – with the notion of identity of *structure*, the two occurrences of y mean that we have two y-shaped trees. As an example for the internal perspective, suppose x, y, z are variables standing for *nodes*, then the expression $x{:}f(y, z)$ could be used to state that the node x is labeled f and has y, z as children. Note that with identity meaning the same *occurrence*, an expression like $x{:}f(y, y)$ cannot describe a tree, because it would mean that the node y occurs as *two distinct* children of the node x.

In the following, we sketch three classes of tree description languages. We focus mainly on the internal perspective, since this is the one that we will adopt, and we pay special attention to two constructs: the ancestor relation between nodes (*dominance*), and relations expressing structural isomorphism between parts of trees (*parallelism*).

(W)SkS. SkS, the second-order monadic logic with k successors, and its weak variant WSkS are among the most expressive decidable logics. The decidability of (W)SkS is due to famous results by Doner, Thatcher and Wright and Rabin [113, 32, 98]. Doner, Thatcher and Wright linked definability in WSkS to recognizability by finite tree automata. Rabin showed that definability in SkS coincides with recognizability by Rabin tree automata over infinite trees.

Terms of SkS are formed from the constant ε, first-order variables $x, y, z, \ldots$, and right concatenation with (unary function symbols) $1, \ldots, k$. Atomic formulas are equations and inequations $t_1 \leq t_2$ between terms, and expressions "$t \in X$" for terms t and second-order

variables X. Formulas are built using atomic formulas, all the usual connectives, and existential and universal quantification over both first-order and second-order variables. While second-order variables range over arbitrary sets in SkS, they are restricted to ranging over finite sets in WSkS.

There are several ways of encoding sets of trees in (W)SkS [78, 23]. They share the same basic idea: The terms denote tree nodes, concatenation ti stands for the i-th child of the node denoted by t, and $\leq$ is interpreted as the prefix relation: $t_1 \leq t_2$ means that the node denoted by t_1 dominates the node denoted by t_2.

While a huge number of properties of and relations between sets of nodes can be expressed in (W)SkS (e.g. union, intersection, prefix-closedness), there are interesting exceptions, relations that would be easy to state from an external perspective on trees. One example is the statement that a certain tree has two identical subtrees at depth one. We have already seen an "external perspective" formulation of such a situation, the equation $x = f(y, y)$, where x, y stand for *trees*. This statement cannot be expressed in SkS.

Feature description languages. Feature description languages [103] describe feature graphs, which can be regarded as logical descriptions of records. Roughly, a feature graph is a directed graph with node and edge labels. The edge labels are called *features*; different outgoing edges of a node are always labeled by different features. So if we fix one "current node" in the feature graph, then we can address another (reachable) node by the word of features on the path to it. Feature trees are just a special case of feature graphs. And the tree notation that encodes nodes as words over the set of natural numbers can be regarded as a special case of feature trees in which the features are just numbers. The main difference between feature trees and constructor trees (ground terms) is that in a feature tree each child of a node can be addressed individually via its feature.

Feature descriptions have their origins in phonology [20] and became a widespread formalism for linguistic theories in the 70s in *unification grammars* [68, 65], which comprise some of the most widely used grammars in theoretical linguistics, like LFG and HPSG. These formalisms are called *constraint-based.* Of the properties of constraint systems that we have listed above, those that are relevant for these formalisms are: they provide declarative descriptions, and they work with partial descriptions that can be augmented incrementally.

An influential language for feature graphs is the one by Kasper and Rounds [66, 67]. The language is interpreted on one distinguished "current node" of a feature graph, and its most important constructs can express the following things: the current node has a certain node label; two paths (= feature words) leading off the current node end at the same node; and some formula φ holds at the node that we reach from the current one by the label ℓ. This last statement is expressed by a formula $\ell : \varphi$. Smolka [110] proposes a constraint system over feature graphs and studies constraint solvers. He is the first to use plain first-order logic to describe feature graphs, and he introduces the more general notion of *feature algebra.* Building on this work, Backofen and Smolka [7] introduce a first-order feature theory FT that is complete and decidable; Aït-Kaci,

Podelski and Smolka [1] further explore the same feature tree descriptions, presenting a constraint system and a simplification system.

Especially interesting for our purposes are logics that pursue the modal aspect of the Kasper-Rounds logic: They view the feature graph as the reachability relation and provide modal operators for traveling in the graph. Whereas in the Kasper-Rounds logic we have formulas prefixed by features, expressions of the form $\ell : \varphi$, Blackburn [8] turns such feature prefixes into modal operators $\langle \ell \rangle$. Blackburn and Meyer-Viol [9] and similarly Kracht [79] go one step further. They work with feature trees and, abstracting over the features labeling a path, they use a modal $\downarrow^* \varphi$ to state that φ is true at some node dominated by the current node. Likewise $\uparrow^* \varphi$ states that φ is true at some ancestor of the current node. So again, as in SkS, the dominance relation plays an important role.

Context unification. Context unification [22] is a variant of linear second order unification [84, 95, 85]. A context unification (CU) equation system is a conjunction of equations between terms. These terms may contain first-order variables standing for trees, and context variables standing for *contexts*. Intuitively, a context is a tree with a hole, which can be written as a term with a constant $\bullet$, the hole, that occurs exactly once. A context can also be seen as a *context function* from trees to trees, for example the context $f(\bullet, b)$ would map the tree $g(a)$ to the tree $f(g(a), b)$: It just plugs $g(a)$ into the hole.

An example for a CU equation system is

$$f(C(a), b) = C(f(a, b)).$$

A solution of this formula is a mapping of context variables to contexts that gives us the same tree on both sides of the equation. Figure 1.1 shows one such solution, which maps C to $f(\bullet, b)$. The occurrences of this context are shown in the picture as shaded areas.

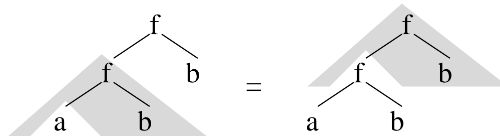


Figure 1.1: One solution of the CU equation system $f(C(a), b) = C(f(a, b))$

So this language adopts the external perspective on trees, where different occurrences of the same context variable stand for structurally isomorphic contexts, but not the same *occurrence* of the context. That means that CU can express structural isomorphism in trees by using the same variable repeatedly. As we will see, this is quite similar to the notion of *parallelism* that we have in CLLS, and the contexts that we have here are almost the same as the segments that parallelism in CLLS is about.

Instead of describing CU as a restriction of second-order unification, we can also say that it is a generalization of *string unification* from words to trees. String unification is the problem of solving word equations. For example, all solutions of the equation $ax = xa$

map x to a word in a^*. String unification has been discovered and studied under several names and in several research contexts [6]. Makanin was the first to present an algorithm for string unification [86]. So context unification lies between string unification, which is decidable, and second-order unification, which is not [54]. But the decidability of context unification is still an open problem [104] – which is also interesting for our purposes, as context unification and the parallelism constraints of CLLS are equally expressive [93, 92].

1.3 Natural Language Semantics, Underspecification, and Parallelism Phenomena

The language CLLS was developed to model phenomena of natural language semantics: certain structural ambiguities and their interaction with parallelism phenomena. In this section, we proceed in three steps: First we briefly talk about formal semantics and its use of lambda calculus. Then we introduce the concept of *underspecification* and its application to structural ambiguities with respect to *quantifier scope*. Finally we discuss parallelism phenomena and their interaction with quantifier scope ambiguity.

1.3.1 Lambda Calculus and Natural Language Semantics

Formal semantics describes those aspects of the semantic structure in natural language that can be captured with the tools of mathematical logic. An important work in this context is Montague Grammar [89], which is still often used as a basis for semantic construction. The aim of Montague Grammar is to show the logical structure of natural language and describe it with the means of universal algebra and mathematical logic. The meaning of a sentence is constructed *compositionally* by assembling simpler meaning units, according to the Fregean principle that the meaning of a sentence is built up recursively from the meaning of its well-formed parts. The meaning of individual words is given in the form of lambda terms. For an overview, see e.g. Gamut [49].

As an example, consider sentence (1.1). Its (oversimplified) semantics is shown in (1.2), and the meanings of the individual words are given in (1.3). Assembling the meanings of the words in the order prescribed by the syntax of the sentence, we first apply the lambda term for "every" to the lambda term for "plan". We apply the result to the lambda term that represents the meaning of "worked", and we get the formula shown in (1.4). Completely beta reducing this formula, we arrive once again at the formula shown in (1.2).

(1.1) Every plan worked.

(1.2) $\forall x.\text{plan}'(x) \to \text{work}'(x)$

(1.3)
$$\begin{array}{rl} \text{every:} & \lambda P \lambda Q \forall x. P(x) \to Q(x) \\ \text{plan:} & \lambda x.\text{plan}'(x) \\ \text{worked:} & \lambda x.\text{work}'(x) \end{array}$$

(1.4) every plan worked: $\big(\lambda P\lambda Q\forall x.P(x) \rightarrow Q(x)\big)\lambda x.\text{plan}'(x)\lambda x.\text{worked}'(x)$

Lambda calculus plays a double role in this formalism: On the one hand it serves as a tool for a compositional semantic construction, as sketched in the previous paragraph. On the other hand the lambda operator is used as a class-building operator, to represent semantic aspects of some natural language expressions.

1.3.2 Underspecification

Ambiguity is a pervasive problem in natural language processing, at all levels of linguistic structure. Multiple sources of ambiguity lead to a combinatoric explosion in the number of readings for a sentence. There are many ways of dealing with ambiguity, the simplest being to enumerate all readings and to process them separately. The technique that we are interested in is *underspecification*: the construction of a single compact description of all readings. Underspecification has the following interesting properties:

- It provides a single representation instead of many.
- Choice points are localized.
- Operations on the underspecified representation operate on all readings at once.
- Monotonic augmentation of an underspecified description can be used instead of destructive changes on a fully specified data structure.
- In some cases of ambiguity different readings can be distinguished, but a listener does not necessarily decide between them (see e.g. Pinkal [95, 96]). Such cases can be modeled by underspecification.
- In ambiguity resolution, roughly two groups of cases can be distinguished: those that force listeners to stop and reconsider, and those that go unnoticed. For a cognitively adequate modeling of human language understanding, the first group of cases can be modeled e.g. by backtracking, the second group by augmentation of an underspecified representation (see e.g. Marcus, Hindle and Fleck [87]).

Scope Ambiguity is a kind of ambiguity that is considered especially hard. It arises when in the logical formula describing a sentence meaning, there is more than one possibility for the scope that some element of the formula can take. This is a phenomenon for which an underspecified representation has often been proposed.

(1.5) Every plan has a catch.

As an example for scope ambiguity, consider sentence (1.5). To get a better understanding of its two readings, we put this sentence into a context: an escape from prison. The sentence can either mean that there is one specific drawback that all plans suffer from; we get this meaning if we continue (1.5) by *...namely the big watchdog in the prison*

yard. Or it can mean that each plan is flawed in a different way, e.g. plan A fails because we do not possess the key to the prison door, and plan B will not work because we are too lazy to dig our way out. This ambiguity results in two different logic formulas for the sentence. The first reading is shown in (1.6) and the second in (1.7). For better readability, we have written "a catch" for $\lambda Q.\exists z.\text{catch}'(z) \wedge Q(z)$, and likewise "every plan" for $\lambda Q.\forall z.\text{plan}'(z) \rightarrow Q(z)$.

(1.6) (a catch)(λx
(every plan)(λy
(have$'(x, y)$))))

(1.7) (every plan)(λx
(a catch)(λy
(have$'(x, y)$))))

The important point is that the two formulas differ only in the order of the two *quantifiers* "a catch" and "every plan". This is the basic idea of all underspecified representations of scope ambiguity [100, 91, 15, 95, 26]. A representation like the one in Fig. 1.2 is often used: The two upper partial formulas have holes, and these holes need to be plugged which other partial formulas. The dotted lines stand for outscoping. That is, both quantifiers need to outscope "$have(x, y)$", but it is not specified in which order. Intuitively, there are two ways of satisfying this description, and they correspond to the two formulas in (1.6) and (1.7).

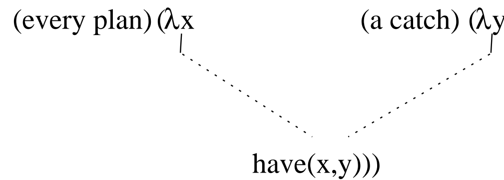


Figure 1.2: The basic idea of an underspecified representation of scope, exemplified on sentence (1.5), "Every plan has a catch."

1.3.3 Parallelism

Parallelism phenomena are ubiquitous in natural language. Intuitively, parallelism means that some structure is repeated in the same or a very similar way. Parallelism often occurs together with *ellipsis*: Some linguistic material is left out even though it should have been present by some syntactic or lexical restrictions.

(1.8) John found the error before Bill did.

(1.9) John sent a letter to Bill, and a parcel too.

(1.10) No student laughed, except John.

(1.11) The bike has a flat tire – no, two.

(1.12) Who is the next to jump? – John.

A few examples are shown in (1.8) through (1.12). Sentence (1.8) is a case of VP ellipsis: The *target sentence* "Bill did" means the same as the non-elliptical "Bill found the error." The meaning of the target sentence can be recovered by recourse to the *source sentence* "John found the error". "John" and "Bill' are called the *contrasting elements* of the source and target sentences. Sentence (1.9) is a bare argument ellipsis, sentence (1.10) shows an exception phrase fragment, and sentence (1.11) is a correction: "no, the bike has two flat tires". The answer in (1.12) means the same as "John is the next to jump."

There have been many different approaches to modeling ellipsis, involving different levels of *linguistic structure*. Levels of linguistic structure relate the surface structure of an expression to its meaning. Examples of such levels are (surface) syntactic structure, morphology, and the level of formal semantic structure that we have briefly discussed in Sec. 1.3.1. The approach of Dalrymple, Shieber and Pereira [30] focuses on the level of formal semantics; it views ellipsis as a missing property of the target contrasting element, which is recovered using higher order unification. In the case of our first example sentence (1.8), the meaning of the target sentence would be given as $P(bill)$, where the property P could be determined by an equation like $find(john, the_error) = P(john)$. Lappin and Shih [82] reconstruct the missing pieces within the surface syntactic structure of the target sentence, taking them from corresponding positions in the source sentence syntax. Hardt [59] sees ellipsis as a case of *referential* identity: Both the source and the target sentence refer to the same kind of event, much like in the sentence "John cut himself", "John" and "himself" refer to the same referent. Kehler [70] gives a central role to discourse structure: In his analysis, the way in which the target sentence meaning is recovered relies crucially on the coherence relation that holds between the source and the target sentence.

Interestingly, ellipsis and scope ambiguity interact. This phenomenon, *quantifier parallelism*, first became apparent in examples proposed by Hirschbühler [62]. An example of this interaction is shown in (1.13).

(1.13) Every linguist attended a workshop. Every computer scientist did, too.

The first sentence in (1.13) contains two scope-bearing elements, "every linguist" and "a workshop". So it has two readings, just like (1.5) above. The second sentence in (1.13) means the same as "Every computer scientist attended a workshop". Here we have two scope-bearing elements again, so (1.13) should have four readings all in all. But of those four, only two exist, plus an additional third reading: Either all linguists attend one common workshop, and all computer scientists visit a (potentially different) common workshop; or everybody has a different workshop that he or she is traveling to; or one and the same workshop is attended by everybody. There are no "mixed" readings in which, e.g., all linguists gather at one workshop, while the computer scientists disperse to different workshops. That is, if "a workshop" takes wide scope in the first sentence, it has to take wide scope in the second sentence too – parallelism enforces a parallel resolution of scope ambiguities. The third reading, where one single workshop is

attended by everybody, arises when “a workshop” moves out of the source sentence and takes scope over both the source and the target sentence.

How can scope ambiguity, parallelism, and their interaction be modeled formally? This is the question that stood at the beginning of the Constraint Language for Lambda Structures, which we present in the next section.

1.4 The Constraint Language for Lambda Structures (CLLS)

The *Constraint Language for Lambda Structures (CLLS)* is a constraint language interpreted over the class of *lambda structures* [42, 41]. A lambda structure is a constructor tree augmented by functions modeling binding. It can be described as a lambda term viewed as a tree. A CLLS constraint describes relations between *nodes* of a lambda structure, i.e. it adopts the internal perspective on trees. The two most important constructs of the language are *dominance* and *parallelism*:

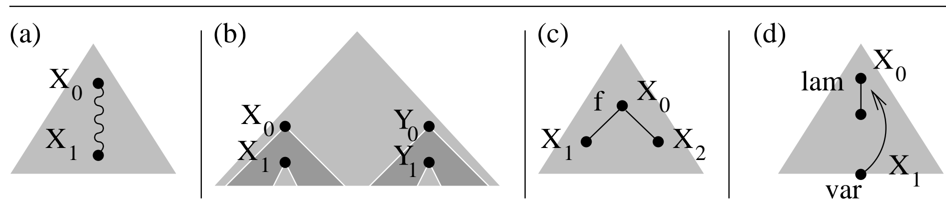


Figure 1.3: Labeling, dominance, parallelism, and lambda binding

- *Dominance* is the ancestor relation between nodes, or, more precisely, the reflexive and transitive closure of the “parent-of” relation. It is illustrated in Fig. 1.3 (a). We write

$$X_0 \triangleleft^* X_1$$

  to state that X_0 dominates X_1. (X_0 and X_1 are variables standing for nodes of a lambda structure.)

- *Parallelism* is structural isomorphism between pairs of tree pieces, called *segments*. This is illustrated in Fig. 1.3 (b). The segments are the deeper shaded regions. A segment is a subtree from which zero or more subtrees have been cut out, leaving behind *holes*. In the picture, X_0 and X_1 delineate one segment: X_0 addresses the root and X_1 the single hole. In the other segment, Y_0 stands for the root and Y_1 for the hole. (Again, X_0, X_1, Y_0, Y_1 stand for nodes in a lambda structure). We write

$$X_0/X_1 \sim Y_0/Y_1$$

  to state that the segment between X_0 and X_1 has the same structure as the segment between Y_0 and Y_1. This notation is extended canonically to parallelism with 0 or more holes.

Furthermore, the language CLLS can express *labeling*, as shown in Fig. 1.3 (c): A labeling constraint states the label of a node along with all its children. The constraint drawn in the picture is written as

$$X{:}f(X_1, X_2).$$

It states first that X is labeled f, and second, that X has X_1 as its left child and X_2 as its right child.

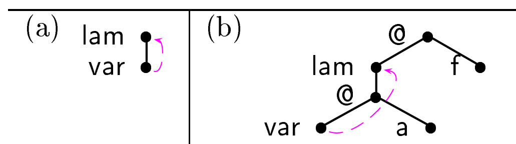


Figure 1.4: Lambda structures representing (a) $\lambda x.x$, (b) $(\lambda x.x(a))f$

The fourth construct in Fig. 1.3 is *lambda binding*. Above we have said that a lambda structure is a constructor tree augmented by functions modeling binding. Figure 1.4 (a) shows the lambda structure for the lambda term $\lambda x.x$. The variable x of the lambda term is completely nameless in the lambda structure. Instead, lambda binding is expressed by the *lambda binding function*, which maps the var-labeled node to its binder. In the picture this mapping is represented by the dashed arrow. Likewise, Fig. 1.4 (b) shows the lambda structure for $(\lambda x.x(a))f$. Here, as in the rest of this thesis, we represent *application* by the symbol @.

Returning to the *lambda binding constraint* in Fig. 1.3 (d), this constraint states that the (var-labeled) X_1 is bound at the (lam-labeled) X_0. We write it as

$$\lambda(X_1){=}X_0.$$

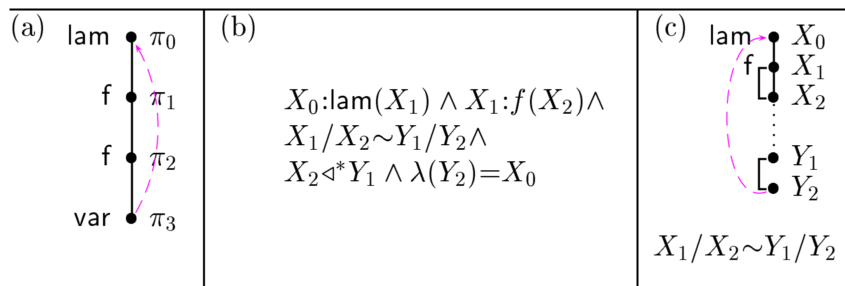


Figure 1.5: (a) A lambda structure, (b) a constraint that it satisfies, and (c) the constraint graph for the constraint in (b)

A CLLS constraint is a conjunction of *literals*, atomic constraints like for example dominance, parallelism, labeling and lambda binding literals. Figure 1.5 demonstrates all these four types of literals. Picture (a) shows a lambda structure, which is non-branching for reasons of simplicity. As a lambda term, it would read $\lambda x.ffx$ (modulo alpha-renaming)

for a string ff. Picture (b) shows a constraint that is satisfied by the lambda structure in (a) with the valuation

$$X_0 \mapsto \pi_0, X_1 \mapsto \pi_1, X_2 \mapsto \pi_2, Y_1 \mapsto \pi_2, Y_2 \mapsto \pi_3.$$

The constraint $X_0{:}\mathsf{lam}(X_1)$ is satisfied since the node π_0 is labeled lam and is the parent of π_1, which is labeled f and is the parent of π_2, so this satisfies $X_1{:}f(X_2)$. The parallelism constraint $X_1/X_2{\sim}Y_1/Y_2$ is satisfied since the segment starting at π_1 and ending at π_2 has the same structure as the segment starting at π_2 and ending at π_3: They both have the structure f . Note that the label of the hole node does *not* count as part of the segment. The dominance constraint $X_2 \triangleleft^* Y_1$ is satisfied since π_2 dominates itself, and $\mathsf{lam}(Y_2){=}X_0$ is satisfied because π_3 has its lambda binder at π_0. CLLS constraints can easily be visualized as tree-like graphs, as picture (c) shows: It is the constraint of picture (b) drawn as a *constraint graph.* Labeling and lambda binding look as in a lambda structure. Dominance is shown as a dotted line. Parallelism is written below the constraint graph; additionally it may be sketched using brackets, as we have done here.

How do the literals of CLLS relate to the tree description languages that we have seen earlier on? Dominance is the same relation that we have seen in SkS and in Blackburn and Meyer-Viol's modal tree logic. Parallelism is similar to the repeated use of the same context variable in CU. The segments that parallelism is about are almost the same as the contexts of CU, except that a segment is always a segment *of* a lambda structure, with which it is connected via the lambda binding function.

Another point worth noting is that apart from the lambda binding function, lambda structures offer a second binding function, which models *anaphoric binding.* An anaphor is an element or a construction that, in order to be interpreted, needs to be associated with something else in the context. For example, in "John cut himself" the anaphoric "himself" needs to be associated with "John". Like the lambda binding function, the anaphoric binding function is described by matching literals of the language CLLS.

1.4.1 CLLS in Underspecified Semantics

The language CLLS can be used to model scope ambiguity and ellipsis: Scope ambiguity can be modeled with dominance constraints, and ellipsis with parallelism constraints [42, 41].

Scope ambiguity. Consider again the sentence we discussed above, "Every plan has a catch," shown in (1.5). Its two readings, shown in (1.7) and (1.6), are lambda terms, so they can also be seen as lambda structures. Above we have remarked that these two readings just differ in the scope of the scope-bearing elements "a catch" and "every plan", where scope in the formula is the same as dominance in the lambda structure.

These two readings can be described by one common constraint: the one shown in Fig. 1.6. (This is again a *constraint graph*, like the one in Fig. 1.5 (c), a graphical representation

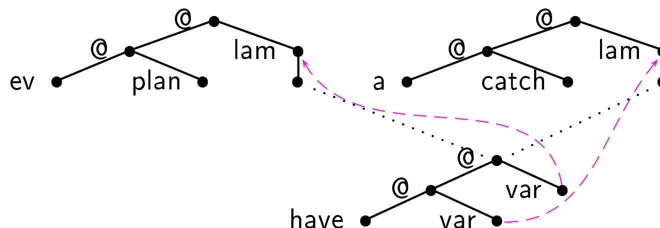


Figure 1.6: Constraint for *Every plan has a catch.*

of a constraint.) The graph has all the symbols of the two higher-order formulas (1.7) and (1.6) as node labels. Variable binding is again indicated by dashed lambda binding edges, and dominance is again drawn as dotted lines.

The important point is this: The constraint states that the "constraint fragment" for "every plan" outscopes the fragment for "have", and that the fragment for "a catch" outscopes the fragment for "have", but it does not specify any order for the fragments for "a catch" and "every plan". However, trees do not branch upwards, so one of two cases must hold: Either the fragment for "every plan" outscopes that for "a catch", or vice versa. So it is exactly the partial information given by the dominance constraints that describes the scope ambiguity.

More generally, each individual reading of a sentence is represented by a lambda structure. A CLLS constraint with several models is an underspecified representation for the set of all these models (i.e. lambda structures). So we can view lambda calculus as our "object-level language" and CLLS as a "meta-language" for talking about formulas in the object-level language.[1]

(1.14) Every man sleeps, and so does Mary.

Ellipsis. Sentence (1.14) shows a simple case of VP ellipsis. The meaning of the target sentence "...and so does Mary" is the same as the meaning of the source sentence "Every man sleeps", except that the source contrasting element "every man" is replaced by the target contrasting element "Mary". This can be modeled by the constraint in Fig. 1.7: First, the part of the constraint below X_0 represents the meaning of the source sentence, "every man sleeps". Second, the part of the constraint below Y_0 represents the meaning of the target sentence: We only know that it contains the meaning of "Mary", which is the node X_1. Finally, the parallelism constraint $X_0/X_1{\sim}Y_0/Y_1$ states that the meaning of the source and the target sentence are the same, except for the substructures representing the contrasting elements.

[1]Note that we do not take the term *meta-language* as a formal notion here. Formally, CLLS is just an object-level language that we use to describe objects of another object-level language.

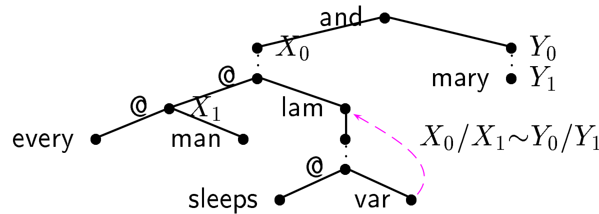


Figure 1.7: Constraint representing the meaning of sentence (1.14): "Every man sleeps, and so does Mary."

And in fact the lambda structure representing the meaning of sentence (1.14), shown in Fig. 1.8, satisfies the constraint in Fig. 1.7.

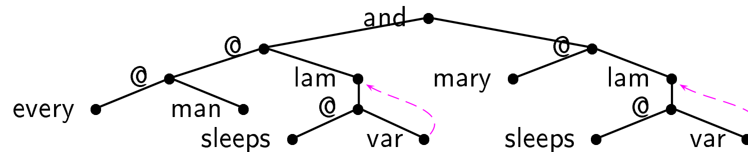


Figure 1.8: Lambda structure representing the meaning of sentence (1.14): "Every man sleeps, and so does Mary."

So we can model ellipsis using parallelism constraints. But how about the interaction between ellipsis and scope ambiguity? This phenomenon, *quantifier parallelism*, is illustrated in sentence (1.13) (p. 9) above: If we have a scope ambiguity in the source sentence of an ellipsis, then this scope ambiguity must be resolved in the same way in the source and in the target sentence. But this is automatically enforced by parallelism: A scope-bearing element in the source sentence and its copy in the target sentence must have the same positions in their respective segments – that is what structural isomorphism means.

1.4.2 CLLS in Underspecified Beta Reduction

CLLS is a language for partial descriptions of lambda terms. So an obvious question to ask is: Can we do beta reduction directly on CLLS constraints? This is the question of *underspecified beta reduction*. Ideally, we would like to be able to beta reduce a CLLS constraint without disambiguating it first in any way.

An obvious approach to solving this problem would be to lift beta reduction canonically from term rewriting on lambda terms to graph rewriting on constraint graphs. But this does not work. The problem is that in a CLLS constraint the structure of the lambda term that it describes may be only partially known. And this can lead to the simple rewriting approach generating spurious solutions – it is unsound.

But there is an alternative approach that *is* sound, an approach that is declarative instead of procedural [11, 12]: describing the *result* of beta reduction using parallelism constraints. Consider Fig. 1.9. Picture (a) is a sketch of a lambda structure with a redex, and picture

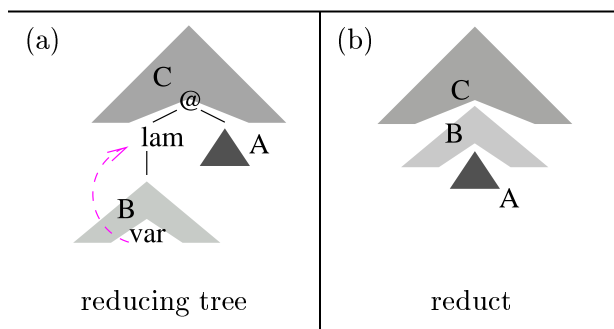


Figure 1.9: Beta reduction on lambda structures – abstract schema

(b) sketches the result of beta reduction. In (a), we have the lambda abstraction with body B and (in this case) one occurrence of the bound variable, and we have the argument A that is to be substituted for the bound variable. The context C that surrounds the redex will not be changed during beta reduction. All three, body B, argument A and context C, are segments of this lambda structure. In the result of beta reduction, (b), all three segments reappear, although their relative positions have changed.

Now the idea is to regard both the lambda term before beta reduction and the reduct as *parts of the same bigger lambda structure*, and to relate the two context segments, the two body segments, and the two argument segments by one parallelism each. Then the parallelism constraints naturally enforce that all ambiguities in the reducing tree are resolved in the same way there as in the reduct – this is the same effect that we get for quantifier parallelism cases.

1.5 A Procedure for CLLS

The central topic of this thesis is a procedure for CLLS constraints, in particular parallelism constraints. The tasks of the procedure are to check satisfiability, and to make explicit information that is given only implicitly in the constraint.

For a fragment of CLLS, *dominance constraints*, constraint solvers exist. Dominance constraints include the dominance and labeling literals we have seen above, but not parallelism literals.[2] The satisfiability problem for dominance constraints is decidable: it is an NP-complete problem [78]. For the language of dominance constraints there is a solver in the constraint programming paradigm, based on finite set constraints, by Duchier and Niehren [34]. For a fragment of dominance constraints, *normal dominance constraints*, which seems to suffice for the linguistic application, satisfiability can be tested in polynomial time [76]. For this fragment Koller, Mehlhorn, Niehren, Althaus, Duchier and Thiel have proposed polynomial-time solvers based on graph algorithms [3].

How can these solvers be extended to procedures for all of CLLS? The only constructs

[2]The fragment does not comprise binding constraints, but only for the sake of clarity; adding them does not pose any problems.

in CLLS that are not present in dominance constraints are parallelism and binding literals. The main problem is to construct a procedure for parallelism constraints; binding constraints are relatively easy to handle. As we have mentioned above, parallelism constraints are equally expressive as CU, and the decidability of CU is still an open problem. As this is a problem that we will not attempt to solve here, our aim must be to construct a *semi-decision procedure for parallelism constraints.* Now a naive procedure is very easy to put up: Just enumerate lambda structures and check for each of them if it satisfies the given constraint. But such a procedure is of course not satisfactory – it is neither feasible, nor does it provide insights into the nature of the problem. In contrast, we will present a procedure that

- terminates for the linguistically relevant cases of CLLS constraints, and computes result constraints for them.
- includes a solver for dominance constraints. Given a dominance constraint as an input, the parallelism constraint procedure behaves exactly like the dominance constraint solver that it encompasses. This is advantageous because dominance constraints play an important role in the linguistic application.
- is built in a modular fashion, so that in principle different dominance constraint solvers can be incorporated.
- introduces a data structure, *correspondence*, that will prove useful in stating the formalism as well as in processing.

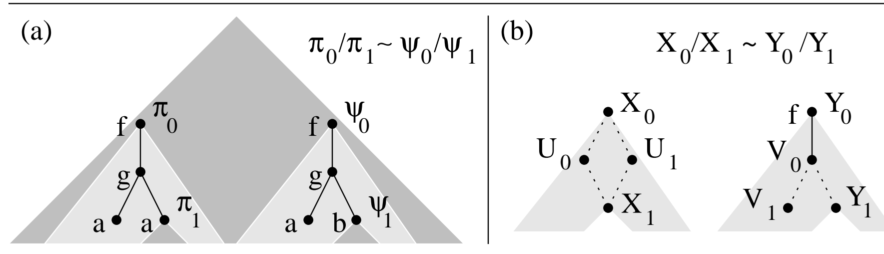


Figure 1.10: Parallelism in a lambda structure, and a parallelism constraint

Next we sketch the main ideas of the procedure for parallelism constraints that we introduce. Consider Fig. 1.10. Picture (a) shows an example of parallelism in a lambda structure: the two segments are structurally isomorphic up to their holes. Picture (b) shows an example of a CLLS constraint, including the parallelism literal $X_0/X_1{\sim}Y_0/Y_1$. We call the two terms X_0/X_1 and Y_0/Y_1 *segment terms.* A procedure for CLLS has to check whether the two segments described by the segment terms X_0/X_1 and Y_0/Y_1 can have the same shape. But as sketched in picture (b), we only have a partial description of these two segments: We know that either U_0 dominates U_1 or vice versa, but we don't know which of the two cases holds, and we know that V_0 dominates both V_1 and Y_1, but we don't know the relative positions of V_1 and Y_1.

How can we test such a constraint for satisfiability? Can a look at the closest relative of parallelism constraints, CU, help us here? No, unfortunately the procedure for CU equation systems is not particularly suitable for parallelism constraints: It determines the shape of a context in a top-down fashion, starting at the root of the context and working downward. In the process, it sometimes has to guess labels. However, in parallelism constraints there is no preferred direction, and we do not want to guess labels. Furthermore, and more importantly, dominance constraints do not seem to correspond to any clear-cut fragment of context unification, and it is essential for our procedure to work well for dominance constraints.

Instead, we use a new data structure that makes use of the node-centered perspective on trees that CLLS takes: A *correspondence function* between two parallel segments maps each node in one segment to the node in the same position in the other segment. In Fig. 1.11 (a), corresponding nodes are linked by arcs. Note that any two corresponding nodes bear the same labels and have corresponding children, except for the holes of the two segments. Among other things, correspondence functions allow for a straightforward formulation of the conditions on binding in interaction with parallelism.

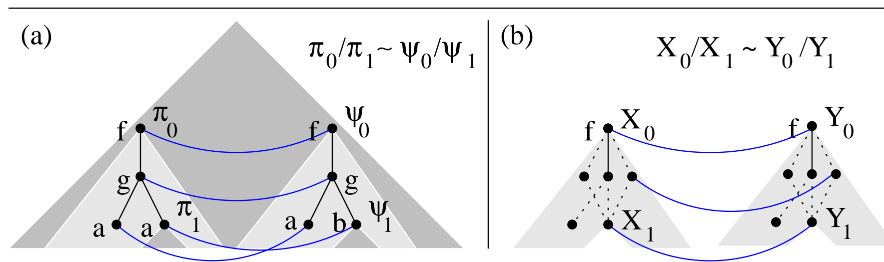


Figure 1.11: Correspondence: (a) correspondence function in a lambda structure, (b) correspondence formulas in a constraint

We use the same idea in our procedure for parallelism constraints: We link corresponding variables by *correspondence formulas*, which state that the two linked variables denote corresponding nodes. Then, the general modus operandi of the procedure will be as follows: First we copy all variables from one parallel segment term to the other, and we link each variable to its copy by a correspondence formula. Second, any relation between two variables "inside" the left segment term must also hold between their correspondents in the right segment term, and vice versa; that is what structural isomorphism is about. If we augment the constraint in Fig. 1.10 (b) accordingly, we arrive at the constraint in Fig. 1.11 (b). For the sake of readability we have drawn in only a few correspondence formulas, again as arcs. Third, we use the dominance constraint solver to sort out the relative positions of the variables "inside" each of the two segment terms. All the while we make sure that one invariant is maintained: Any relation that holds between variables "inside" one parallel segment term will also hold between their correspondents "inside" the other segment term.

We will formulate this semi-decision procedure for CLLS as a high-level, rule-based proce-

dure. It transforms constraints, or more accurately, it augments them, until a *saturation* is reached. A saturation is similar to a solved form: It is a constraint from which a model can be directly read off. We are going to prove the following properties of the procedure:

- Any saturation that the procedure computes is satisfiable.
- The procedure is sound in the sense that all its rules are equivalence transformations.
- It is complete in the sense that it computes all *minimal* saturations for a given input constraint.
- The notion of *minimal saturation* can be compared to the most general unifiers of unification problems. While unifiers are compared by the subsumption ordering, we compare CLLS constraints by a partial order that can be described roughly as the subset relation modulo α-renaming of variables introduced during processing.

As mentioned above, there are two different applications of CLLS that we will study: underspecified natural language semantics, and underspecified beta reduction. From the point of view of these applications, the CLLS procedure enumerates readings in the case of scope ambiguities, and makes structural isomorphism explicit in the cases of ellipsis resolution and underspecified beta reduction. For these application areas, extensions to both the language and the procedure are necessary.

- Is the formal language adequate for modeling the phenomena arising in underspecified semantics and in underspecified beta reduction? It turns out that a straightforward extension of parallelism is needed for both applications: In the sketch for beta reduction on lambda structures (Fig. 1.9), we have used several pairs of parallel segments to relate the reducing tree and the reduct. Similarly there are cases of ellipsis for which several pairs of parallel segments are needed. However it is not sufficient to use *independent* parallelism relationships, since normal parallelism is too restrictive in its conditions on lambda binding. For example, in Fig. 1.9 (a) (p. 15) we need to allow a lambda binder from the B segment to the C segment, and this binder has to parallel a lambda binder from the B segment to the C segment in picture (b). What we need in these cases is to be able to treat a *group* of segments as if it were a single segment. The new relation that ensues is called the *group parallelism relation*. Its conditions on binding are less strict than in normal parallelism and can be expressed straightforwardly in terms of correspondence functions.
- In the application to modeling ellipsis a further extension to the parallelism relation will prove useful. Normal parallelism might state something like $\pi_0/\pi_1{\sim}\psi_0/\psi_1$, "node π_0 up to π_1 is parallel to node ψ_0 up to node ψ_1", which means that the *subtrees* below π_1 and ψ_1 are excluded from the parallelism. But sometimes it is necessary to except not subtrees but *segments*, stating something like "node π up to the segment between π_1 and π_2 is parallel to node ψ up to the segment between ψ_1 and ψ_2". As in normal parallelism, the excluded segments have to be in the same

position within the two parallel segments, as sketched in Fig. 1.12. This can again be described by a concept of *correspondence*. The new parallelism relation will be called *jigsaw parallelism*. It does not add any expressive power – any situation that the jigsaw parallelism relation can describe can already be expressed using group parallelism. However there are cases where a single jigsaw parallelism *constraint* can only be described by a *disjunction* of group parallelism constraints.

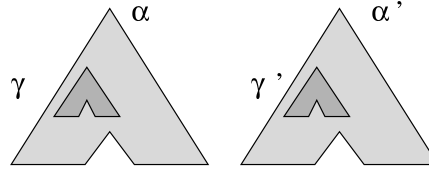


Figure 1.12: Jigsaw parallelism: Segment α up to segment γ is parallel to α' up to γ'

- We use the same constraints to describe the result of a beta reduction step that we use to describe the meaning of elliptical sentences. So in principle we can use the same procedure to handle both applications. However, in performing underspecified beta reduction we would like to maintain as much underspecification as possible – the CLLS procedure that we have sketched above makes explicit more choices that are implicit in the constraint than we would like for this application. In particular, it disambiguates scope ambiguities.

  So we introduce a procedure specialized on computing the result of an underspecified beta reduction step. It makes the structural isomorphism explicit, but, at the cost of being incomplete, it can avoid disambiguation in many cases by exploiting knowledge about the relative positions of parallel segments in underspecified beta reduction. To do that, we use yet another variant of the concept of correspondence: *underspecified correspondence*.

1.6 Contributions

Summing up the points mentioned in the previous section, the contributions of this thesis are as follows:

- The most important result is a semi-decision procedure for CLLS. It is a high-level, rule-based procedure that computes *saturations*, from which models can be directly read off. We show that the procedure is sound and complete, and we introduce a notion of *minimal saturation*.

  A central concept both for the description of the formalism and for the procedure is the notion of *correspondence*, which also proves vital for extensions both to the language and the procedure.

- We extend the language CLLS by moving from parallelism to *group parallelism*, where the conditions on binding are more liberal, and we extend the procedure for solving the constraints accordingly.

We further extend the language to encompass *jigsaw parallelism* constraints, where segments rather than subtrees are excepted from parallelism. We show how both these extensions can be used for modeling ellipsis.

- We present a procedure that computes the result of an *underspecified beta reduction step*, building on the CLLS procedure. It is incomplete, but avoids disambiguation in many cases.

Source Material

Part of the material presented in this thesis has already been published in the following papers:

- Katrin Erk and Joachim Niehren. Parallelism Constraints, 2000 [46].
- Katrin Erk, Alexander Koller and Joachim Niehren. Processing Underspecified Semantic Representations in the Constraint Language for Lambda Structures, 2000 [44].
- Manuel Bodirsky, Katrin Erk, Alexander Koller and Joachim Niehren. Beta Reduction Constraints, 2001 [12].
- Manuel Bodirsky, Katrin Erk, Alexander Koller and Joachim Niehren. Underspecified Beta Reduction, 2001 [13].
- Katrin Erk and Alexander Koller. VP Ellipsis by Tree Surgery, 2001 [43].

Part of the discussion of underspecified beta reduction also appeared in the master's thesis of Manuel Bodirsky [11].

1.7 Plan of this Thesis

This thesis consists of two parts, *A Procedure for CLLS Constraints* and *Applying Parallelism Constraints.*

A Procedure for CLLS Constraints. In this part of the thesis we present a semi-decision procedure for CLLS. The first chapter lays the ground for all others that follow: It introduces the formalism, gives the basic definitions and some examples and discusses related formalisms and modeling approaches. The following three chapters present the semi-decision procedure for CLLS. In Chapter 3 we discuss the part of the procedure that handles dominance constraints, and we prove soundness and completeness for this part. We reuse and extend the same proof outlines in the two chapters that follow. Chapter 4 introduces the part of the procedure that deals with parallelism constraints. This is the central chapter, and the one with the most interesting proofs. Chapter 5 completes the procedure with the part that handles binding.

Applying Parallelism Constraints. In this part of the thesis we consider two applications of parallelism constraints: modeling ellipsis in a framework of underspecified semantics, and underspecified beta reduction. In Chapter 6 we discuss underspecified beta reduction. We define group parallelism, and we present two procedures for computing the result of a beta reduction step: a canonical extension to the CLLS procedure that can handle group parallelism literals, and a procedure that is incomplete but can avoid disambiguation in many cases. Chapters 7 and 8 are about modeling ellipsis with parallelism. In Chapter 7 we introduce jigsaw parallelism, and we show how both group parallelism and jigsaw parallelism can be used to model ellipsis. In Chapter 8 we take a closer look at the phenomenon of ellipsis, we discuss different approaches to modeling ellipsis, and we position the CLLS approach in relation to them.

Finally, Chapter 9 lists further research questions, and Chapter 10 sums up and concludes.

Part I

A Procedure for CLLS Constraints

Chapter 2

CLLS

This chapter contains the basic concepts and definitions that we will use throughout the thesis. It defines *lambda structures* and the *Constraint Language for Lambda Structures* (CLLS), which was first introduced in 1998 by Egg, Niehren, Ruhrberg and Xu [42].

Lambda structures are constructor trees augmented by a function that maps bound variables to their lambda binders, plus another node mapping function that is needed for the application to linguistics. Each lambda structure corresponds to a lambda term that is unique up to α-equivalence. CLLS offers constraints that describe relations between nodes of a lambda structure.

In this chapter, we present the language CLLS as a hierarchy of three languages. The language of *dominance constraints* $\mathcal{C}_{\mathrm{d}}$ describes trees. The most important node relations that it can describe are *labeling*, which describes a node's label as well as its immediate descendants, and *dominance*, which states that one node is above another, without specifying how far apart they are. The language of *parallelism constraints* $\mathcal{C}_{\mathrm{p}}$ extends $\mathcal{C}_{\mathrm{d}}$ by *parallelism*, which says that two segments have the same structure. Finally, the language *CLLS* extends $\mathcal{C}_{\mathrm{p}}$ by *binding* and thus moves beyond tree descriptions to descriptions of lambda structures.

The tripartite hierarchy of languages matches three major classes of linguistic phenomena that they can be used to model: dominance constraints can be used for a compact underspecified representation of scope ambiguities. Parallelism constraints can additionally model parallelism phenomena. The binding literals of CLLS handle anaphoric binding.

2.1 Lambda Structures

In this section we introduce lambda structures in three steps: First we define constructor trees. Then we augment trees by a parallelism relation. The third step extends trees by binding functions, which yields lambda structures.

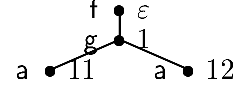
2.1.1 Constructor Trees

We assume a signature Σ of function symbols ranged over by $f, g, \ldots$, each of which is equipped with an arity $\mathsf{ar}(f) \geq 0$. We assume that Σ contains at least one constant and

a symbol of arity at least 2.

A *finite (constructor) tree* θ is a ground term over Σ. A *node* of a tree can be identified with its *path* from the root down, expressed by a word over $\mathbb{N}$ (the set of natural numbers excluding 0). We use the letters π, ψ for paths. We write ε for the empty path and $\pi_1\pi_2$ for the concatenation of two paths π_1 and π_2. A path π_1 is a prefix of a path π if there exists some (possibly empty) path π_2 such that $\pi_1\pi_2 = \pi$.

A tree can be characterized uniquely by a *tree domain* (the set of its paths) and a *labeling function*. A tree domain D is a finite nonempty prefix-closed set of paths. A labeling function is a function $L : D \to \Sigma$ from a tree domain to Σ fulfilling the condition that for every node $\pi \in D$ and $k \geq 1$, $\pi k \in D$ iff $k \leq \mathsf{ar}(L(\pi))$. We write D_θ for the domain of a tree θ and L_θ for its labeling function. For instance, the tree $\theta = f(g(a, a))$ shown to the right satisfies $D_\theta = \{\epsilon, 1, 11, 12\}$, $L_\theta(\epsilon) = f$, $L_\theta(1) = g$, and $L_\theta(11) = a = L_\theta(12)$.



Since we will be talking about lambda structures later on, it is useful to view finite trees as *tree structures*. The tree structure of a finite tree θ over Σ is a first-order structure with domain D_θ. It provides a labeling relation $:f$ for each $f \in \Sigma$:

$$:f = \{(\pi, \pi 1, \ldots, \pi n) \mid L_\theta(\pi) = f, \mathsf{ar}(f) = n\}$$

Overloading notation somewhat, we also write θ for the tree structure of a tree θ. We write

$$\theta \models \pi_0{:}f(\pi_1, \ldots, \pi_n)$$

for $(\pi_0, \pi_1, \ldots, \pi_n) \in :f$. This relation states that the node π_0 of θ is labeled by f and has π_i as its i-th child (for $1 \leq i \leq n$). The labeling relation is illustrated in Fig. 2.1 (a) for a symbol f of arity 2.

Every tree structure θ can be extended conservatively by relations for inequality, dominance, and disjointness:

Definition 2.1 (Dominance, disjointness). *Let θ be a tree structure.*

- *The* dominance *relation on D_θ is defined as $\triangleleft^* = \{(\pi_0, \pi_1) \mid \pi_0$ is a prefix of $\pi_1\}$.*
- *The* disjointness *relation on D_θ is defined as $\perp = \{(\pi_0, \pi_1) \mid$ neither $\pi_0 \triangleleft^* \pi_1$ nor $\pi_1 \triangleleft^* \pi_0$ holds in $\theta\}$.*

For better readability, we use the infix notation $\pi \triangleleft^* \psi$, $\pi \perp \psi$ instead of the tuple notation. The dominance relation is illustrated in Fig. 2.1 (b): It is the reflexive and transitive closure of the parent relation. So a node π_0 *dominates* π_1 iff it is its ancestor. We also use *strict dominance*: We write $\pi_0 \triangleleft^+ \pi_1$ if both $\pi_0 \triangleleft^* \pi_1$ and $\pi_0 \neq \pi_1$ hold in θ. Two nodes π_0, π_1 lie in *disjoint* position iff there is some other node ψ_0 such that $\psi_0{:}f(\ldots \psi_i, \ldots, \psi_j \ldots)$ holds in θ for distinct children ψ_i, ψ_j of ψ_0, and $\psi_i \triangleleft^* \pi_0$ while $\psi_j \triangleleft^* \pi_1$. Disjointness is illustrated in Fig. 2.1 (c).

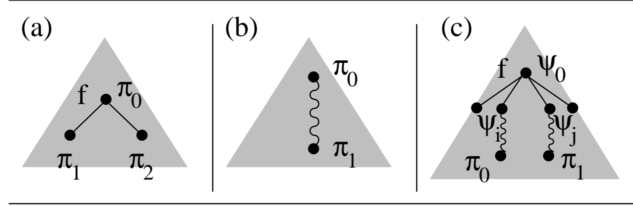


Figure 2.1: Labeling, dominance, and disjointness

2.1.2 The Parallelism Relation

We extend tree structures by a *parallelism relation*, which states structural isomorphism between pairs of tree pieces, called *segments*.

A segment is a subtree in which some subtrees have been replaced by *holes*. For example, Fig. 2.2 shows a segment with root node π_0, and the holes of the segment are at the tree nodes π_1 and π_2. A segment is uniquely defined by its root node and the sequence of its hole nodes from left to right. We write a segment with root π_0 and holes π_1 and π_2 as $\pi_0/\pi_1, \pi_2$.

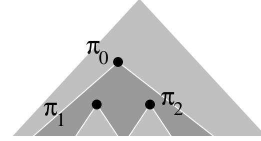


Figure 2.2: A segment

Definition 2.2 (Segments). *A segment α of a tree θ is a tuple $\pi_0/\pi_1 \ldots, \pi_n$ of nodes in D_θ such that $\pi_0 \triangleleft^* \pi_i$ and $\pi_i \perp \pi_j$ hold in θ for all $1 \leq i \neq j \leq n$. The* root *$r(\alpha)$ of the segment is π_0, and $hs(\alpha) = \pi_1, \ldots, \pi_n$ is its (possibly empty) sequence of* holes *ordered from left to right. The set $\mathsf{b}(\alpha)$ of* nodes of α *is*

$$\mathsf{b}(\alpha) = \{\pi \in D_\theta \mid r(\alpha) \triangleleft^* \pi, \text{ and for all } 1 \leq i \leq n, \neg(\pi_i \triangleleft^+ \pi)\}$$

To exclude the holes of the segment, we define $\mathsf{b}^-(\alpha) = \mathsf{b}(\alpha) - hs(\alpha)$, and to exclude all "border nodes", we use $\mathsf{i}(\alpha) = \mathsf{b}^-(\alpha) - r(\alpha)$.

When two segments have the same structure, there exists a *correspondence function* between them:

Definition 2.3 (Correspondence function). *A* correspondence function *between two segments α, β is a bijective mapping $c : \mathsf{b}(\alpha) \to \mathsf{b}(\beta)$ such that c maps the root of α to the root of β and the i-th hole of α to the i-th hole of β for each i, and for every $\pi \in \mathsf{b}^-(\alpha)$ and every label f,*

$$\pi{:}f(\pi 1, \ldots, \pi n) \Leftrightarrow c(\pi){:}f(c(\pi 1), \ldots c(\pi n)).$$

Two corresponding nodes must bear the same labels and have corresponding children, except when the nodes are holes. Whenever it exists, the correspondence function between two segments α and β is unique.

We now define the parallelism relation between pairs of segments. We base the definition on correspondence functions.

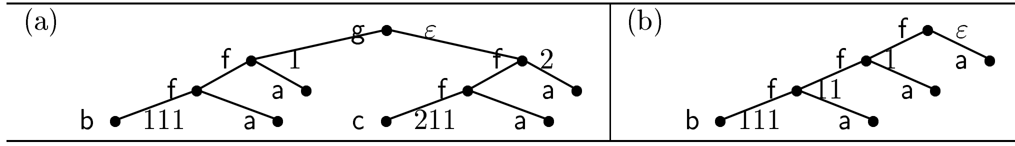


Figure 2.3: In (a) 1/111 and 2/211 are parallel, in (b) $\varepsilon/11$ and 1/111 are parallel.

Definition 2.4 (Parallelism relation). Parallelism *in a tree structure θ is the two-place relation $\alpha \sim \beta$ on segments of θ that holds of a pair α, β iff there exists a correspondence function between α and β.*

For example, in the left tree in Fig. 2.3 the two segments 1/111 and 2/211 are parallel, and in the right tree the two segments $\varepsilon/11$ and 1/111 are parallel. (Note that parallel segments may overlap.)

To provide a better idea of correspondence functions, we prove the following characterization: A correspondence function is a bijection that relates each node in one segment to the node that occupies the same position in the other segment.

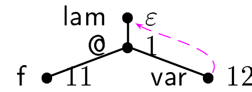
Proposition 2.5. *If $c : \mathsf{b}(\alpha) \to \mathsf{b}(\beta)$ is a correspondence function, then $c(r(\alpha)\pi) = r(\beta)\pi$ for all paths π such that $r(\alpha)\pi \in \mathsf{b}(\alpha)$.*

Proof. By induction on the length of the path π. We have $c(r(\alpha)\varepsilon) = c(r(\beta)\varepsilon)$ because c maps root to root. Now suppose that $r(\alpha)\pi \in \mathsf{b}(\alpha)$ with $c(r(\alpha)\pi) = r(\beta)\pi$, and $r(\alpha)\pi i \in \mathsf{b}(\alpha)$. Then $c(r(\alpha)\pi i) = c(r(\beta)\pi)i = c(r(\beta))\pi i$ by the last condition of Def. 2.3. □

Defining parallelism in terms of a correspondence function is in keeping with the CLLS perspective on trees, which focuses on relations between nodes of a single tree. The concept of correspondence functions will prove important for the CLLS procedure that we develop, in particular for processing parallelism literals (Chapter 4).

2.1.3 Lambda Structures

A *lambda structure* is a constructor tree extended by two node mappings. The mapping λ encodes lambda binding. For example, the lambda term $\lambda x.f(x)$ is represented by the graph to the right. In this case, the lambda binding function, represented as a dashed arrow, contains $\lambda(12) = \varepsilon$. An occurrence of a λ-abstraction in the lambda term is represented as a node labelled lam in the lambda structure, an occurrence of an application is represented as a node labelled @, and an occurrence of a bound variable is represented as a node labelled var. We will also allow $\forall$ and $\exists$ to bind variables, so the range of the function λ will consist of lam-, $\forall$- and $\exists$-labelled nodes.



Apart from the lambda binding function λ, lambda structures possess a second mapping ante from nodes to nodes. This second mapping will be used in the linguistic application to model *anaphoric binding*, the kind of binding that is expressed by the common index 1 in sentence (2.10) (p. 38).

So we assume from now on that the signature Σ contains, on top of the nullary and the binary function symbol we have assumed above, the unary function symbol lam (for lambda abstraction), the unary function symbols $\forall$ and $\exists$, the symbol @ of arity 2 (for functional application), the constant var (for occurrences of a bound variable in a lambda term), and the constant ana (which will be the label of all nodes in the domain of the ante function).

Definition 2.6 (Lambda structures). *A* lambda structure $\mathcal{L}^\theta$ *over* Σ *is a tuple* $(\theta, \lambda, \mathsf{ante})$*, where*

- θ *is a tree structure over* Σ*,*
- λ *is a total function* $\lambda : L_\theta^{-1}(\mathsf{var}) \to L_\theta^{-1}(\{\mathsf{lam}, \exists, \forall\})$ *such that* $\lambda(\pi)$ *is always a prefix of* π*, and*
- ante *is a partial function* $\mathsf{ante} : L_\theta^{-1}(\mathsf{ana}) \rightharpoonup D_\theta$*.*

In the following definition, we deal with the interaction of parallelism and binding: In a lambda structure, two parallel segments need to have not only the same structure, but also a parallel binding structure. We define the parallelism relation $\sim$ on segments of a lambda structure in two steps: First we define a symmetric relation $\sim_\lambda$ that describes conditions on lambda binding, then we define $\sim$ as a non-symmetric subrelation of $\sim_\lambda$.

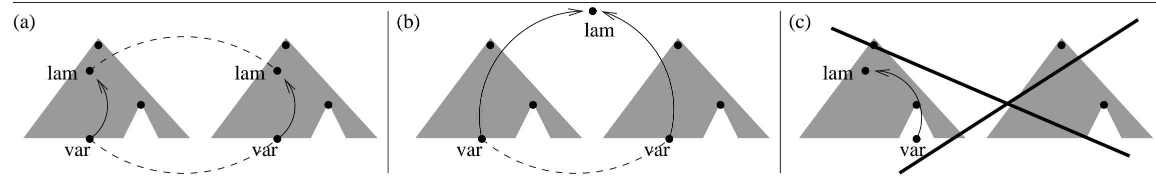


Figure 2.4: Illustrating (a) (λ.same), (b) (λ.out), (c) (λ.hang)

Definition 2.7 (Parallelism relation). *The relation* $\sim_\lambda$ *of a lambda structure* $\mathcal{L}^\theta$ *is the largest* symmetric *relation between segments of* $\mathcal{L}^\theta$ *such that* $\alpha \sim_\lambda \beta$ *implies that first, there exists a correspondence function* c *between* α *and* β*, and second, the following conditions are fulfilled for all* $\pi \in \mathsf{b}^-(\alpha)$*:*

(λ.same) *For a* var*-labeled node* π *bound within the segment, the corresponding node is bound correspondingly:*

$$\lambda(\pi) \in \mathsf{b}^-(\alpha) \Rightarrow \lambda(c(\pi)) = c(\lambda(\pi))$$

(λ.out) *For a* var*-node bound outside the segment, the corresponding node has the same binder:*

$$\lambda(\pi) \notin \mathsf{b}^-(\alpha) \Rightarrow \lambda(\pi)=\lambda(c(\pi))$$

(λ.hang) *There are no "hanging binders":*

$$\lambda^{-1}(\pi) \subseteq \mathsf{b}^-(\alpha)$$

Parallelism $\sim$ *in a lambda structure* $\mathcal{L}^\theta$ *is the largest relation between segments of* $\mathcal{L}^\theta$ *such that*

$$\alpha \sim \beta$$

implies $\alpha \sim_\lambda \beta$*, and for the correspondence function* c *between* α *and* β *the following conditions are fulfilled for all* $\pi \in \mathsf{b}^-(\alpha)$*:*

(ante.same) *For an* ana*-node bound within the segment, the correspondent has two possible antecedents:*

$$\mathsf{ante}(\pi) \in \mathsf{b}(\alpha) \Rightarrow \mathsf{ante}(c(\pi))=\pi \;\vee\; \mathsf{ante}(c(\pi))=c(\mathsf{ante}(\pi))$$

(ante.out) *If an* ana*-node is bound outside the segment, then its correspondent has the same anaphoric binder:*

$$\mathsf{ante}(\pi) \notin \mathsf{b}(\alpha) \Rightarrow \mathsf{ante}(c(\pi)) = \pi$$

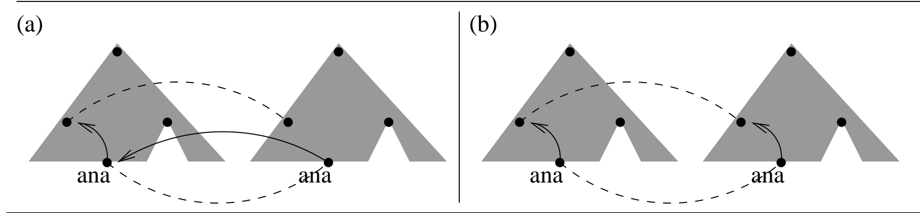


Figure 2.5: Illustrating (ante.same). The dotted arcs stand for correspondence, and the solid arrows stand for anaphoric binding.

The conditions on lambda binding are symmetric, that is, they enforce binding in α to follow binding in β as well as the other way round, while the conditions on anaphoric binding are not symmetric. Figure 2.4 illustrates the conditions (λ.same), (λ.out) and (λ.hang). (Correspondence is represented as a dashed arc.) Figure 2.5 illustrates the two possibilities of binding that condition (ante.same) allows.

2.2 The Constraint Language for Lambda Structures

Now we define the *Constraint Language for Lambda Structures* [42, 41], which is interpreted over lambda structures.

We assume an infinite set $\mathcal{V}ar$ of (node) variables ranged over by X, Y, Z, U, V, W. The abstract syntax of the language CLLS is given in Fig. 2.6: A CLLS constraint is a conjunction of predicates that describe relations between a lambda structure's nodes.[1] A a single such predicate is a *literal*.

φ, ς	$::=$	$X \triangleleft^* Y \mid X{:}f(X_1, \ldots, X_n) \mid X \bot Y \mid X \neq Y$	$(\mathsf{ar}(f) = n)$	(1)
	$\mid$	$X_0/X_1, \ldots, X_n \sim Y_0/Y_1, \ldots, Y_n$	$n \geq 0$	(2)
	$\mid$	$\lambda(X){=}Y \mid \mathsf{ante}(X){=}Y$		(3)
	$\mid$	$\texttt{false} \mid \varphi \ \wedge \ \varsigma$		(4)

Abbreviations: $X{=}Y$ for $X \triangleleft^* Y \wedge Y \triangleleft^* X$ and $X \triangleleft^+ Y$ for $X \triangleleft^* Y \wedge X \neq Y$

Figure 2.6: The Constraint Language for Lambda Structures (CLLS)

For simplicity, we view inequality ($\neq$) and disjointness ($\bot$) literals as symmetric. We also write XRY, where $R \in \{\triangleleft^*, \triangleleft^+, \bot, \neq, =\}$.

A parallelism literal relates two *segment terms* $X_0/X_1, \ldots, X_n$ and $Y_0/Y_1, \ldots, Y_n$, which denote segments. We will use the letters A, B, C, D to denote segment terms:

$$A, B, C, D ::= X_0/X_1, \ldots, X_n \quad n \geq 0$$

If $n = 0$, then the segment is a subtree. A segment term denotes a segment, but it is *not* the case the the i-th hole variable of a segment term has to denote the i-th hole node: Rather, each hole node has to interpret at least one hole variable, and each hole variable has to denote some hole node.

In a lambda binding literal $\lambda(X){=}Y$, X denotes a var-labeled node that is bound at the lambda binder for which Y stands. In an anaphoric binding literal $\mathsf{ante}(X){=}Y$, X denotes an ana-labeled node for which the anaphoric binder is the node for Y.

First order formulas Φ built from constraints and the usual logical connectives are interpreted over the class of lambda structures in the usual Tarskian way. We write $\mathcal{V}ar(\Phi)$ for the set of variables occurring in Φ. If a pair $(\mathcal{L}^\theta, \sigma)$ of a lambda structure $\mathcal{L}^\theta$ and a variable assignment $\sigma : \mathcal{G} \to D_\theta$, for some set $\mathcal{G} \supseteq \mathcal{V}ar(\Phi)$, satisfies Φ, we write this as $(\mathcal{L}^\theta, \sigma) \models \Phi$. Overloading notation a bit, we call both $\mathcal{L}^\theta$ and $(\mathcal{L}^\theta, \sigma)$ a *model* of Φ. We say that Φ is *satisfiable* iff it possesses a model. Entailment $\Phi \models \Phi'$ means that all models of Φ are also models of Φ'.

So the semantics of the language CLLS is given by a class of models: the set of lambda structures.

[1]The relation symbols that CLLS uses are the same as the matching relations on lambda structures. There should be no danger of confusion, as relation symbols are always applied to node variables whereas relations can only be applied to the nodes of a lambda structure.

2.2.1 Sublanguages

The language CLLS can be viewed as a hierarchy of languages:

- The language $\mathcal{C}_{\mathrm{d}}$ of *dominance constraints* consists of lines (1) and (4) of Fig. 2.6.
- The language $\mathcal{C}_{\mathrm{p}}$ of *parallelism constraints* consists of lines (1), (2), and (4) of Fig. 2.6.
- The language CLLS consists of lines (1), (2), (3), and (4) of Fig. 2.6.

This hierarchy of languages $\mathcal{C}_{\mathrm{d}}$, $\mathcal{C}_{\mathrm{p}}$ and CLLS corresponds to a hierarchy in the models: tree structures, tree structures with a parallelism relation, and lambda structures.

2.2.2 Constraint Graphs

We often draw constraints as graphs with the nodes representing variables; a labeled variable is connected to its children by solid lines, while a dotted line represents dominance. For example, the graph for $X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$ is displayed in Fig. 2.7. As trees do not branch upwards, this constraint is unsatisfiable. In these constraint graphs, we represent lambda binding by a curving arrow and anaphoric binding by a straight arrow from the bound variable to the binder.

An important concept related to constraint graphs is that of a *fragment*: It is a tree-shaped subgraph in which all the nodes are connected by solid lines. A fragment has a root and leaves; an unlabeled leaf is called a *hole*. We also call a single unlabeled node a fragment if it is connected to the rest of the graph only by dominance edges. For example, the constraint graph in Fig. 2.7 contains two fragments. The upper fragment comprises X, X_1, X_2, while the lower fragment, consisting of Y, is a fragment of a single unlabeled graph node.

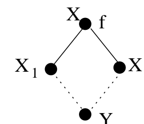


Figure 2.7: An unsatisfiable constraint

A constraint graph does not contain all the information of the constraint that it represents, i.e. there may be literals in the constraint that are not represented in the graph:

- Disjointness and inequality literals are not represented in a constraint graph.
- Parallelism literals are not always represented in a constraint graph. If we want to indicate the segment terms in the graph, we use either a shaded triangle with holes, or a bracket connecting the root variable to the hole variable. Likewise, correspondence is not always represented in a constraint graph. If we want to visualize it, we use a dashed arc.
- A dominance literal $X \triangleleft^* Y$ is shown only if it links two different fragments of the constraint graph.

That means that among others, the following dominance literals are not represented in the constraint: reflexive dominance $X \triangleleft^* X$; a dominance literal $X \triangleleft^* Y$ where X is also the parent of Y by some labeling literal; dominance literals that are in the transitive closure of two or more other dominance edges shown in the graph.

2.2.3 Variations

There exist a number of variations on the definition of CLLS. Several previous papers define segments and segments terms to have exactly one hole [42, 46, 44, 41]. The reason why we consider the more general case is that the application to underspecified beta reduction uses segments with more than one hole.

Of the language $\mathcal{C}_\mathrm{d}$ of dominance constraints, there are two quite different versions: the one we have used here, and a language that allows for set operators on relation symbols [34, 44]. For example, this language contains expressions like $X(\triangleleft^* \cup \perp)Y$, which states that the node interpreting X must be either above the node for Y, or in a disjoint position from it. Note that this is not a disjunction but a single relation. The language with set operators allows for solvers with better propagation. However, these solvers are more complicated than the one we present in Chapter 3, and as our focus is on parallelism, we want to keep our account of dominance constraints as simple as possible.

Some predicates that we include in the definition of CLLS are missing in other accounts, notably disjointness and inequality (which we will need for the processing of parallelism constraints).

For the condition (λ.out) on lambda binding and parallelism (Def. 2.7) the recent overview paper by Egg, Koller and Niehren gives a slightly weaker version [41], which is used for the proper treatment of an ellipsis phenomenon called *antecedent-contained deletion*.

2.3 Modeling Scope, Ellipsis, and Anaphora with CLLS

In this section, we illustrate the language CLLS by discussing three important linguistic phenomena that can be modeled with it: scope ambiguity, ellipsis, and anaphoric binding.

2.3.1 Modeling Scope with Dominance Constraints

Sentence (2.1) (repeated from (1.5) in the introduction) is an example of a *scope ambiguity*: There are two readings of the sentence that only differ in the *scope* of the two *quantifiers* “every plan” and “a catch”. In the representation of these two readings as logical formulas, shown in (2.2) and (2.3), this ambiguity translates to different scopes of the variables x and y.

(2.1) Every plan has a catch.

In the formula in (2.2) "a catch" takes wide scope over "every plan", which gives us the reading where all plans have the same catch; in the formula (2.3) "every plan" takes wide scope, that is, each plan has its own reason for not working.

(2.2) (a catch)(λx
(every plan)(λy
(have x) y))

(2.3) (every plan)(λx
(a catch)(λy
(have x) y))

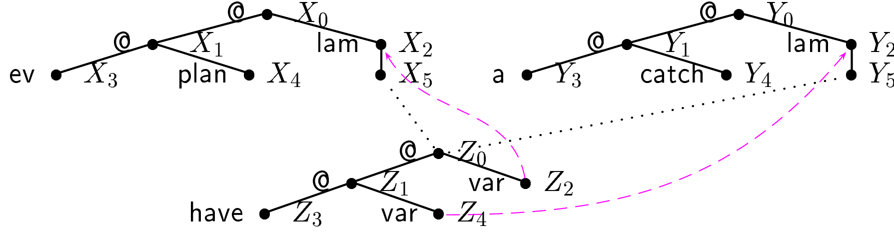


Figure 2.8: Constraint for " Every plan has a catch."

Figure 2.8 shows the constraint representing the meaning of sentence (2.1). It contains three fragments. Roughly speaking, the two upper fragments represent the meanings of the quantifiers "every plan" and "a catch", while the lower fragment represents the meaning of "has". The three fragments are connected only by dominance literals, which connect the leaves X_5 and Y_5 of the two upper fragments to the root Z_0 of the lower fragment.

The constraint graph in Fig. 2.8 is a graphical representation of the following constraint:

$$\begin{array}{l} X_0{:}@(X_1,X_2) \wedge X_1{:}@(X_3,X_4) \wedge X_3{:}\mathsf{every} \wedge X_4{:}\mathsf{plan} \wedge X_2{:}\mathsf{lam}(X_5) \wedge X_5 \triangleleft^* Z_0 \wedge \\ Y_0{:}@(Y_1,Y_2) \wedge Y_1{:}@(Y_3,Y_4) \wedge Y_3{:}\mathsf{a} \wedge Y_4{:}\mathsf{catch} \wedge Y_2{:}\mathsf{lam}(Y_5) \wedge Y_5 \triangleleft^* Z_0 \wedge \\ Z_0{:}@(Z_1,Z_2) \wedge Z_1{:}@(Z_3,Z_4) \wedge Z_3{:}\mathsf{have} \wedge Z_4{:}\mathsf{var} \wedge \lambda(Z_4)=Y_2 \wedge Z_2{:}\mathsf{var} \wedge \\ \lambda(Z_2)=X_2 \end{array}$$

The constraint, let us call it φ for short, states that both X_5 and Y_5 must dominate Z_0. Each model of φ is a tree. Since trees do not branch upwards, either the node denoted by X_5 must dominate the tree node for which Y_5 stands, or the other way round. If we take a closer look at the tree fragments for "every plan" and "a catch", we can see that they cannot *overlap*: If we want to identify some variable Y_i with a variable X_j, we have exactly two possibilities: We could set either $X_0=Y_5$ or $Y_0=X_5$, anything else would make the whole constraint unsatisfiable because the labels conflict. So, as we have said, the fragments of Fig. 2.8 cannot overlap, which means that in each model of φ either the

node for X_5 must dominate the node for Y_0, or the node for Y_5 must be above the one that X_0 stands for.

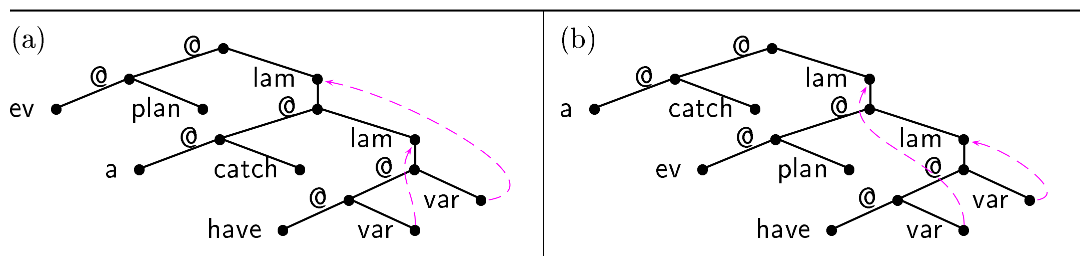


Figure 2.9: Two models of the constraint in Fig. 2.8

Figure 2.9 shows two models of φ that exactly match the the two readings of sentence (2.1). These are not the only models of φ. In fact, it has infinitely many. One reason for this is that CLLS cannot specify the root node of a tree. That means that any tree that has the left or right tree in Fig. 2.9 as a subtree is also a model of φ. Another reason is that when we only know that X_6 dominates Z_0, a segment of arbitrary size can separate them in a model.

The existence of these larger models is indispensable for a treatment of parallelism constraints. For our current example in Fig. 2.8 the intended models only contain nodes that are mentioned in the constraint, i.e. nodes for which the constraint contains a variable. However it is not clear how such a minimality condition could be formulated in the presence of parallelism literals. For example the lambda structure in Fig. 2.11 is the "intended" model of the constraint in Fig. 2.10, but the lambda structure contains nodes that do not interpret any variable of the constraint.

(2.4) Peter began a book.

Moreoever, these larger models are necessary for a CLLS analysis of a linguistic phenomenon called *reinterpretation* [77, 36, 37, 38] as illustrated in sentence (2.4): Peter can only begin an activity; maybe he began to *read* the book, or he began to *write* it, etc. In constructing the semantics of this sentence, the missing activity has to be added to the sentence meaning. So in the end the intended model, the sentence semantics, will contain material (e.g. the representation of "to read") that is not present in the original CLLS constraint that we have put up for the sentence.

2.3.2 Modeling Ellipsis with Parallelism Constraints

An *ellipsis* is a construction in which linguistic material is left out even though it is necessary by either syntactic or lexical conditions. One possible reason for such an omission is that the missing material is already present somewhere else, as for example in sentence (2.5). The elliptical part "so does Mary", the *target sentence*, means "Mary

sleeps" – the missing material is found in the *source sentence* or *antecedent* "every man sleeps". "Every man" is the *source parallel element* or *source contrasting element*, and "Mary" is the *target parallel element* (or *target contrasting element*).

(2.5) Every man sleeps, and so does Mary.

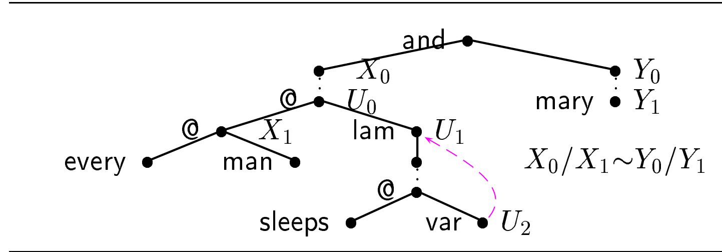


Figure 2.10: Constraint for "Every man sleeps, and so does Mary."

We represent the meaning of the sentence by the constraint in Fig. 2.10. The part of the constraint graph dominated by X_0 represents the meaning of the source sentence "every man sleeps". For the target sentence, we have a representation for "Mary", as well as a parallelism literal: The meaning of the source sentence "every man sleeps" is the same as the meaning of the target sentence, except for the contributions of the two subjects "every man" and "Mary". (Note that in the semantics construction for sentence (2.5), the function words "so does" do not receive any representation at the level of linguistic meaning, except as the parallelism literal $X_0/X_1{\sim}Y_0/Y_1$.)

In each model of the constraint in Fig. 2.10, the segment denoted by X_0/X_1 must be structurally isomorphic to the one denoted by Y_0/Y_1. Furthermore, the lambda binding $\lambda(U_2){=}U_1$ links U_2 to a binder within X_0/X_1, so if U_2 denotes a node π, then binding for the correspondent of π must obey condition (λ.same) of Def. 2.7. Figure 2.11 shows a model of the constraint in Fig. 2.10; again it is the intended one.

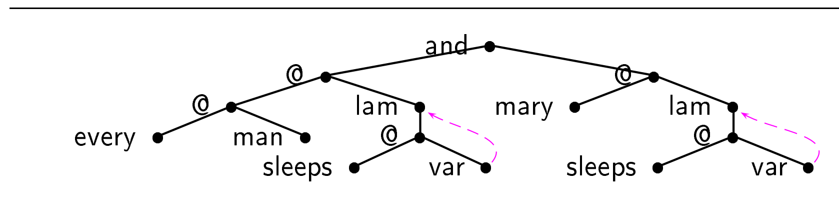


Figure 2.11: A model of the constraint shown in Fig. 2.10

2.3.3 Interaction of Parallelism and Scope: Quantifier Parallelism

Why is it interesting to deal with parallelism phenomena within a formalism for underspecified semantics? The point is that scope ambiguity and parallelism are not independent phenomena; they interact in quite interesting ways, a fact that first became apparent

in sentences that Hirschbühler [62] proposed. This phenomenon, *quantifier parallelism*, is exemplified in sentence (2.6), repeated from (1.13).

(2.6) Every linguist attended a workshop. Every computer scientist did, too.

The source sentence of (2.6) contains two scope-bearing elements, "ever linguist" and "a workshop", and thus has two readings. The meaning of the target sentence is "...every computer scientist attended a workshop", two scope-bearing elements again, so the sentence should have 4 readings all in all. But it only has three: either "a workshop" takes scope over both the source and the target sentence, which means that one single workshop is attended by everybody; or "a workshop" has wide scope within both the source and the target sentence, which means that all linguists attend one common workshop, and all computer scientists visit a (potentially different) common workshop; or "a workshop" takes narrow scope in both sentences, which means that everybody is travelling to his or her own workshop. This means that parallelism enforces a parallel resolution of scope ambiguities.

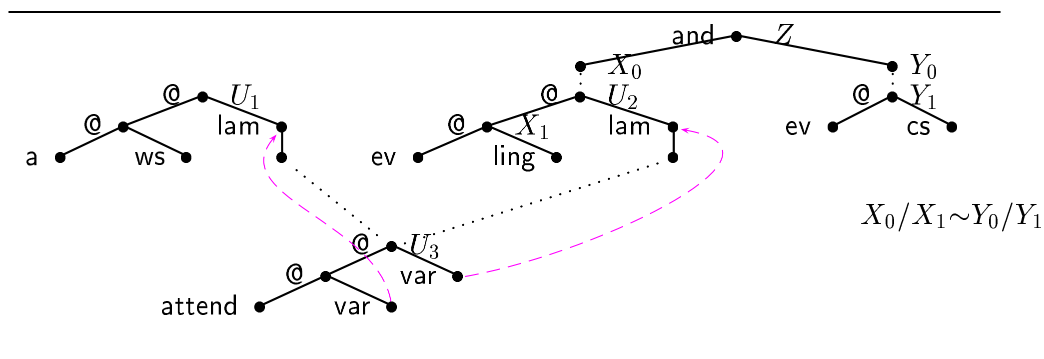


Figure 2.12: Constraint for sentence (2.6): "Every linguist attended a workshop. Every computer scientist did, too."

Let us see how CLLS handles this interaction. Figure 2.12 shows the CLLS constraint for sentence (2.6). X_0 is the root of the source sentence semantics, and Y_0 is the root of the target sentence semantics. The contrasting elements, "every linguist" and "every computer scientist", are dominated by X_0 and Y_0 respectively. But the constraint does not contain any dominance literal between X_0 and U_1: Most quantifiers, including universal quantifiers, must not move outside their sentence (i.e. they are dominated by the uppermost node of their sentence); this is called a *scope island constraint.* But this constraint does not hold for indefinite quantifiers: the fragment for "a workshop" is allowed to dominate Z.

There are three possible positions for the fragment for "a workshop". It can either be dominated by the fragment for "every linguist", or it can go between X_0 and U_2, or it can dominate Z. In either case, the parallelism literal enforces structural isomorphism between the two segments interpreting X_0/X_1 and Y_0/Y_1. This gives us the three correct

readings of the sentence, sketched in Fig. 2.13. Note that while in Fig. 2.13 (a) and (b), lambda binding is parallel according to condition (λ.same), in (c) the binder is above both parallel segments and thus binds both a node in the source segment and its correspondent in the target segment according to (λ.out).

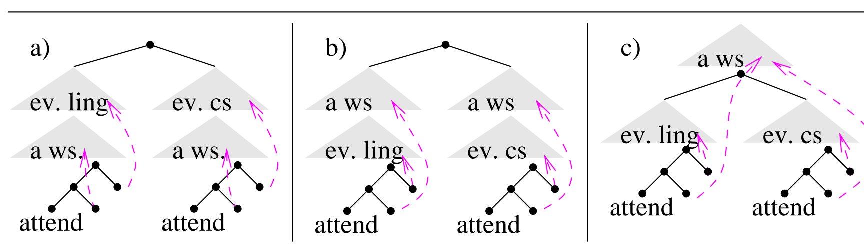


Figure 2.13: Sketch of three models for the constraint in Fig. 2.12

2.3.4 Modeling Anaphoric Binding with Binding Constraints

An anaphor is an element or a construction that, in order to be interpreted, needs to be associated with something else in the context. A few examples are given in sentences (2.7) through (2.9). In these examples, the anaphora are underlined, and they are associated with the italicized part of the sentence.

(2.7) *Mary* likes <u>her</u> cat.

(2.8) *Mary and Sue* like <u>each other</u>.

(2.9) Some cats who *shred carpets* <u>do so</u> repeatedly.

Coreference between an anaphoric element and its *antecedent*, i.e. the element that it is associated with, is often indicated by *coindexing* the two elements, as shown in sentence (2.10), where "Mary" and "her" are coindexed. So the common index 1 means that the "her" in (2.10) refers to "Mary".

(2.10) Mary$_1$ likes her$_1$ cat.

In lambda structures, coreference can be expressed by the *anaphoric binding function* ante of Def. 2.6. Figure 2.14 shows an example of a lambda structure with anaphoric binding: $\mathsf{ante}(\pi) = \psi$. This lambda structure is the semantics for sentence (2.10).[2] The semantic contribution of "her" is the ana-labeled node along with the anaphoric binding from that node to the node labeled "mary".

[2]In the lambda structure in Fig. 2.14, the representation of "Mary" is just "*mary*", while in Fig. 2.11, the representation of "Mary" is "$\lambda P.P(mary)$". We use the simpler form here just for reasons of simplicity, since it does not make a real difference: In both cases the completely beta reduced term is the same.

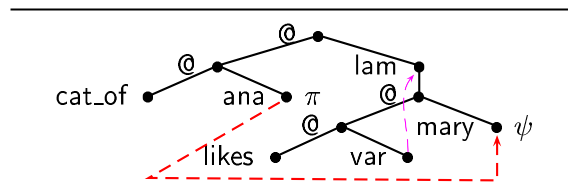


Figure 2.14: Lambda structure for sentence (2.10). Note the anaphoric binding.

2.3.5 Interaction of Parallelism and Anaphora: Strict/Sloppy Ambiguities

Again, anaphora and parallelism are not separate phenomena. An anaphor in the source sentence of an ellipsis can lead to what is called a *strict/sloppy ambiguity.* Consider sentence (2.11). As before, coindexing indicates that the "her$_1$" in the ellipsis source sentence refers to "Mary$_1$". The meaning of the elliptical target sentence is "... and Sue likes her cat". But what does the "her" in the target sentence refer to? There are two possibilities: It can refer either to "Mary" or to "Sue". The reading in which Sue likes Mary's cat is called *strict*, and the reading in which Sue likes her own cat is called *sloppy*.

(2.11) Mary$_1$ likes her$_1$ cat, and Sue does, too.

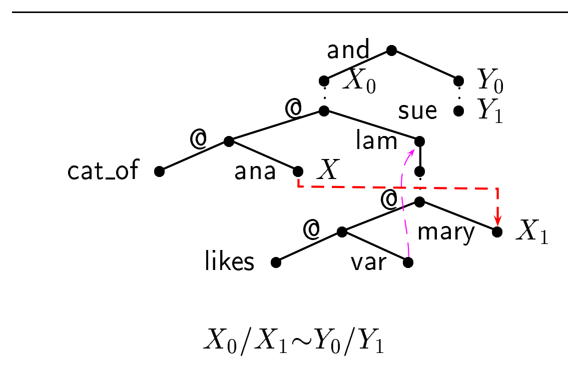


Figure 2.15: The constraint for sentence (2.11): "Mary$_1$ likes her$_1$ cat, and Sue does, too."

Handling the interaction of parallelism and anaphoric binding is exactly what the conditions (ante.same) and (ante.out) of Def. 2.7 are about. We can illustrate condition (ante.same) on the example we have just been discussing, sentence (2.11). A CLLS constraint representing the meaning of this sentence is shown in Fig. 2.15. Here, condition (ante.same) licenses two possible anaphoric binders for the correspondent of X: either the correspondent is bound by X – this gives the strict reading –, or it is bound by the correspondent of X_1, which leads to the sloppy reading.

Fig. 2.16 shows two models of Fig. 2.15 that reflect the two readings of the sentence. For the sloppy reading (picture (b)), the anaphoric reference can be read off in a straightfor-

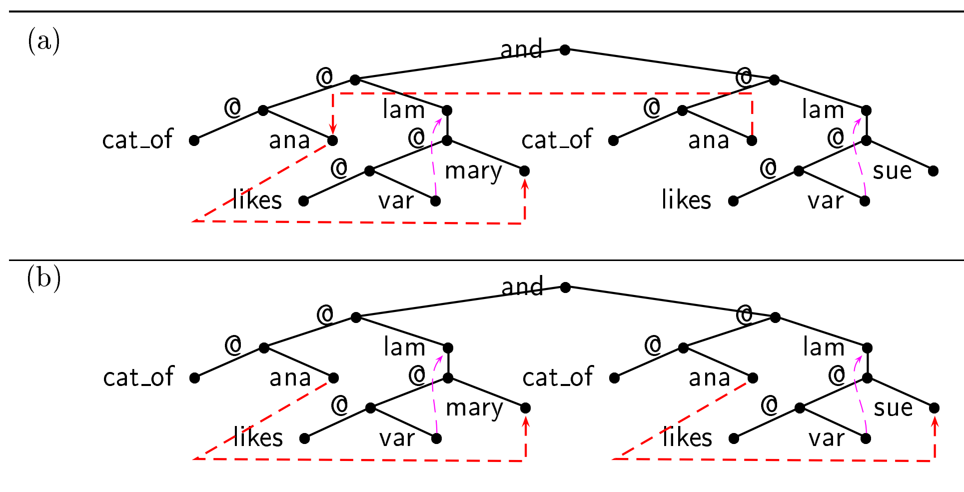


Figure 2.16: Two models for the constraint in Fig. 2.15

ward way. For the strict reading in picture (a), we get a *chain* of anaphoric links, which we have to follow to the end to find the referent: The ana-labeled node in the target subtree is mapped to the ana-node in the source subtree, which again is mapped to the mary-node.

Why is the target ana-node not linked directly to the mary-node in the source subtree? The reason is that there are more complicated cases, called *many-pronoun puzzles*, like (2.12) and (2.13), where a simpler analysis would fail. See Egg, Koller and Niehren [41] for a discussion of sentence (2.12), and Dalrymple, Shieber and Pereira [30] for (2.13). Xu [118] proposed the link chain analysis for CLLS and showed that it derives the right readings for many-pronoun-puzzles like (2.12) and (2.13).

(2.12) John revised his paper before the teacher did, and so did Bill. [52]

(2.13) John realizes that he is a fool, but Bill does not, even though his wife does. [29]

2.3.6 Interaction of Scope, Parallelism, and Anaphora

In sentence (2.14), we have an interaction of ellipsis with both scope and anaphora. The CLLS constraint for this sentence is shown in Fig. 2.17. For the sake of readability, we have abbreviated the semantics of “a book she liked” as an empty triangle with a ana-labeled variable Z. The definition of the parallelism relation and its conditions on binding ensure that the scope ambiguities are resolved the same way in source and target sentence, and that we only get the “right” anaphoric bindings. This leads to the three correct readings sketched in Fig. 2.18: Either both Mary and Sue read the same book, or each reads a book that she herself likes, or they read different books that Mary likes. A more extensive discussion of sentence (2.14) and the derivation of the three readings can be found in the recent overview paper on CLLS by Egg, Koller and Niehren [41].

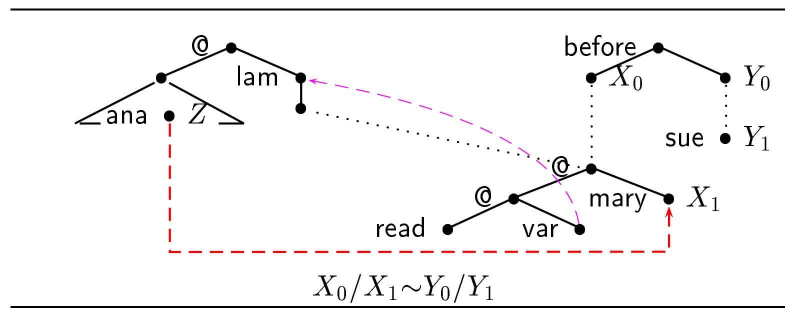


Figure 2.17: Constraint for sentence (2.14)

(2.14) Mary read a book she liked before Sue did.

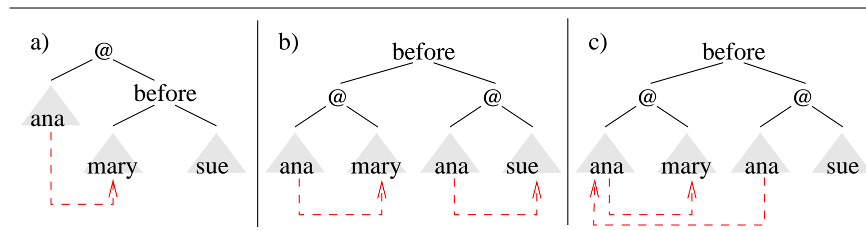


Figure 2.18: A sketch of the three readings of sentence (2.14)

2.3.7 A Note on Models and Readings

What does it mean for a CLLS constraint to represent the semantics of a sentence? For each sentence that we have discussed in this section and the constraint that we put up for it, we have been able to point out models that represent the correct readings of the sentence. But on the other hand we have said that each CLLS constraint has infinitely many models. How are the "intended" models distinguished from all the others?

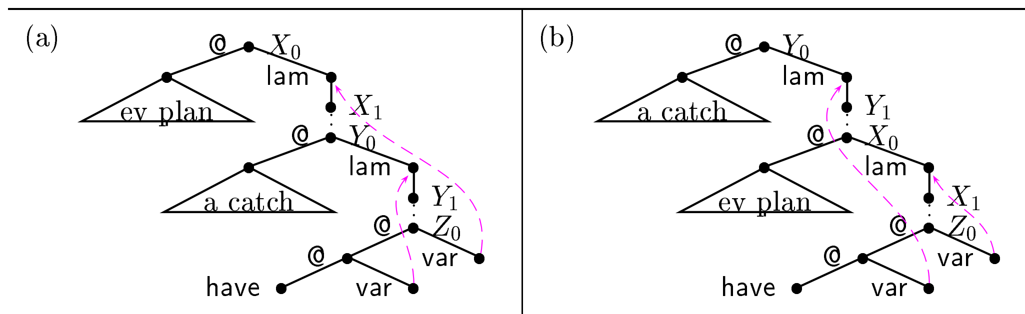


Figure 2.19: Sketch of the two saturated constraints computed for Fig. 2.8

In the following chapters, we present a semi-decision procedure for CLLS. This procedures computes result constraints for a given input constraint, called *saturations*, which

look almost like lambda structures. For example, for the constraint in Fig. 2.8 (p. 34), which represents the meaning of "Every plan has a catch", the procedure computes two saturations sketched in Fig. 2.19. For the sake of readability, we have abbreviated the semantics of "every plan" and "a catch" by empty triangles. Compare the two constraints with the two "intended models" in Fig. 2.9 (p. 35): If we take the constraint in Fig. 2.19 (a), remove the dominance edge between X_1 and Y_0 and identify the two variables, and if we do the same with Y_1 and Z_0, then we get a graph that looks exactly like the lambda structure in Fig. 2.9 (a). In the same way, we can transform the constraint graph in Fig. 2.19 (b) by "contracting dominance edges", and the result looks exactly like the lambda structure in Fig. 2.9 (b). This idea of "contracting dominance edges" is illustrated again in Fig. 2.20.

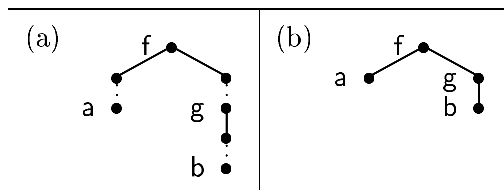
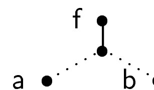


Figure 2.20: "Contracting dominance edges" on (a) yields (b).

For cases like those that we have discussed in this section, the CLLS procedure allows us to compute readings. Given the CLLS constraint representing the meaning of such a sentence, its saturations that we compute will exactly match the correct readings, in the sense that we have sketched above: If we contract dominance edges in the graphs of the saturations, the result looks like the lambda structures that are the correct readings.

Incidentally, it is not the case that *all* saturations that the procedure could ever compute have this property that we can just contract some dominance edges and reach constraint graphs that look like lambda structures. In the constraint graph shown to the right, there is a node with two distinct "dominance children" that cannot be identified. But as far as I know, such constraints never occur as saturations of linguistically relevant constraints.



2.3.8 A Note on Capturing

Usually variable binding in lambda terms is accomplished by variable names: A binder λx binds all occurrences of the variable x in its scope. But if by some operation, another λx gets inbetween the first λx and an occurrence of x, it *captures* the variable.

The usual way of excluding capturing is via freeness conditions. However, this is problematic in the case of underspecified descriptions of lambda structures. Suppose we use variable names to indicate binding, as follows: We encode the variable names into the labels by using new labels lam_x and var_x for each object-level variable x.

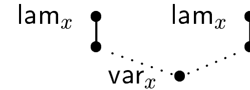


Figure 2.21: Lambda binding?

Now consider the constraint graph in Fig. 2.21. It contains *two* binders lam_x with a scope ambiguity between them, and it is unclear which of the two is supposed to bind the var_x. In each model, the lowest binder labeled lam_x wins. So when the structure of the lambda term is not fully known, variable names are not sufficient to make it unequivocally clear which binder binds which variable occurrences. The problem is compounded in the presence of parallelism literals.

The solution that lambda structures offer, indicating variable binding not by variable names but by a binding function, provides a general solution to the problem, a solution which does not involve any overhead in processing.

The explicit binding functions of lambda structures are somewhat similar to *de Bruijn indices* [31]. In this notation, a bound variable is represented by a number. A number n means that to reach the binder of this variable, n other λ-abstractions have to be passed on the path up from the variable to the root. For example, the term $\lambda x.x(\lambda y.xy)$ is written as $\lambda 0(\lambda 10)$. However, every time a lambda term changes, e.g. by beta reduction, de Bruijn indices have to be readjusted.

2.4 Related Formalisms

In this and the following section, we discuss work related to CLLS. This section is devoted to related *formalisms*, i.e. tree description languages, and the following section names some related *approaches* to modeling scope, parallelism, and anaphoric binding.

Three criteria for distinguishing between tree description languages will be especially relevant for our purposes:

Feature trees vs. constructor trees. In a feature tree, each child of a node can be addressed individually by the *feature*, i.e. the edge label, of the edge leading to it.

A constructor tree is a node-labeled tree corresponding to a ground term. The children of a node can be accessed all at the same time, but not individually. Feature trees are more general than constructor trees: A constructor tree can be seen as a feature tree in which the edge labels are natural numbers.

CLLS describes constructor trees, as does context unification. All other formalisms that we mention in this section describe feature trees.

Talking about nodes vs. talking about trees. This point concerns the perspective on trees that a language takes: it can either take an *internal* perspective, talking about the *nodes* of a single tree, or take an *external* perspective, describing *relations between trees*. In the computer science tradition, languages that talk about trees are more common. In the computational linguistics field, the node-centered paradigm is prevalent.

In CLLS, the variables stand for nodes, so we focus on "node description" languages in this section, discussing only one language that takes the external perspective of relations between trees, namely context unification.

Occurrences vs. Structure. Does identity mean identity of structure, or identical occurrence?

More concretely, suppose we have a language in which the variables denote trees, and equality is equality of structure. Then the equation $x = f(y, y)$ is satisfiable: x is a tree with root label f and two identical subtrees as its children. However, CLLS is a language that talks about *occurrences* of nodes. Here the constraint $X{:}f(Y, Y)$ is unsatisfiable – the node interpreting Y would have to be in a disjoint position from itself.

2.4.1 Second-order Monadic Logic

Most tree description languages coming from the computer science tradition adopt the external perspective – the most notable exception being (W)SkS [113, 114]. SkS is the second-order monadic logic with k successors, and WSkS is its weak variant. The decidability of (W)SkS is due to famous results by Doner, Thatcher and Wright and Rabin [113, 32, 98]. Doner, Thatcher and Wright linked definability in WSkS to recognizability by finite tree automata. Rabin showed that definability in SkS coincides with recognizability by Rabin tree automata over infinite trees. The languages SkS and WSkS are among the most expressive decidable logics. The time complexity of SkS is non-elementary.

The language (W)SkS possesses first-order variables $x, y, z \ldots$ and second-order monadic variables $X, Y, Z \ldots$. *Terms* are formed from the constant ε, first-order variables, and right concatenation with the unary function symbols $1 \ldots k$. For example, x1523, ε4112 are terms. *Atomic formulas* Atomic formulas are equations and inequations $t_1 \leq t_2$ between terms, and expressions "$t \in X$" for terms t and second-order variables X. Formulas are built from atomic formulas using the usual logical connectives and existential and universal quantification over both first-order and second-order variables. While second-order variables range over arbitrary sets in SkS, they are restricted to ranging over finite sets in WSkS.

The interpretation that is interesting for our purposes interprets terms as strings in $\{1, \ldots, k\}^*$, ε as the empty word, $=$ as string equality, and $\leq$ as the prefix ordering. Second-order variables are interpreted as sets of strings. The atomic formula $x \in X$ is true iff the denotation of x is contained in the denotation of X.

How can this language be used to encode trees? In the tree definition that we have used throughout, a path is a word in $\{1, \ldots, k\}^*$. In this encoding, ε is the root node, wi is the i-th child of the node denoted by the term w, and the prefix ordering $\leq$ on terms of (W)SkS is the same as dominance between tree nodes. To encode a tree domain, we need to be able to state prefix-closedness and closedness under left brother. Both properties

can be expressed in WSkS.

The main problem in encoding node-labeled trees is the encoding of the node labels. Different encodings are possible. Koller, Niehren and Treinen [78] encode a tree as a single set, its tree domain. The labels of the nodes are represented by special words: Koller, Niehren and Treinen use unary function symbols $0, \ldots, k$ instead of $1, \ldots, k$ and represent the fact that a node π is labelled f by a word $\pi 0^n$ if f is the n-th function symbol with this particular arity. Comon et al.[23] also encode a tree as its tree domain, but they encode labeling by one set S_f for each function symbol f in the signature Σ (which requires Σ to be finite). The set S_f contains all tree nodes labeled by the function symbol f.

SkS adopts the internal, node-centered perspective on trees; although it can talk about sets of nodes, there are some simple relations between trees that it cannot express. For example, suppose x and y are tree-valued variables, then the equation $x = f(y, y)$ cannot be expressed in SkS because this property of having two identical subtrees is one that can only be tested by stronger tree automata. There is even a simpler example: Suppose x, y, z are tree-valued variables, then the equation $x = f(y, z)$ cannot be expressed in SkS: If we extend the language such that it allows for concatenation wi of terms w with letters i to the right as well as concatenation iw of terms with letters to the left, then it becomes undecidable. This fact is discussed e.g. in an overview article by Thomas [114] and in an article by Müller and Niehren [90].

2.4.2 Feature Description Languages

Feature descriptions can be regarded as a logical description of records. A feature system is an algebraic structure defined in terms of a set A of *sorts* and a set L of *features*. Intuitively, it may be seen as a graph in which nodes are labeled with sorts (where one node may be of more than one sort), each edge is labeled by a feature, and different outgoing edges of a node are always labeled by different features. A node is addressed from another node by the feature word on a path between them.

Feature descriptions originated in phonology [20] and became a widespread formalism for linguistic theories in the 70s with Lexical-Functional Grammar [68, 65]. An influential feature description language is the one by Kasper and Rounds [66, 67]. The description language by Kasper and Rounds contains, among others, the following atomic formulas: constants $a \in A$, path equations $p \doteq q$ for $p, q \in L^*$, and feature prefixes $\ell : \varphi$ for a feature $\ell \in L$ and a formula φ. A Kasper-Rounds formula is evaluated at a fixed node π of a feature system. A constant a states that this node is of sort a, a path equation $p \doteq q$ says that the two paths $p, q \in L^*$ originating in π are equal, and the formula $\ell : \varphi$ states that the formula φ holds at the node reached from π via the feature $\ell \in L$.

Blackburn [8] investigates the modal nature of the Kasper-Rounds language. In his formalism a sort a becomes a propositional formula φ_a, and a prefixed formula $\ell : \varphi$ becomes a modal $\langle l \rangle \varphi$. In this context, the work of Blackburn and Meyer-Viol [9] and Kracht [79] is especially interesting for our purposes: They investigate modal logics for

finite k-ary trees, a restriction of feature systems to finite *feature trees*, and their tree description languages include a modal for dominance. The basic idea is to use the tree as the reachability relation and to provide modal operators for traveling in the tree. In the notation of Blackburn and Meyer-Viol, a formula $\downarrow^k \phi$ means that ϕ is true at the tree node that is the k-th child of the current node. Also there are operators $\uparrow$ (true at the parent), $\downarrow^*$ (true at some node dominated by the current node) and $\uparrow^*$ (true at some node dominating the current node). For a propositional logic enriched with these modal operators, validity is decidable. Blackburn and Meyer-Viol also discuss a combination of this logic with another modal logic describing feature structures attached to each tree node, which is viewed as an internal structure of the propositions in the modal tree description language.

Smolka [110] studies a feature description language as a constraint language interpreted over a *feature algebra*, which can be seen as a special case of feature systems: a restriction, among other things, to finite, rooted, connected graphs. The constraint language is similar to Kasper-Rounds, but allows for quantification over node variables. For a fragment of the language, Smolka proposes a constraint solver. Building on this work, Backofen, Smolka, Aït-Kaci, and Podelski [7, 1] define a constraint system FT of feature trees. This approach takes an external perspective on trees. A sort constraint Ax says that the tree x has a root of sort A, a feature constraint xfy states that x has a subtree y at feature f, and an equality constraint $x \doteq y$ says that x and y have the same shape. Satisfiability of this language is decidable; a constraint solver (in the shape of simplification rules) is given.

2.4.3 Context Unification

Context unification [84, 85], a variant of second-order linear unification, is the closest relative of the language $\mathcal{C}_\mathrm{p}$. However, while $\mathcal{C}_\mathrm{p}$ adopts the internal perspective on trees, CU takes the external perspective, specifying relations between trees.

Formally, CU can be defined as equation solving in the two-sorted algebra of trees and contexts. A *context* γ over the signature Σ is a tree over the extended signature $\Sigma \uplus \{\bullet\}$ that contains exactly one occurrence of the constant $\bullet$.[3] The *hole* of a context γ is the unique path $\pi \in D_\gamma$ such that $L_\gamma(\pi) = \bullet$. Alternatively, a context γ can be regarded as a function mapping trees to trees: In mapping a tree θ to the tree $\gamma[\theta]$, the occurrence of $\bullet$ in γ is replaced by θ, i.e. $\gamma[\theta] = \gamma[\theta/\bullet]$.

The *algebra of trees and contexts* over Σ is a two-sorted algebra, the domains of which are the set of trees and the set of contexts over Σ. The operations provided by this algebra are tree construction and functional application of contexts to trees. For each sort of the algebra, we assume an infinite set of variables: a set $\mathcal{V}_1$ of *tree variables* x, y, z, and a set $\mathcal{V}_2$ of *context variables* C. A *tree-valued term* t is built from tree variables, applications

[3] However there exists another variant of CU, studied by Lévy [84], that allows an arbitrary number of leaves labelled $\bullet$ in a tree. This variant is equal to CU in expressivity.

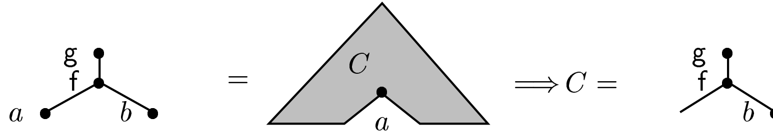


Figure 2.22: $g(f(a,b)) = C(a)$

of function symbols in Σ, and application of context variables.

$$t ::= x \mid C(t) \mid f(t_1, \ldots, t_n) \quad f \in \Sigma, \mathsf{ar}(f) = n$$

A *CU equation system* is a finite conjunction $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n$ of equations between tree-valued terms, which is interpreted over the algebra of trees and contexts; Tree variables are mapped to trees, and context variables to contexts. A mapping σ is a *solution* of a CU equation system if $\sigma(t_i) = \sigma(t'_i)$ for all equations $t_i = t'_i$ in the system.

For instance, the CU equation

$$g(f(a,b)) = C(a)$$

has exactly one solution: C must be mapped to the context function $\lambda X.g(f(X,b))$ (Fig. 2.22). This context function corresponds to the context $g(f(\bullet, b))$.

One interesting point about CU is that with respect to its expressiveness it is situated between *string unification* [86], which is decidable, and second-order unification, which is undecidable (see Fig. 2.23). For CU itself, decidability is still an open problem [104].

second order unification	undecidable	see [54]
context unification	(unknown)	see [104]
string unification	decidable	see [86]

Figure 2.23: Context unification in context

String unification is the problem of solving word equations. A *string unification equation system* is a conjunction of equations $w_1 = v_1 \wedge \ldots \wedge w_n = v_n$, where w_i, v_i, $1 \leq i \leq n$, are words over some alphabet $\Sigma \cup \mathcal{V}$. Σ is a set of terminals and $\mathcal{V}$ a set of variables. A solution is a valuation $\sigma : \mathcal{V} \to \Sigma^*$ such that $\sigma(w_i)$ and $\sigma(v_i)$ is the same word over Σ for all i. For example, $gx = xg$ is a string unification equation. All solutions of this equation map x to a word in g^*. String unification has been discovered and studied under several names and in several research contexts [6]. There exists a (very complicated) algorithm for it due to Makanin [86]. Context unification can be regarded as a generalization of string unification from words to trees.

How exactly is CU related to CLLS? CU is equally expressive as a fragment of CLLS that consisting of labeling literals, and parallelism literals with exactly one hole. We write $\mathcal{C}_{\text{lp}}$ for this language. On the one hand, every CU equation system can be encoded in *equality up-to constraints* [93], which can be translated into $\mathcal{C}_{\text{lp}}$ constraints [92]. On the other hand, any conjunction of labeling, dominance, and parallelism constraints can be

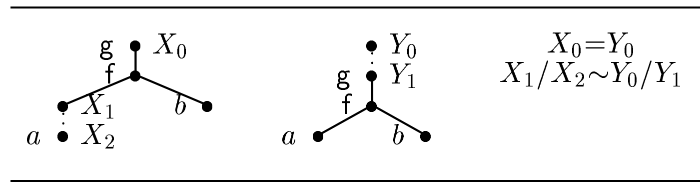


Figure 2.24: Constraint for the equation $g\big(f(C(a), b)\big) = C\big(g(f(a, b))\big)$

written as a CU equation system [92]. Note that dominance literals can be expressed in $\mathcal{C}_{\mathrm{lp}}$: A dominance literal $X \triangleleft^* Y$ can be written as the parallelism literal $X/Y{\sim}X/Y$.

The similarity between contexts and segments is immediately obvious. However, there is an important difference. A context is a function from trees to trees, and as such can be regarded independently of any tree it might be embedded in. A segment is embedded in its surrounding tree by the binding conditions. There are only segments *of* a tree, not segments on their own.

To illustrate how CU equation systems can be translated into CLLS constraints and vice versa, we sketch simpler encodings than the ones given in the literature [93, 92]. Consider the CU equation

(2.15) $g(f(C(a), b)) = C(g(f(a, b)))$.

We translate this equation into the $\mathcal{C}_{\mathrm{lp}}$ constraint in Fig. 2.24, as follows:

- The left-hand side term of the equation (2.15) can be divided into three parts: the context $g(f(\bullet, b))$, the context variable C, and the tree a. Likewise, the right-hand side term can be divided into two parts: the context variable C, and the tree $g(f(a, b))$.

  The parts that do not consist of a context variable can be translated to labeling literals in a straightforward fashion: The context $g(f(\bullet, b))$ is translated to the segment X_0/X_1 of the constraint in Fig. 2.24, the tree a is translated to the segment $X_2/$ of the constraint, and the tree $g(f(a, b))$ is translated to the segment $Y_1/$ of the constraint.
- The two occurrences of the same context variable C are translated into a parallelism literal: $X_1/X_2{\sim}Y_0/Y_1$.
- Finally, the left-hand side term and the right-hand side term of the equation (2.15) describe the same tree. We translate this by equating the roots of the two constraint graphs that we have drawn: $X_0{=}Y_0$.

Conversely, how can we encode a $\mathcal{C}_{\mathrm{lp}}$ constraint in a CU equation system? The main problem is that the two languages adopt different perspectives on trees: How can we encode information about a specific node when we can only talk about trees? The trick

in overcoming this obstacle is to employ contexts to talk about nodes: We use, for each variable X in our $\mathcal{C}_{\text{lp}}$ constraint, a context variable C_X standing for the "tree from the root down to X". Additionally, we use a tree variable r to denote the entire tree.

To make the encoding simpler, we allow, besides equations between tree-valued terms, equations $s = s'$ between context-valued terms. This adds nothing to the expressive power of the language [92]. Then we can translate constraints into equations as follows:

$$
\begin{array}{rll}
X \triangleleft^* Y & \text{as} & \exists C.\big(C_Y = C_X \circ C\big) \\
X{:}f(X_1, \ldots, X_n) & \text{as} & \bigwedge_{1 \leq i \leq n} \exists x_1, \ldots, x_n. \qquad\qquad \text{if } n \geq 1 \\
 & & \quad \big(C_{X_i} = C_X \circ f(x_1, \ldots, x_{i-1}, \bullet, x_{i+1}, \ldots, x_n)\big) \\
X{:}a & \text{as} & r = C_X(a) \\
X_1/X_2 {\sim} Y_1/Y_2 & \text{as} & \exists C.(C_{X_2} = C_{X_1} C \wedge C_{Y_2} = C_{Y_1} C)
\end{array}
$$

Furthermore, to make sure that a context variables C_X correctly encodes the position of X, we use the following conjunction of equations:

$$
\mathit{Root}(\varphi) =_{\text{def}} \bigwedge_{X \in \mathcal{V}ar(\varphi)} \exists x. r = C_X(x)
$$

That is, there exists a single tree r such that all C_X encode positions in it.

For example, the unsatisfiable constraint $\varphi = Y{:}f(X_1, X_2) \wedge X_1 \triangleleft^* X \wedge X_2 \triangleleft^* X$ is translated to the CU equation system

$$
\mathit{Root}(\varphi) \wedge C_{X_1} = C_X \circ f(\bullet, x_2) \wedge C_{X_2} = C_X \circ f(x_1, \bullet) \wedge C_Y = C_{X_1} \circ C \wedge C_Y = C_{X_2} \circ C'.
$$

This system is unsatisfiable: from $C_Y = C_{X_1} \circ C$ and $C_Y = C_{X_2} \circ C'$, we get $C_{X_1} \circ C = C_{X_2} \circ C'$. We can substitute C_{X_1} by $C_X \circ f(\bullet, x_2)$ and C_{X_2} by $C_X \circ f(x_1, \bullet)$, which gives us $f(\bullet, x_2) \circ C = f(x_1, \bullet) \circ C'$. This is clearly unsatisfiable as the holes are in different positions on the two sides.

Note that in the translation that we have just sketched, both dominance literals and parallelism literals were encoded into the same CU construct, context variables. Dominance constraints can be encoded into *stratified CU*, a decidable fragment of CU [108, 109], but they do not seem to correspond to any clear-cut fragment of stratified CU. So in CLLS we have two different language constructs, dominance constraints and parallelism constraints, that differ in their expressive power and thus also the algorithmic complexity of procedures for solving them; in CU these two correspond to just one construct, which matches the expressive power of parallelism constraints. This point will become important below, when we talk about approaches to modeling scope and ellipsis with CU.

2.5 Related Modeling Approaches

In this section we discuss related approaches to modeling scope, parallelism, and anaphoric binding.

2.5.1 Linguistic Applications of Dominance

In computational linguistics, the dominance relation has been widely used in analyses of both the syntax and the semantics of natural language. Its use in computational linguistics was first proposed in the 80's by Marcus, Hindle and Fleck [87]: The aim of *D-Theory* parsing was a cognitively adequate handling of local ambiguities, i.e. cases where after the first few words of a sentence there is more than one possible analysis, but the ambiguity is resolved by the end of the sentence. The parsing algorithm should not backtrack in cases where humans showed no hesitation in reading the sentence. To that end, instead of working on a set of parse trees, D-theory uses a single underspecified tree representation allowing for labeling and dominance statements. In this framework, each underspecified description has to have a unique *standard referent* (model). Later papers add further constraints for a closer modeling of human sentence understanding, for example precedence, the left-of relation between nodes of a tree [55, 112, 16].

Dominance is also used for a variant of *Tree Adjoining Grammars* [64], a grammar formalism that constructs a parse tree by *adjoining* further trees to it. While the original formulation of this operation uses destructive changes to a tree, Vijay-Shanker describes it as a monotonous adding of information to *quasi-trees* [115, 102], tree descriptions consisting of labeling and dominance information. A quasi-tree must have a unique minimal tree that satisfies it. In contrast, this condition is not imposed on *D-Trees*, introduced by Rambow, Vijay-Shanker and Weir [99].

To model scope ambiguities, dominance was first used by Reyle in the early 90's [100]: *UDRT* is the underspecified variant of Discourse Representation Theory (DRT), a formalism for natural language semantics that focuses on anaphoric reference and accessibility conditions for it, modeling accessibility by stacked *boxes* containing formulas and referents (elements that can be referred to anaphorically). UDRT adds two constructs to DRT: There may be labels attached to boxes, and between these labels, dominance statements can be expressed.

Muskens [91] applies the same technique – labeled formulas, and dominance between labels – to both the (syntactic) parse trees and the lambda terms of a sentence's semantics.

With *Predicate Logic Unplugged*, Bos [15] generalizes the approach to a meta-formalism to be combined with any object-level language. Again, formulas of the object-level language are labeled, and label variables (holes) can replace a formula. Dominance constraints state that some hole must dominate some label. A solution of such an expression is a *plugging*, a mapping from holes to labels that respects the dominance constraints.

In *Minimal Recursion Semantics*, Copestake, Flickinger and Sag [26] use similar techniques as in Predicate Logic Unplugged: A *handle* is either a label preceding a formula or a label variable, and dominance between handles is expressed by the *qeq* relation $=_q$, "equality modulo quantifiers": Either the label variable is directly identified with the formula label that it dominates, or or one or more quantifiers float in between the label variable and the formula label.

Pinkal [95] distinguishes three levels of semantic underspecification. On the first level are lexical ambiguities, referential ambiguities and similar local phenomena. Scope ambiguities constitute the second level, the level of underspecification in the global semantic structure. The third level of underspecification arises when the syntactic information from which the semantic representation is built is incoherent, ambiguous or incomplete. Pinkal proposes a higher-order unification formalism called *Underspecified Semantic Description Language (USDL)* to deal with phenomena at all three levels. The language USDL is a variant of CU. Subsequently Niehren, Pinkal and Ruhrberg proposed a CU treatment of both scope ambiguities and parallelism phenomena [94], which Egg and Kohlhase extended by a dynamic treatment of referents [40]. However this analysis runs into problems of combinatoric explosion when many scope-bearing elements are present [74]. The problem is that this approach has to use context variables in their full expressivity for expressing scope ambiguity. As we have pointed out above when we sketched the translation of CLLS constraints to CU equation systems, there is no obvious translation of dominance constraints that would be computationally cheaper than CU in general. In contrast, the CLLS analysis, which replaced the CU analysis, has the distinction between dominance constraints and parallelism constraints, so it can use the less "expensive" formalism for modeling scope ambiguity.

Duchier and Gardent propose using dominance constraints for an underspecified representation of discourse structure [33]. Ambiguities pertaining to the relation between different discourse elements can be represented in the same way as scope ambiguities. A similar approach is taken by Schilder [107].

2.5.2 Related Analyses of Parallelism

In this section we list only the analyses of parallelism phenomena that are most closely related to the CLLS approach; a general discussion of ellipsis and of different types of approaches follows in Chapter 8.

A "classical" analysis of ellipsis is the one by Dalrymple, Shieber and Pereira [30], henceforth DSP. They relate the source and target VP semantics using higher-order unification. We sketch the analysis with a simple example, the sentence (2.16). The meaning of this sentence is modeled by the formula (2.17) together with the equation (2.18): the common part of source and target sentence must be some property that holds of both the source and target exception(s). Solving the equation yields the solution (2.19). When we apply this property to the target exception "Mary" we obtain the meaning of the target sentence, (2.20).

(2.16) John sleeps, and Mary does too.

(2.17) $sleep(john) \wedge P(mary)$

(2.18) $P(john) = sleep(john)$.

(2.19) $P = \lambda x.sleep(x)$.

(2.20) $sleep(mary)$.

Crouch [28] follows the same idea as DSP, but uses substitution instead of full higher-order unification. In this approach, the semantic representations are phrased in Quasi-Logical Form (QLF) [2], a formalism based on lambda calculus in which quantifier scope is represented by special *scope nodes*; they are uninstantiated until quantifier scope is resolved. Apart from the better computational complexity of this approach in comparison to DSP, Crouch stresses the need for a semantic formalism that is declarative, giving a (partial) description of the intended semantic composition, rather than procedural and dependent on the order in which ambiguities are resolved.

As we have mentioned above, Niehren, Pinkal and Ruhrberg [94] propose an analysis of ellipsis that uses context unification rather than higher-order unification in the general case. Their approach integrates the treatment of ellipsis with an underspecified description of scope ambiguities.

Higher-order unification has also been employed to model other parallelism phenomena: For the interpretation of *focus*, Gardent and Kohlhase use higher-order unification to extract the *focus semantic value* [50]. For example, for the sentence "John only likes MARY" – where focus is indicated using upper-case –, the focus semantic value is the set of properties of the form *liking y*, where y is an individual. This value is then used to describe what the semantics of the "only" is in this sentence. In *corrections* in discourse, strict/sloppy ambiguities are possible, as e.g. in "John$_1$ loves his$_1$ wife." – "No, PETER loves his wife." Gardent, Kohlhase and van Leusen use higher-order unification to model these phenomena in a similar fashion as DSP [51].

Reyle [101] discusses parallelism phenomena within the formalism of UDRT. As we have said above, UDRT attaches labels to the boxes of formulas and referents that are typical for DRT, and between these labels, dominance can be stated. To model pararallelism, these labels are now decorated with indices. Given two pairs of boxes with corresponding indices, then dominance must order both pairs in the same way. Reyle uses this mechanism for two purposes, on the one hand inference in an underspecified framework – from an underspecified premise an underspecified conclusion is drawn, with parallelism linking them –, and on the other hand for handling ambiguities related to plural: occurrences of the same ambiguous plural expression can be indexed to ensure they are disambiguated the same way. Schiehlen [106] takes up this coindexing technique to handle the interaction of scope and ellipsis in the UDRT setting. However, in this approach everything that is *included* in the parallelism has to be specified explicitly, while in DSP and the CLLS analysis all the material in the parallel regions is included by definition.

The approach by Hardt [59] focuses on the similarities of anaphora and ellipsis. Using a DRT setting, this analysis gives the source sentence a referent in the universe that the target sentence can then refer to. In this approach, examples where the source sentence can only be found by inference play an important role, especially when this inference parallels steps that need to be performed for anaphora resolution.

2.5.3 Related Analyses of the Interaction of Ellipsis and Anaphora

Williams [117] models strict/sloppy ambiguities as ambiguities in the *source* rather than in the target sentence. For example, for sentence (2.10), the source sentence of the elliptical (2.11), there would be two representations:

1. Mary$_1$ ($\lambda x.x$ saw her$_1$ mother) and
2. Mary$_1$ ($\lambda x.x$ saw x's mother).

The first representation results in a strict reading of the target sentence, the second in a sloppy reading. Note that there is no separate construct for representing anaphoric binding, rather it is modeled as lambda binding.

DSP also models anaphoric binding by lambda binding. To handle strict/sloppy ambiguities, they distinguish *primary* and *secondary* occurrences of the subtree that has the shape of the exception. Consider sentence (2.21). In solving the equation (2.23) for this ellipsis, primary occurrences – underlined in the example – have to be abstracted, while secondary occurrences may or may not be. The underlined occurrence of "dan" is primary because it constitutes the contrasting element; the other occurrence is secondary because it arises from the representation of the pronoun "his". There are four solutions to the higher-order unification problem in (2.23). Two of them, $\lambda x.likes(\underline{dan}, wife{-}of(dan))$ and $\lambda x.likes(\underline{dan}, wife{-}of(x))$, do not obey the restriction that a primary occurrence must be abstracted, so they are eliminated. In the other two solutions to the unification problem, $\lambda x.likes(x, wife{-}of(dan))$ and $\lambda x.likes(x, wife{-}of(x))$, the primary occurrence of "dan" is abstracted, and indeed these two solutions correspond to the strict and the sloppy reading of sentence (2.21).

(2.21) Dan likes his wife, and George does, too.

(2.22) $likes(\underline{dan}, wife{-}of(dan)) \wedge P(george)$

(2.23) $P(dan) = likes(\underline{dan}, wife{-}of(dan))$

Kehler [70] connects the semantic representations of a pronoun and its antecedent by the *linking relation*, which corresponds to Chomsky's anaphoric binding relation in syntactic representations [19]. Linking relations in the source clause determine linking relation in the target clause, by the operations of *referring* and *copying*. The operation of *referring* is similar to connecting the target anaphor to its own correspondent, while the operation of *copying* is similar to connecting the target anaphor to the correspondent of the source binder. One linking relationship in the source clause gives rise to two possible linking relationships in the target clause. This analysis is basically the same as the conditions on anaphoric binding in CLLS (Def. 2.7), which were proposed by Xu [118].

2.6 Summary

In this chapter we have introduced the Constraint Language for Lambda Structures, CLLS. Lambda structures are finite constructor trees augmented with two node mappings for modeling lambda and anaphoric binding. CLLS is a language of partial descriptions of lambda structures, offering constraints that describe relations between nodes.

CLLS can be seen as a hierarchy of three sublanguages:

- $\mathcal{C}_\mathrm{d}$ constraints, with *labeling* and *dominance* as their most important types of literals. Labeling

  $$X_0{:}f(X_1, \ldots, X_n)$$

  expresses that the node that X_0 denotes bears the label f and has the nodes denoted by $X_1, \ldots, X_n$ (in this order) as children, and dominance

  $$X_0 \triangleleft^* X_1$$

  states that the node for which X_0 stands is an ancestor of the node for X_1. Models for $\mathcal{C}_\mathrm{d}$ constraints are tree structures, i.e. node-labelled trees without the additional mappings.

- $\mathcal{C}_\mathrm{p}$ constraints, which extend $\mathcal{C}_\mathrm{d}$ by *parallelism* literals. A parallelism literal

  $$X_0/X_1, \ldots, X_n {\sim} Y_0/Y_1, \ldots, Y_n$$

  states that the *segment* denoted by $X_0/X_1, \ldots, X_n$ has the same structure as the segment for $Y_0/Y_1, \ldots, Y_n$. A segment is a tree from which some subtrees have been cut out, leaving behind holes.

  Models for $\mathcal{C}_\mathrm{p}$ constraints are tree structures extended by a parallelism relation between pairs of segments. Parallelism between two segments can be characterized by a *correspondence function* which links each node in one segment to the node at the same position in the other segment. Corresponding nodes must bear the same labels and have corresponding children.

- CLLS extends $\mathcal{C}_\mathrm{p}$ by lambda and anaphoric binding literals. A lambda binding literal

  $$\lambda(X){=}Y$$

  states that the node denoted by X is var-labeled and has its lambda binder at the node denoted by Y, and an anaphoric binding literal

  $$\mathsf{ante}(X){=}Y$$

  says that the node denoted by X is anaphorically bound at Y. Models for CLLS are lambda structures. Their parallelism relation must obey a number of restrictions in their interaction with lambda and anaphoric binding: binding within two parallel segments is parallel; if a node is bound outside its segment, its correspondent has the same binder; and hanging binders, i.e. a variable outside being bound inside a segment involved in parallelism, are prohibited.

These three parts of CLLS correspond to three main phenomena that it can model within the application to underspecified semantics: dominance can be used to model scope ambiguities, parallelism can be used to model ellipsis, and anaphoric binding literals can model anaphoric binding.

A sublanguage of CLLS (consisting only of labeling and parallelism literals) is equally expressive as context unification [93, 92], the decidability of which is still an open problem. This is especially interesting as this class of unification problems lies right at the border between string unification, which is decidable, and second-order unification, which is not. Contexts of CU and segments of CLLS are closely related; however, contexts have "a life of their own" as mappings from trees to trees, while segments are always embedded within their surrounding tree, to which they are linked by binding relations.

The language CLLS was introduced in 1998 by Egg, Niehren, Ruhrberg and Xu [42]. A more extensive description is given in a recent overview paper [41]. The language CLLS as we have defined it in this chapter is the one used in the 2000 paper on parallelism [46], except for the possibility of having more than one hole in a segment term; this extension is first present in the first paper on underspecified beta reduction in CLLS [12].

Chapter 3

Solving Dominance Constraints

In this and the following two chapters we develop a procedure for solving CLLS constraints. The procedure divides naturally into three parts, like the language CLLS: In the previous chapter we have seen that CLLS can be regarded as a hierarchy of three languages, dominance constraints, parallelism constraints, and all of CLLS. The three parts of the procedure match this hierarchy. The basis of the procedure is an algorithm for $\mathcal{C}_{\mathrm{d}}$, the class of dominance constraints.

This algorithm for $\mathcal{C}_{\mathrm{d}}$ is the topic of the present chapter. We discuss a *constraint solver* for dominance constraints, a terminating procedure that tests satisfiability. We formulate it as a *saturation-based* algorithm. It accumulates information, never eliminating anything it has found, until a state of saturation is reached. When such a saturated constraint has been found, a model can be read off it directly. For any dominance constraint, only finitely many saturated constraints are computed. Satisfiability of $\mathcal{C}_{\mathrm{d}}$ is an NP-complete problem.

There are algorithms for $\mathcal{C}_{\mathrm{d}}$ that are more sophisticated and more geared towards an implementation [34]. But as we will formulate the parallelism constraint procedure in this simple and abstract paradigm, we already use it for the $\mathcal{C}_{\mathrm{d}}$ solver.

This chapter, like the previous chapter on the language CLLS, does not report new results. Rather, it forms the background for the new procedures for parallelism constraints and for CLLS as a whole, which we discuss in the following chapters. We use the same techniques for the proofs in this chapter as in the two following ones. However here we use them on a simpler problem, such that this chapter can serve as a gentle introduction to the problems that we will be considering later.

3.1 A Solver for Dominance Constraints: $\mathcal{P}_{\mathrm{d}}$

In this section we present a constraint solver for the language $\mathcal{C}_{\mathrm{d}}$ of dominance constraints, i.e. an algorithm that decides the satisfiability of $\mathcal{C}_{\mathrm{d}}$ constraints. Dominance constraints are interpreted over the class of lambda structures, so testing satisfiability means that the algorithm has to decide on the existence of a *model*, a lambda structure plus a valuation. The algorithm does not just give a yes/no answer, it computes result constraints. From each of those a model can be read off.

Let $\ell, \ell_1, \ldots, \ell_5, \ell'_4, \ell'_5$ be literals.

(a) a deterministic rule $\ell_1 \wedge \ell_2 \to \ell$:		
$\{\ldots, \{\underline{\ell_1, \ell_2}, \ell_3\}, \ldots\}$	$\to$	$\{\ldots, \{\ell_1, \ell_2, \ell_3, \ell\}, \ldots\}$
(b) a distribution rule $\ell_1 \wedge \ell_2 \to (\ell_4 \wedge \ell_5) \vee (\ell'_4 \wedge \ell'_5)$:		
$\{\ldots, \{\underline{\ell_1, \ell_2}, \ell_3\}, \ldots\}$	$\to$ $\to$	$\{\ldots,$ $\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5\},$ $\{\ell_1, \ell_2, \ell_3, \ell'_4, \ell'_5\},$ $\ldots\}$

Figure 3.1: Applying saturation rules to a set of clauses

We formulate the algorithm $\mathcal{P}_{\mathrm{d}}$ as a *saturation algorithm*, which consists of a set of *saturation rules*. It operates on a set of *clauses*: A clause is a set of literals. Abusing notation a little, we view a clause also as a constraint and vice versa: We regard a clause, and also a constraint, as both the set and the conjunction of the literals occurring in it. Hence we can say that a literal ℓ is *in* a constraint φ iff $\ell \in \varphi$, and a constraint φ' is *in* φ iff $\varphi' \subseteq \varphi$.

A saturation algorithm adds more literals to the clauses and more clauses to the set according to the saturation rules, until the set is saturated, i.e. nothing new can be added anymore. The saturation rules that we use have the form

$$\varphi_0 \to \bigvee_{i=1}^{n} \varphi_i$$

for clauses $\varphi_0, \ldots, \varphi_n$ and $n \geq 1$. A rule is deterministic if $n = 1$. Application of a deterministic rule is illustrated in Fig. 3.1 (a): We choose a clause that contains the rule's left-hand side, and we add the right-hand side to the clause. A rule with $n > 1$ is indeterministic, also called a *distribution rule*. Consider Fig. 3.1 (b): Again, a clause containing the left-hand side is chosen. This clause is replaced by two new clauses, each consisting of the old clause plus either $\ell_4 \wedge \ell_5$ or $\ell'_4 \wedge \ell'_5$.

We next define a *saturation step*, a single rule application. The applicability of a rule ρ is dependent on an *application condition* appc_ρ.

Definition 3.1 (Saturation step, application condition). *Let $\mathcal{S}$ be a set of saturation rules. A* saturation step $\to_{\mathcal{S}}$ *consists of one application of a rule $\rho \in \mathcal{S}$. Let $\rho = \varphi_0 \to \bigvee_{i=1}^{n} \varphi_i$ for clauses $\varphi_0, \ldots, \varphi_n$. Then*

$$\frac{\varphi_0 \subseteq \varphi}{\varphi \to_{\mathcal{S}} \varphi \;\wedge\; \varphi_i} \; \textit{for } i \in \{1, \ldots, n\} \textit{ if } \mathrm{appc}_\rho(\varphi)$$

where the application condition is

$$\mathrm{appc}_\rho(\varphi) =_{\mathrm{def}} \textit{ for all } 1 \leq i \leq n : \varphi_i \not\subseteq \varphi$$

What the application condition appc_ρ says is that a saturation rule $\rho = \varphi_0 \rightarrow \bigvee_{i=1}^{n} \varphi_i$ is applicable to a clause φ iff φ contains the left-hand side φ_0 but *none* of the right-hand side choices $\varphi_1, \ldots, \varphi_n$. If φ_i is present in the clause for some $1 \leq i \leq n$, then the choice has already been made and the rule need not be applied anymore. (Note that the application condition governs the applicability of a rule to each individual clause in a clause set; a rule that is not applicable to one clause in the set may still be applicable to another.)

A saturation algorithm *terminates* when no rule is applicable to any clause in the set anymore. So the application condition that we have just introduced will ensure the termination of our dominance constraint algorithm: We will formulate the algorithm such that it never adds fresh variables to the clause set it operates on, and it can only add finitely many different literals for each variable in its clause set.

In a saturation algorithm, the choice of the next rule to apply is *don't care* indeterministic: It does not matter which rule is chosen first. On the other hand, distribution rules are *don't know* indeterministic – each choice in the right-hand side of the rule has to be explored.

Definition 3.2 (Clash-free, saturated, failed). *Let $\mathcal{S}$ be a set of saturation rules. A clause is* clash-free *iff it does not contain* `false`*, and* $\mathcal{S}$-saturated *iff it is irreducible with respect to $\rightarrow_{\mathcal{S}}$ and clash-free. If a clause contains* `false`*, it is also called* failed.

We also call a saturated constraint a *saturation* for short. These saturations are the result constraints that our dominance constraint solver computes.

3.1.1 The Rules in Detail

Remember that the class $\mathcal{C}_\text{d}$ of dominance constraints has the following abstract syntax:

$$\begin{array}{lll} \varphi, \varsigma & ::= & X \triangleleft^* Y \mid X{:}f(X_1, \ldots, X_n) \mid X \perp Y \mid X \neq Y \qquad (\mathsf{ar}(f) = n) \\ & \mid & \texttt{false} \mid \varphi \ \wedge \ \varsigma \end{array}$$

Additionally, we use the abbreviations

$$X{=}Y \text{ for } X \triangleleft^* Y \wedge Y \triangleleft^* X \text{ and } X \triangleleft^+ Y \text{ for } X \triangleleft^* Y \wedge X \neq Y.$$

Inequality and disjointness literals are viewed as symmetric.

Figure 3.2 shows the solver $\mathcal{P}_\text{d}$ for dominance constraints. The first two rules, (D.clash.ineq) and (D.clash.disj), detect unsatisfiable constraints and extend them by `false`. We call such rules *clash rules.* (D.clash.ineq), which has the form $X{=}Y \ \wedge \ X{\neq}Y \rightarrow$ `false`, states that two variables cannot denote the same tree node and different tree nodes at the same time. The rule (D.clash.ineq), which is $X \perp X \rightarrow$ `false`, says that no tree node can be in a disjoint position from itself. The remaining rules will extend all unsatisfiable constraints to a point where one of these clash rules applies.

Rule	
(D.clash.ineq)	$X{=}Y \ \wedge\ X{\neq}Y \to \texttt{false}$
(D.clash.disj)	$X{\perp}X \to \texttt{false}$
(D.dom.refl)	$\varphi \to X{\triangleleft^*}X$ where $X \in \mathcal{V}ar(\varphi)$
(D.dom.trans)	$X{\triangleleft^*}Y \ \wedge\ Y{\triangleleft^*}Z \to X{\triangleleft^*}Z$
(D.lab.decom)	$X{:}f(X_1,\ldots,X_n) \ \wedge\ Y{:}f(Y_1,\ldots,Y_n) \ \wedge\ X{=}Y \to \bigwedge_{i=1}^n X_i{=}Y_i$
(D.lab.ineq)	$X{:}f(\ldots) \ \wedge\ Y{:}g(\ldots) \to X{\neq}Y$ where $f \neq g$
(D.lab.dom)	$X{:}f(\ldots,Y,\ldots) \to X{\triangleleft^+}Y$
(D.lab.disj)	$X{:}f(\ldots X_i,\ldots,X_j,\ldots) \to X_i{\perp}X_j$ where $1 \leq i < j \leq n$
(D.disj)	$X{\perp}Y \ \wedge\ X{\triangleleft^*}X' \ \wedge\ Y{\triangleleft^*}Y' \to Y'{\perp}X'$
(D.distr.notDisj)	$X{\triangleleft^*}Z \ \wedge\ Y{\triangleleft^*}Z \to X{\triangleleft^*}Y \ \vee\ Y{\triangleleft^*}X$
(D.distr.child)	$X{\triangleleft^*}Y \ \wedge\ X{:}f(X_1,\ldots,X_n) \to Y{=}X \ \vee\ \bigvee_{i=1}^n X_i{\triangleleft^*}Y$

Figure 3.2: Solving $\mathcal{C}_{\mathrm{d}}$ constraints: algorithm $\mathcal{P}_{\mathrm{d}}$

The rules in the second block are deterministic saturation rules. (D.dom.refl) and (D.dom.trans), which are $\varphi \to X{\triangleleft^*}X$ for $X \in \mathcal{V}ar(\varphi)$ and $X{\triangleleft^*}Y \ \wedge\ Y{\triangleleft^*}Z \to X{\triangleleft^*}Z$, state that dominance is a reflexive and transitive relation. The rules (D.lab...) are concerned with the labeling relation. (D.lab.decom) is a decomposition rule which states $X{:}f(X_1,\ldots,X_n) \wedge Y{:}f(Y_1,\ldots,Y_n) \wedge X{=}Y \to \bigwedge_{i=1}^n X_i{=}Y_i$, propagating equality from two equal variables to their children. (D.lab.ineq), by stating $X{:}f(\ldots) \ \wedge\ Y{:}g(\ldots) \to X{\neq}Y$ for $f \neq g$, expresses the fact that two differently labelled variables can never denote the same node. The rule (D.lab.dom), of the form $X{:}f(\ldots,Y,\ldots) \to X{\triangleleft^+}Y$, declares that a parent dominates its children. The rule (D.lab.disj), which states $X{:}f(\ldots X_i,\ldots,X_j,\ldots) \to X_i{\perp}X_j$ for $1 \leq i < j \leq n$, says that two different children of the same node must lie in disjoint positions. The rule (D.disj), of the form $X{\perp}Y \wedge X{\triangleleft^*}X' \wedge Y{\triangleleft^*}Y' \to Y'{\perp}X'$, propagates disjointness from two variables to their descendants.

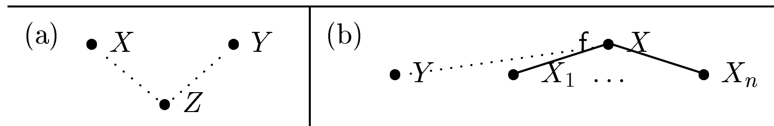


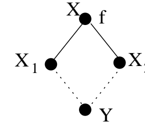
Figure 3.3: Situations in which (a) (D.distr.notDisj) and (b) (D.distr.child) apply

(D.distr.notDisj) and (D.distr.child) describe the only two situations in which $\mathcal{P}_{\mathrm{d}}$ needs to distribute. They are illustrated in Fig. 3.3. The rule (D.distr.notDisj) has the form $X{\triangleleft^*}Z \wedge Y{\triangleleft^*}Z \to X{\triangleleft^*}Y \ \vee\ Y{\triangleleft^*}X$. It states that if X and Y have a common descendant Z, their denotations cannot be in disjoint positions because trees do not branch upwards. So one of the two variables must dominate the other. The rule (D.distr.child) has the

form $X \triangleleft^* Y \;\wedge\; X{:}f(X_1, \ldots, X_n) \rightarrow Y{=}X \;\vee\; \bigvee_{i=1}^{n} X_i \triangleleft^* Y$. It applies to a variable X that both dominates another variable Y and is labeled. Then Y must be either equal to X, or it lies below one of X's children.

3.1.2 Examples

As a first example of how the saturation rules work, let us reconsider the unsatisfiable constraint $X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$ of Fig. 2.7, repeated to the right. By (D.lab.disj), we infer $X_1 \perp X_2$, to which (D.disj) adds $Y \perp Y$. But then the clash-rule (D.clash.disj) applies, signifying that the constraint is unsatisfiable.



Next, consider the constraint $X{:}f(X)$, which is also unsatisfiable. By (D.dom.refl) we get $X \triangleleft^* X$, which is the same as $X{=}X$, an abbreviation for $X \triangleleft^* X \wedge X \triangleleft^* X$. Then (D.lab.dom) gets us $X \triangleleft^+ X$, which is the same as $X \triangleleft^* X \wedge X{\neq}X$. But now (D.clash.ineq) applies and adds `false`, since we have both $X{=}X$ and $X{\neq}X$.

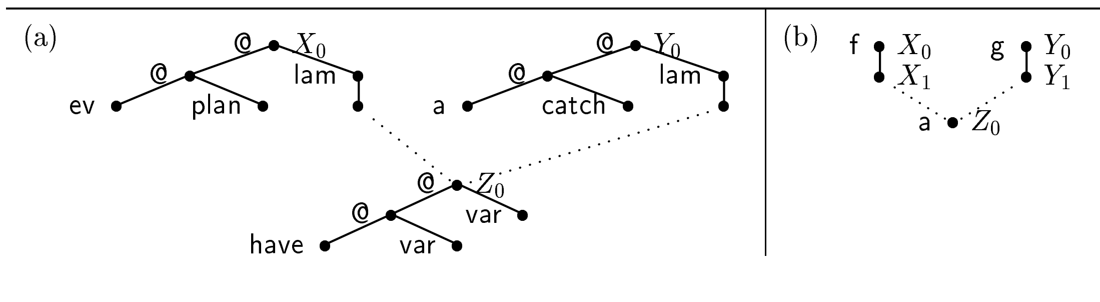


Figure 3.4: (a) Constraint for *Every plan has a catch*, and (b) a smaller, abstract version

The distribution rule (D.distr.notDisj) is central to solving constraints that model scope ambiguities, like the one in Fig. 3.4 (a) (this is Fig. 1.6 without the binding edges). To concentrate on the important aspects, we demonstrate our algorithm $\mathcal{P}_{\mathrm{d}}$ on a smaller, abstract version of Fig. 3.4 (a), which is shown in Fig. 3.4 (b).

By (D.lab.dom), $X_0 \triangleleft^+ X_1$ and $Y_0 \triangleleft^+ Y_1$. Hence by (D.dom.trans) we get $X_0 \triangleleft^* Z_0$ as well as $Y_0 \triangleleft^* Z_0$. So we must have either $X_0 \triangleleft^* Y_0$ or $Y_0 \triangleleft^* X_0$ by (D.distr.notDisj). We pursue the first alternative. Now we are in the situation sketched in Fig. 3.3 (b): We have $X_0 \triangleleft^* Y_0$, and X_0 is labeled. So the rule (D.distr.child) offers two possibilities: either $X_0{=}Y_0$ or $X_1 \triangleleft^* Y_0$. Again we pursue the first alternative (intuitively, we are now trying to "overlap" the f-fragment with the g-fragment). However, (D.lab.ineq) gives us $X_0{\neq}Y_0$, so now we have $X_0{=}Y_0$ as well as $X_0{\neq}Y_0$, which fails by (D.clash.ineq). So let us consider the second alternative of (D.distr.child) above, which was $X_1 \triangleleft^* Y_0$. We now have $X_0{:}f(X_1), X_1 \triangleleft^* Y_0, Y_0{:}g(Y_1)$, and $Y_1 \triangleleft^* Z_0$. This constraint can be saturated without any further distribution. This saturation is the one shown in Fig. 3.5 (a).

Now suppose that, instead of pursuing the choice $X_0 \triangleleft^* Y_0$ of (D.distr.notDisj) above, we follow the other alternative $Y_0 \triangleleft^* X_0$. Then the saturation proceeds just as in the case of

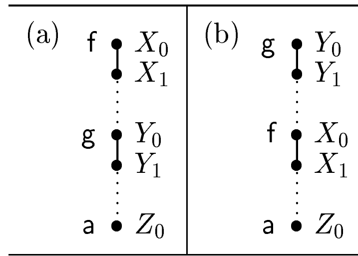


Figure 3.5: The two saturations computed for Fig. 3.4 (b)

$X_0 \lhd^* Y_0$. Again we get a single saturation, which is shown in Fig. 3.5 (b).

The bigger constraint in Fig. 3.4 (a) is saturated in the same way as the small one we have just considered. Again, we get two saturations, sketched in Fig. 3.6.

3.2 Some Properties of the Algorithm: Soundness, Termination, Shape of Saturations

In this and the following sections we examine properties of the algorithm $\mathcal{P}_\mathrm{d}$. All results are collected in a theorem in Sec. 3.5.

3.2.1 Soundness

The constraint solver $\mathcal{P}_\mathrm{d}$ is *sound* in the sense that all its rules are equivalence transformations.

Definition 3.3 (Soundness). *We call a saturation rule* $\varphi \to \Phi$ sound *for lambda structures iff* $\varphi \models\!\!\!\!= \Phi$.

But we are working in a saturation paradigm: We never eliminate any literals. So a saturation rule $\varphi \to \Phi$ is already sound if $\varphi \models \Phi$.

It is easy to see that in each rule of $\mathcal{P}_\mathrm{d}$, the left-hand side entails the right-hand side.

Lemma 3.4 (Soundness). *The* $\mathcal{C}_\mathrm{d}$*-solver* $\mathcal{P}_\mathrm{d}$ *is sound for lambda structures.*

3.2.2 Termination

Lemma 3.5 (Termination). $\mathcal{P}_\mathrm{d}$ *is terminating.*

Proof. The algorithm $\mathcal{P}_\mathrm{d}$ never adds fresh variables to the clause set that it is working on. For each variable, there are only finitely many different literals that can be added to each clause. Finally, the application condition prohibits rules from adding the same constraints to the same clause over and over again: For each rule $\rho = \varphi \to \bigvee_{i=1}^{n} \varphi_i$ of

$\mathcal{P}_\mathrm{d}$, the application condition appc_ρ states that ρ can be applied to a clause only if φ is in it, but none of the φ_i is already contained. □

Satisfiability of dominance constraints is an NP-complete problem, as Koller, Niehren and Treinen [78] have shown. They encode SAT by forcing fragments of a constraint graph to overlap, but giving them two possible ways of overlapping, in this way encoding true and false.

3.2.3 Saturated Constraints

For each input dominance constraint, the algorithm computes a set of saturations, constraints to which no rule of $\mathcal{P}_\mathrm{d}$ is applicable anymore.

Lemma 3.6. *For any dominance constraint, $\mathcal{P}_\mathrm{d}$ computes a finite set of saturations (which may be empty).*

Proof. This is proven by the same arguments as Lemma 3.5 above: The algorithm $\mathcal{P}_\mathrm{d}$ never adds fresh variables, and there are only finitely many different literals that can be added to each clause and hence only finitely many rule applications are possible. □

If we look at other constraint solvers that work by transforming or augmenting a constraint, the results of their computation are often called *solved forms*. Typically, solved forms are an independently defined subclass of constraints that are simpler than the original ones. The saturations that the solver $\mathcal{P}_\mathrm{d}$ computes are basically solved forms too, except that saturations are defined not independently but in relation to $\mathcal{P}_\mathrm{d}$, and that technically they are not simpler than the input constraint since they subsume it.

Saturated constraints are like solved forms in that they are constraints from which a model can be directly read off – we will show this in the following section. So in a way they *are* more simple than dominance constraints in general; more precisely, their *constraint graphs* have a very simple structure, which we are now going to characterize informally.

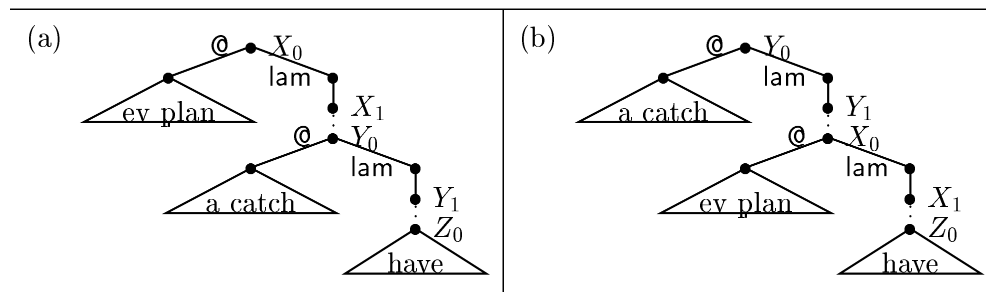


Figure 3.6: Sketch of the two saturated constraints computed for Fig. 3.4 (a)

Remember that in the previous chapter, we have said that constraint graphs do not represent disjointness and inequality literals, and that they shows dominance literals only when they connect two different *fragments* (tree-shaped pieces connected by solid lines) of the graph.

As an example of what constraint graphs of saturated constraints look like, consider the constraint in Fig. 3.4. For this constraint the algorithm $\mathcal{P}_\mathrm{d}$ computes two different saturations, the constraint graphs of which are sketched in Fig. 3.6. These constraint graphs are tree-shaped, except that some edges are dominance edges.

More generally, the constraint graph of a saturated constraint is a forest with two different kinds of tree nodes, labeled and unlabeled ones, and two different kinds of edges, dominance and labeling edges. For each node we can say that

- either it is labeled, all its outgoing edges are labeling edges, and its children are ordered;
- or it is unlabeled, all its outgoing edges are dominance edges, and its children are unordered.

So a $\mathcal{P}_\mathrm{d}$-saturated constraint is similar to a forest of quasi-trees [115, 102] or D-trees [99]: Duchier and Gardent [33] use a formulation of D-trees that allows for nodes with more than one outgoing dominance edge. With this definition, each $\mathcal{P}_\mathrm{d}$-saturated $\mathcal{C}_\mathrm{d}$-constraint can be regarded as a forest of D-trees.

3.3 Satisfiability of Saturated Constraints

In this section we show that $\mathcal{P}_\mathrm{d}$-saturations are satisfiable, more precisely: that from each saturation a model can be read off. The proof that we use has the same shape as those in the papers by Koller, Niehren and Treinen [78] and by Duchier and Niehren [34]. We proceed in two steps. First we consider only a subclass of constraints, which we call *simple* constraints. Then we lift the result to arbitrary $\mathcal{P}_\mathrm{d}$-saturated constraints.

3.3.1 Simple Constraints

We first consider *simple* constraints, the constraint graphs of which are already tree-shaped. We show that from each *simple* $\mathcal{P}_\mathrm{d}$-saturated constraint, a model can be read off.

Definition 3.7 (Labeled, simple). *Let φ be a $\mathcal{C}_\mathrm{d}$ constraint. A variable $X \in \mathcal{V}ar(\varphi)$ is called* labeled *in φ iff $\exists X' \in \mathcal{V}ar(\varphi)$ such that $X{=}X'$ and $X'{:}f(X_1, \ldots, X_n)$ are in φ for some term $f(X_1, \ldots, X_n)$. We call φ* simple *if all its variables are labeled and there exists some variable $Z \in \mathcal{V}ar(\varphi)$ such that $Z \triangleleft^* X$ is in φ for all $X \in \mathcal{V}ar(\varphi)$.*

So in a simple constraint every variable is labeled, and there is a *root variable* Z dominating all others.

Lemma 3.8 (Satisfiability of simple saturations). *A simple $\mathcal{P}_{\mathrm{d}}$-saturated $\mathcal{C}_{\mathrm{d}}$-constraint is satisfiable.*

Proof. Let φ be a simple $\mathcal{P}_{\mathrm{d}}$-saturated $\mathcal{C}_{\mathrm{d}}$-constraint. We construct a tree structure θ that is a model for φ. We proceed by induction on the number of literals in φ. Let Z be a root variable in φ. Since all variables in φ are labeled, there is a variable Z' and a term $f(Z_1, \ldots, Z_n)$ such that $Z{=}Z'$ and $Z'{:}f(Z_1, \ldots, Z_n)$ are in φ. Let

$$\begin{array}{lll} V & =_{\mathrm{def}} & \{X \in \mathcal{V}ar(\varphi) \mid Z{=}X \text{ in } \varphi\} \text{ and} \\ V_i & =_{\mathrm{def}} & \{X \in \mathcal{V}ar(\varphi) \mid Z_i{\lhd^*}X \text{ in } \varphi\}. \end{array}$$

for all $1 \le i \le n$. We show that $V, V_1, \ldots, V_n$ form a partition of $\mathcal{V}ar(\varphi)$:

- First, $\mathcal{V}ar(\varphi) = V \cup V_1 \cup \ldots \cup V_n$: Let $X \in \mathcal{V}ar(\varphi)$ such that $Z_i{\lhd^*}X \notin \varphi$ for all $1 \le i \le n$. As Z is a root variable, $Z{\lhd^*}X \in \varphi$, so by saturation with (D.distr.child), φ must contain $Z{=}X$.
- Second, $V, V_1, \ldots, V_n$ are disjoint sets: Suppose there is some variable $X \in \mathcal{V}ar(\varphi)$ with $X \in V$ as well as $X \in V_i$ for some i. Then φ contains $Z_i{\lhd^*}X$ as well as $X{\lhd^*}Z$, hence it contains $Z_i{\lhd^*}Z$ by closure under (D.dom.trans) as well as $Z{\lhd^*}Z_i, Z{\neq}Z_i$ by (D.lab.dom) – a contradiction since then φ would also contain `false` by (D.clash.ineq). Now suppose there are $1 \le i < j \le n$ and a variable $X \in \mathcal{V}ar(\varphi)$ such that $X \in V_i$ as well as $X \in V_j$. By (D.lab.disj) φ contains $Z_i{\perp}Z_j$, which with $Z_i{\lhd^*}X$ and $Z_j{\lhd^*}X$ gives us $X{\perp}X$ by (D.disj) – again a contradiction since then φ would contain `false` by (D.clash.disj).

For a set $W \subseteq \mathcal{V}ar(\varphi)$ we define $\varphi|_W$ as the conjunction of all literals $\varsigma \in \varphi$ with $\mathcal{V}ar(\varsigma) \subseteq W$. We show that

$$\varphi \models\!\!\!\!\!\!=\, \varphi' \quad \text{holds where} \quad \varphi' =_{\mathrm{def}} \varphi|_V \wedge Z{:}f(Z_1, \ldots, Z_n) \wedge \bigwedge_{i=1}^{n} \varphi|_{V_i}.$$

It obviously holds that $\varphi \models \varphi'$: The only literal that may be in $\varphi' - \varphi$ is $Z{:}f(Z_1, \ldots, Z_n)$, and that is entailed by φ because $Z'{:}f(Z_1, \ldots, Z_n), Z{=}Z'$ are in φ. Next we show that $\varphi' \models \varphi$ holds because φ is a $\mathcal{P}_{\mathrm{d}}$-saturated constraint:

- Suppose $X{:}g(X_1, \ldots, X_m)$ is in φ for some variable X and term $g(X_1, \ldots, X_m)$. If $Z_i{\lhd^*}X$ is in φ for some $1 \le i \le n$, then $X{:}g(X_1, \ldots, X_m)$ is in $\varphi|_{V_i}$ since φ is saturated under (D.lab.dom) and (D.dom.trans). Otherwise, $Z{=}X$ is in φ, and thus $Z{=}X$ is in $\varphi|_V$. In this case, $f = g$ and $n = m$ by saturation with (D.lab.ineq) and (D.clash.ineq) coupled with the clash-freeness of φ. As φ is saturated under (D.lab.decom), it must contain $Z_i{=}X_i$ for $1 \le i \le n$, hence $Z_i{=}X_i$ must be in $\varphi|_{V_i}$. So, φ' contains $Z{=}X \wedge Z{:}f(Z_1, \ldots, Z_n) \wedge \bigwedge_{i=1}^{n} Z_i{=}X_i$, which entails $X{:}g(X_1, \ldots, X_m)$ as required.

- Now suppose $XRY \in \varphi$ for some variables X, Y and $R \in \{\triangleleft^*, \neq, \perp\}$. There are four possible cases:
  - If $X \in V_i$, $Y \in V_j$ with $1 \leq i \neq j \leq n$, then R cannot be $\triangleleft^*$: In this case φ would contain $X \triangleleft^* Y$, $Y \triangleleft^* Y$ by (D.dom.refl) and $X \perp Y$ by (D.disj) (because φ contains $Z_i \perp Z_j$), which yields $Y \perp Y$ by (D.disj), making (D.clash.disj) applicable, but φ is clash-free. Concerning the other two possible values for R, φ' entails $Z_i \perp Z_j$ and thus $X \perp Y$ as well as $X \neq Y$.
  - The cases where X and Y both belong to V or to the same V_i are obvious.
  - If $X \in V$ and $Y \in V_i$ for some i, then R cannot be $\perp$ by the same argument that we used in the first case above. Concerning the other two possible values of R, φ' entails $Z \triangleleft^+ Z_i$ and thus $X \triangleleft^* Y$ and $X \neq Y$.
  - The case of $X \in V_i$ and $Y \in V$ is symmetric to the previous one, except that now R cannot be $\triangleleft^*$: φ contains $Z_i \triangleleft^* X$ by definition, which with $X \triangleleft^* Y$ and $Y = Z$ would mean that φ contains $Z_i \triangleleft^* Z$. But φ contains $Z \triangleleft^* Z_i, Z \neq Z_i$ by (D.lab.dom), a contradiction since φ is clash-free and closed under (D.clash.ineq).

Next note that all $\varphi|_{V_i}$ are simple $\mathcal{P}_\mathrm{d}$-saturated constraints. By the inductive hypothesis there exist models $(\theta_i, \sigma_i) \models \varphi|_{V_i}$ for all $1 \leq i \leq n$. Now, since $V, V_1, \ldots, V_n$ is a partition of $\mathcal{V}ar(\varphi)$, we can combine the models for the smaller constraints into a model of φ: $(f(\theta_1, \ldots, \theta_n), \sigma)$ is a model of φ if $\sigma|_{V_i} = \sigma_i$ for $1 \leq i \leq n$, and $\sigma(X) = \sigma(Z)$ is the root node of $f(\theta_1, \ldots, \theta_n)$ for all $X \in V$. □

3.3.2 Non-simple Constrains

Now we show that we can *extend* each non-simple $\mathcal{P}_\mathrm{d}$-saturated constraint φ to a constraint $\varphi \wedge \varphi'$ that is simple and still $\mathcal{P}_\mathrm{d}$-saturated. We proceed by successively labeling unlabeled variables. Suppose, for instance, we want to label the unlabeled variable X in Fig. 3.7 (a). Then we would like to make all variables minimally dominated by X into X's children. We formalize this as follows:

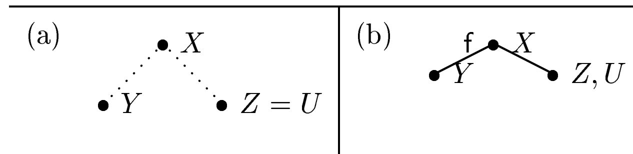


Figure 3.7: Extension by labeling

Definition 3.9 (Connectedness set). *Given a $\mathcal{C}_\mathrm{d}$ constraint φ, we define a partial order $\prec_\varphi$ on its variables by: $X \prec_\varphi Y$ holds iff $X \triangleleft^* Y \in \varphi$ but not $Y \triangleleft^* X \in \varphi$.*

Let $X \in \mathcal{V}ar(\varphi)$ an unlabeled variable. Then we define the set $con_\varphi(X)$ of variables connected to X in φ *as*

$$con_\varphi(X) = \{Y \in \mathcal{V}ar(\varphi) \mid Y \text{ minimal with } X \prec_\varphi Y\}$$

In the constraint in Fig. 3.7 (a), $\text{con}_\varphi(X) = \{Y, Z, U\}$. However, when picking variables to serve as children of X, we choose only one of Z, U as we have $Z{=}U$:

Definition 3.10 (Disjointness set). *Let φ be a $\mathcal{C}_\text{d}$ constraint and $V \subseteq \mathcal{V}ar(\varphi)$. We call V a* φ-disjointness set *if for any two distinct variables $Y_1, Y_2 \in V$, $Y_1 \triangleleft^* Y_2 \notin \varphi$.*

The idea is that all variables in a φ-disjointness set can safely be placed at disjoint positions in at least one of the trees that satisfy φ. So concerning our example in Fig. 3.7 (a), we label X by some function symbol of arity 2, extending the constraint, for instance, by $X{:}f(Y, Z)$. The result is shown in Fig. 3.7 (b). However, we have to make sure that we preserve saturatedness during extension. For example, when adding $X{:}f(Y, Z)$ to the constraint in Fig. 3.7 (a), we also add $Y \perp Z$ such as not to make (D.lab.disj) applicable.

The following technical lemma by Duchier and Niehren [34] will be useful: In a $\mathcal{P}_\text{d}$-saturated constraint φ, each variable in a connectedness set $\text{con}_\varphi(X)$ is equal to one of the variables in a maximal disjointness set within $\text{con}_\varphi(X)$.

Lemma 3.11. *Let φ be $\mathcal{P}_\text{d}$-saturated and $X \in \mathcal{V}ar(\varphi)$. If V is a maximal φ-disjointness set within $con_\varphi(X)$, then for all $Y \in con_\varphi(X)$ there exists some $Z \in V$ such that $Y{=}Z$ is in φ.*

Now we proceed to the main lemma of this subsection: An unlabeled variable in a $\mathcal{P}_\text{d}$-saturation can be labeled while keeping up saturatedness.

Lemma 3.12 (Extension by labeling). *Every $\mathcal{P}_\text{d}$-saturated $\mathcal{C}_\text{d}$-constraint with an unlabeled variable U_0 can be extended to a $\mathcal{P}_\text{d}$-saturated constraint in which U_0 is labeled.*

Proof. Let φ be a $\mathcal{P}_\text{d}$-saturated $\mathcal{C}_\text{d}$-constraint. Let $U_0 \in \mathcal{V}ar(\varphi)$ be unlabeled, and let $\{U_1, \ldots, U_m\}$ be a maximal φ-disjointness set in $\text{con}_\varphi(U_0)$. Let us assume for the moment that Σ contains a function symbol f of arity m. Then we define the following extension $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ of $\varphi \wedge U_0{:}f(U_1, \ldots, U_m)$:

$$\begin{aligned}\mathsf{ext}_{U_0,\ldots,U_m}(\varphi) =_\text{def} \varphi \;\wedge\; & U_0{:}f(U_1, \ldots, U_m) \;\wedge\; \textstyle\bigwedge_{i=1}^m U_0 \neq U_i \;\wedge \\ & \bigwedge_{\substack{U_i \triangleleft^* Z, U_j \triangleleft^* W \in \varphi, \\ 1 \leq i < j \leq n}} Z \perp W \;\wedge \\ & \bigwedge_{\substack{Z:g(\ldots) \in \varphi, \\ g \neq f}} Z \neq U_0\end{aligned}$$

For better readability, we abbreviate $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ by $\mathsf{ext}(\varphi)$. We consider each rule of $\mathcal{P}_\text{d}$ in turn and show that it is not applicable to $\mathsf{ext}(\varphi)$.

(D.clash.ineq): This rule has the form $X{=}Y \wedge X{\neq}Y \rightarrow \texttt{false}$. $\mathsf{ext}(\varphi)$ contains no new dominance literals. Suppose a new inequality literal $U_0 {\neq} U_i$ makes (D.Clash.Ineq) applicable. Then φ must contain $U_0{=}U_i$, which is impossible since $U_i \in \text{con}_\varphi(U_0)$. If a new inequality literal $Z {\neq} U_0$ makes the clash rule applicable, then $Z{:}g(\ldots)$ and $U_0{=}Z$ must be in φ, which is impossible since U_0 is unlabeled in φ.

(D.clash.disj): This rule has the form $X \perp X \to \texttt{false}$. The only new disjointness literals in $\mathsf{ext}(\varphi)$ have the form $Z \perp W$ for $U_i \triangleleft^* Z, U_j \triangleleft^* W$ in φ with $i \neq j$. Assume $Z{=}W$ is in φ. Then by (D.dom.trans) and (D.distr.notDisj), either $U_i \triangleleft^* U_j$ or $U_j \triangleleft^* U_i$ must be in φ, which is impossible since $\{U_i, U_j\}$ is a disjointness set.

(D.dom.refl), (D.dom.trans), (D.distr.notDisj): No new variables or dominance literals have been added.

(D.lab.decom): This rule has the form $X{:}f(X_1, \ldots, X_n) \wedge Y{:}f(Y_1, \ldots, Y_n) \wedge X{=}Y \to \bigwedge_{i=1}^n X_i{=}Y_i$. For this rule to be applicable to U_0 and some literal $Z{:}g(Z_1, \ldots, Z_n) \in \varphi$, $Z{=}U_0$ must be in φ already. But U_0 is unlabeled in φ.

(D.lab.ineq): This rule has the form $X{:}f(\ldots) \wedge Y{:}g(\ldots) \to X{\neq}Y$ for $f \neq g$. The only new labeling literal is $U_0{:}f(U_1, \ldots, U_m)$. $Z{\neq}U_0$ is in $\mathsf{ext}(\varphi)$ for all Z labeled anything but f.

(D.lab.dom): This rule has the form $X{:}f(\ldots, Y, \ldots) \to X \triangleleft^+ Y$. $U_0{:}f(U_1, \ldots, U_m)$ is the only labeling literal in $\mathsf{ext}(\varphi) - \varphi$. We have $U_0 \triangleleft^* U_i \in \varphi$ for all $1 \leq i \leq m$ because $\{U_1, \ldots, U_m\} \subseteq \mathrm{con}_\varphi(U_0)$. $U_0{\neq}U_i$ is in $\mathsf{ext}(\varphi)$ by definition for all $1 \leq i \leq m$.

(D.lab.disj): This rule has the form $X{:}f(\ldots X_i, \ldots, X_j, \ldots) \to X_i \perp X_j$ for $1 \leq i < j \leq n$. The only new labeling literal is $U_0{:}f(U_1, \ldots, U_m)$. By saturation under (D.dom.refl), $U_i \triangleleft^* U_i \in \varphi$ for all $1 \leq i \leq m$, so $U_i \perp U_j$ is in $\mathsf{ext}(\varphi)$ for all $1 \leq i < j \leq m$.

(D.disj): This rule has the form $X \perp Y \wedge X \triangleleft^* X' \wedge Y \triangleleft^* Y' \to Y' \perp X'$. The only disjointness constraints new in $\mathsf{ext}(\varphi)$ have the form $Z \perp W$, where $U_i \triangleleft^* Z, U_j \triangleleft^* W \in \varphi$ for some $j \neq i$. If $Z \triangleleft^* Z'$ and $W \triangleleft^* W'$ are in φ, then by saturation under (D.dom.trans) $U_i \triangleleft^* Z', U_j \triangleleft^* W' \in \varphi$, so $Z' \perp W'$ is in $\mathsf{ext}(\varphi)$.

(D.distr.child): This rule has the form $X \triangleleft^* Y \wedge X{:}f(X_1, \ldots, X_n) \to Y{=}X \vee \bigvee_{i=1}^n X_i \triangleleft^* Y$. Suppose $U_0 \triangleleft^* Z \in \varphi$, but neither $Z \triangleleft^* U_0$ nor $U_i \triangleleft^* Z$ is in φ for any $i \in \{1, \ldots, m\}$. Then $U_0 \prec_\varphi Z$. If $Z \in \mathrm{con}_\varphi(U_0)$, we have the following situation: The disjointness set $\{U_1, \ldots, U_m\}$ is maximal within $\mathrm{con}_\varphi(U_0)$, so $Z{=}U_i$ for some $i \in \{1, \ldots, m\}$ by lemma 3.11, a contradiction. So suppose Z is not minimal, i.e. there exists some $Y \in \mathrm{con}_\varphi(U_0)$ such that $Y \triangleleft^* Z \in \varphi$. But then again, $U_i{=}Y$ for some $i \in \{1, \ldots, m\}$, so $U_i \triangleleft^* Z$.

We now turn to the case that the signature does not contain a function symbol for the arity we need. We can get around this problem by encoding a function symbol of arity m with a nullary symbol and one symbol of arity ≥ 2, the existence of which we have assumed. This encoding may introduce new variables, but only finitely many. For a detailed description of this construction see Koller [75], lemma 4.11. □

By adding a finite number of literals, we can label one unlabeled variable in the constraint while keeping the constraint $\mathcal{P}_\mathrm{d}$-saturated. If we repeat this process a finite number of

times, we have extended the non-simple $\mathcal{P}_\mathrm{d}$-saturated constraint to a simple one, from which we can then read off a model right away.

Proposition 3.13. *Every $\mathcal{P}_\mathrm{d}$-saturated $\mathcal{C}_\mathrm{d}$-constraint can be extended to a simple $\mathcal{P}_\mathrm{d}$-saturated constraint.*

Proof. Let φ be $\mathcal{P}_\mathrm{d}$-saturated. Without loss of generality we can assume that φ has a root variable. (Otherwise we choose a fresh variable X and consider $\varphi \wedge \bigwedge\{X \triangleleft^* Y \mid Y \in \mathcal{V}ar(\varphi)\}$ instead of φ.) By lemma 3.12, we can successively label all variables in φ. □

Lemma 3.8 and Prop. 3.13 together yield the satisfiability of $\mathcal{P}_\mathrm{d}$-saturated constraints.

Lemma 3.14 (Satisfiability of saturations). *A $\mathcal{P}_\mathrm{d}$-saturated $\mathcal{C}_\mathrm{d}$-constraint is satisfiable.*

3.4 Completeness

In this section we show that the algorithm $\mathcal{P}_\mathrm{d}$ computes a *complete set of saturated constraints*, i.e. a set of saturated constraints from which all models can be read off in a simple way. In principle, there are at least two ways in which we could define completeness here: either as computing a set of saturated constraints describing all minimal models, or as computing all minimal saturated constraints. However there exists no natural notion of a minimal model. For example, the constraint

$$X{:}a \wedge Y{:}b$$

can have many "smallest" models, depending on the signature Σ. So we define completeness as *computing all minimal saturated constraints* for a given constraint. We define both minimality and the notion of a *minimal saturation for* a constraint in terms of a partial order on constraints.

Definition 3.15 (Minimal saturation for a constraint). *Let φ, ς be clauses over some first-order language $\mathcal{L}$, S a set of saturation rules and $\preceq$ a partial order on clauses over $\mathcal{L}$. Then ς is an* S-saturated constraint for φ with respect to $\preceq$ *iff ς is an S-saturated constraint with $\varphi \preceq \varsigma$, and ς is a* $\preceq$-minimal S-saturated constraint for φ *iff ς is $\preceq$-minimal with the property of being an S-saturated constraint for φ with respect to $\preceq$.*

For $\mathcal{C}_\mathrm{d}$ constraints, the partial order that we use is simply *subset inclusion*. So a $\mathcal{C}_\mathrm{d}$ constraint ς is a $\mathcal{P}_\mathrm{d}$-saturated constraint for a constraint φ iff $\varphi \subseteq \varsigma$, and it is a minimal $\mathcal{P}_\mathrm{d}$-saturated constraint for φ iff there exists no $\mathcal{P}_\mathrm{d}$-saturated constraint ς' for φ with $\varsigma' \subset \varsigma$.

Definition 3.16 (Completeness). *We call a saturation procedure* complete *with respect to a partial order $\preceq$ on clauses iff it computes all* $\preceq$-minimal saturated constraints *for any given clause.*

We show that given a $\mathcal{C}_\mathrm{d}$ constraint φ and a minimal saturated constraint ς for it, $\mathcal{P}_\mathrm{d}$ can compute ς from φ: If a saturation rule is applicable to φ, we can apply it in such a way that we stay in a subset of ς.

Lemma 3.17 (Completeness). *Let φ be a $\mathcal{C}_\mathrm{d}$ constraint and ς a minimal $\mathcal{P}_\mathrm{d}$-saturated constraint for φ. Then $\varphi \rightarrow^*_{\mathcal{P}_\mathrm{d}} \varsigma$.*

Proof. By well-founded induction on the strict partial order $\supset$ on the set $\{\varphi' \mid \varphi' \subseteq \varsigma\}$. If φ is $\mathcal{P}_\mathrm{d}$-saturated then $\varphi \rightarrow^*_{\mathcal{P}_\mathrm{d}} \varphi = \varsigma$ by minimality and we are done. Otherwise, there is a rule $\rho = \varphi_0 \rightarrow \vee_{i=1}^n \varphi_i$ in $\mathcal{P}_\mathrm{d}$ that applies to φ. Since $\varphi_0 \subseteq \varphi \subseteq \varsigma$ and ς is $\mathcal{P}_\mathrm{d}$-saturated, there exists an i such that $\varphi_i \subseteq \varsigma$. The constraint $\varphi \wedge \varphi_i$ is strictly bigger than φ, otherwise ρ would not apply to φ – see the application condition appc_ρ in Def. 3.1 (p. 58). Furthermore by Lemma 3.5, $\mathcal{P}_\mathrm{d}$ always terminates. Hence the inductive hypothesis already holds for $\varphi \wedge \varphi_i$: We have $\varphi \wedge \varphi_i \rightarrow^*_{\mathcal{P}_\mathrm{d}} \varsigma$ and thus $\varphi \rightarrow^*_{\mathcal{P}_\mathrm{d}} \varsigma$. □

So $\mathcal{P}_\mathrm{d}$ can compute all minimal saturated constraints for a given constraint. However, it does not compute *only* minimal saturated constraints. The only saturation rules that can lead to nonminimal saturated constraints are distribution rules where the right-hand side disjuncts are not exclusive. The algorithm $\mathcal{P}_\mathrm{d}$ possesses exactly one such rule, (D.distr.notDisj). Suppose we apply it to the constraint

$$X \triangleleft^* Z \wedge Y \triangleleft^* Z \wedge X{:}a.$$

This yields two clauses: One of them contains $X \triangleleft^* Y$, and the other contains $Y \triangleleft^* X$. For the clause containing $X \triangleleft^* Y$, we can now apply (D.distr.child) to $X{:}a \wedge X \triangleleft^* Y$, yielding $X{=}Y$. We apply (D.distr.child) again to the same clause, this time to $X{:}a \wedge X \triangleleft^* Z$, which gives us $X{=}Z$. In the other clause, the one that contains $Y \triangleleft^* X$, we also get $X{=}Z$ by (D.distr.child),but not $X{=}Y$ – this second clause is a proper subset of the first clause.

It is easy to show that each model of a constraint is also a model of one of its minimal saturated constraints.

Proposition 3.18. *Let φ be a $\mathcal{C}_\mathrm{d}$ constraint and (θ, σ) a model for φ. Then φ possesses a minimal $\mathcal{P}_\mathrm{d}$-saturated constraint that is also satisfied by (θ, σ).*

Proof. Let ς be φ extended by all literals entailed by (θ, σ). ς is satisfiable – it is satisfied by (θ, σ). It is also a saturated constraint since each saturation rule only adds entailed constraints. There must be a minimal saturated constraint ς' for φ with $\varsigma' \subseteq \varsigma$: either it is ς itself, or there exists some $\varsigma' \subset \varsigma$ such that ς' is a $\mathcal{P}_\mathrm{p}$-saturated constraint but no $\varsigma'' \subset \varsigma'$ is. □

3.5 Recapitulation: Properties of the Algorithm $\mathcal{P}_\mathrm{d}$

In the previous sections we have shown a number of properties of the algorithm $\mathcal{P}_\mathrm{d}$, which we now sum up.

Theorem 3.19. *The dominance constraint solver $\mathcal{P}_\mathrm{d}$ has the following properties:*

1. *It is sound for lambda structures, i.e. all its rules are equivalence transformations.*
2. *It is terminating.*
3. *For each $\mathcal{C}_\mathrm{d}$ constraint it computes a finite set of saturations.*
4. *Each $\mathcal{P}_\mathrm{d}$-saturated $\mathcal{C}_\mathrm{d}$-constraint is satisfiable.*
5. *$\mathcal{P}_\mathrm{d}$ is complete: Given a $\mathcal{C}_\mathrm{d}$ constraint φ, $\mathcal{P}_\mathrm{d}$ computes all minimal $\mathcal{P}_\mathrm{d}$-saturations for φ.*

Proof. 1. by Lemma 3.4, 2. by Lemma 3.5, 3. by Lemma 3.6, 4.by Lemma 3.14, and 5. by Lemma 3.17. □

3.6 Related Approaches

Rogers and Vijay-Shanker [102] study a feature logic that contains literals expressing dominance, equality, parenthood, and precedence ("left-of"), and allows for arbitrary logical connectives (including negation). They discuss an algorithm for deriving a set of quasi-trees [115] equivalent to a given description, and then an algorithm for reading off satisfying trees from the quasi-trees. As we have remarked before, quasi-trees are graphs that are very similar to graphs for $\mathcal{P}_\mathrm{d}$-saturated constraints, except that in a quasi-tree no node has more than one dominance child. The algorithm that transforms a tree description into a set of quasi-trees is formulated as a resolution proof system and uses treeness axioms as inference rules. Note that this algorithm computes saturated constraints with a unique minimal model: Quasi-trees can be characterized by the fact that each of them has one unique minimal satisfying tree.

Cornell [27] discusses a tree description language that contains relations expressing dominance, precedence and their inverses, furthermore equality, plus disjunctions of all these relations – however this language does not comprise labeling. For example a constraint $dep(x, y)$ states that x either dominates, equals, or precedes y. Satisfiability of these constraints can be tested in quadratic time, as Bodirsky and Kutz recently showed [14], using a greedy top-down tree construction algorithm.

Duchier and Gardent [33] consider a sublanguage of $\mathcal{C}_\mathrm{d}$: constraints that are conjunctions of dominance and labeling literals. They use a constraint programming approach to solve these constraints. The central idea is to represent the relative position of variables via four sets for each variable X. These sets contain the variables that may be above X, below it, equal to it, and in a disjoint position. Propagators then reduce the number of possible relations between each pair of variables.

Duchier and Niehren [34] take the same constraint programming approach as Duchier and Gardent, using the four position sets. The language that they work with is $\mathcal{C}_\mathrm{d}$ plus

set operators. It allows for constraints XRY, where R is a set of relations. This $\mathcal{C}_\mathrm{d}$ solver has been implemented in the language Oz [111]. The implementation centrally uses finite set constraints and disjunctive propagators. The implementation forms part of a system that demonstrates the use of CLLS in underspecified semantics [21]. Duchier and Niehren show that this constraint programming algorithm is equivalent to a saturation-based one. The saturation algorithm uses rules similar to those we have discussed in this chapter, but allows for stronger propagation. For example, in the second example of Sec. 3.1.2, Fig. 3.4, $\mathcal{P}_\mathrm{d}$ has to use distribution to see that the two fragments for "every plan" and "a catch" cannot overlap, while the saturation algorithm by Duchier and Niehren can determine this by propagation.

Duchier and Thater [35] transfer this approach to d-tree grammar by introducing "electrostatic trees", dominance and labeling constraints in which the variables have positive or negative polarities. Negative variables are similar to holes in Hole Semantics [15], they have to be "plugged" by positive variables. The main difference to the algorithm by Duchier and Gardent is that Duchier and Thater regard not one constraint but a disjunction of constraints: They use the algorithm for parsing, the parser has to choose between different lexical entries for each word, and each lexical entry contributes a different constraint.

Satisfiability of $\mathcal{C}_\mathrm{d}$ is an NP-complete problem [78], but only because different fragments of the constraint can overlap. Althaus, Duchier, Koller, Mehlhorn, Niehren and Thiel [3] define a sublanguage of $\mathcal{C}_\mathrm{d}$, the language of *normal dominance constraints*, for which satisfiability is testable in polynomial time. A normal dominance constraint consists of a set of fragments plus a set of dominance edges between the fragments. The fragments can never overlap, and solving such constraints means deriving a partial order of the fragments that respects all dominance edges. For the satisfiability test, the problem is reduced to the weighted perfect matching problem on graphs.

3.7 Summary

In this chapter we have discussed the constraint solver $\mathcal{P}_\mathrm{d}$ for $\mathcal{C}_\mathrm{d}$. The solver processes a set of clauses (= constraints), saturating them until nothing new can be added anymore. Saturation is a simple paradigm, it retains all information it has ever gathered without trying for optimizations.

The algorithm $\mathcal{P}_\mathrm{d}$ extends a set of clauses, which initially consists of one input clause, using saturation rules that enforce treeness. In particular, the algorithm applies distribution in two situations: when two variables both dominate a third, and when a dominance "hangs off" a labeled variable.

A constraint that is saturated under $\mathcal{P}_\mathrm{d}$ is one to which no saturation rule applies anymore. It can be characterized as a forest of trees with two kinds of edges, labeling and dominance edges, where each node has at most one kind of outgoing edges, and outgoing labeling edges are ordered, while outgoing dominance edges are not. We have shown that

each $\mathcal{P}_{\mathrm{d}}$-saturated constraint is satisfiable: If its constraint graph is already tree-shaped, then a model can be read off right away. Otherwise we can transform it by successively labeling unlabeled nodes in such a way that the constraint stays $\mathcal{P}_{\mathrm{d}}$-saturated.

The constraint solver is terminating, and it is sound in the sense that all rules are equivalence transformations. It is also complete: It computes all $\subseteq$-minimal saturated constraints for a given constraint, i.e. all $\subseteq$-minimal supersets of the constraint that are saturations. We have shown completeness by proving that given a constraint φ, a minimal saturated constraint for it, and a saturation rule applicable to φ, we can apply the rule in such a way that the result of the rule application is still a subset of the minimal saturated constraint.

Chapter 4

Solving Parallelism Constraints

In this chapter we present a semi-decision procedure for parallelism constraints. The procedure incorporates the dominance constraint solver of the previous chapter. It is again a saturation procedure, which keeps adding material to a set of clauses until a state of saturation is reached in which nothing new can be added anymore. For satisfiable parallelism constraints the procedure computes saturated constraints from which models can be read off directly.

A parallelism literal states that two segments are structurally isomorphic. In Chapter 2 we have defined the parallelism relation in terms of *correspondence functions*, which link corresponding nodes in the two parallel segments. It is the same idea that we now use for our semi-decision procedure, in the form of *correspondence formulas*. A correspondence formula links "corresponding variables", i.e. variables that will denote corresponding nodes. We express correspondence between variables in terms of a new type of literals, *path parallelism* literals. The path parallelism relation is very similar to the parallelism relation, except that it talks about tree paths instead of segments.

The proofs of completeness and satisfiability of saturated constraints have the same basic structure as in the previous chapter. But especially the completeness proof is more interesting in the current case, as we now have saturation rules that introduce additional existentially quantified variables. The proof that saturated constraints are reached after a finite number of steps, which was trivial in the case of $\mathcal{P}_\mathrm{d}$, becomes much more intricate now.

4.1 A Semi-Decision Procedure for Parallelism Constraints: $\mathcal{P}_\mathrm{p}$

It is trivial to formulate a semi-decision procedure for the language $\mathcal{C}_\mathrm{p}$ of parallelism constraints: Just enumerate lambda structures and check for each if it satisfies the given constraint. But such a procedure is of course not satisfactory – it is neither feasible, nor does it provide insights into the nature of the problem. In contrast, the procedure that we introduce in this section

- terminates for the linguistically relevant constraints and computes saturations that correspond to the correct readings.

- introduces *correspondence formulas* as a data structure for handling parallelism within partial tree descriptions.
- includes an algorithm for solving dominance constraints. Given a dominance constraint as an input, the parallelism constraint procedure behaves exactly like the dominance constraint solver that it encompasses. This is advantageous because, as we have seen in Chapter 2, dominance constraints play an important role in the linguistic application.
- is built in a modular fashion: a different dominance constraint solver can be substituted for the one we use here. For example, the saturation algorithm of Duchier and Niehren [34], which needs less distribution, can be employed. Actually, a recent overview paper on processing CLLS [44] combines this latter dominance constraint solver with the rules for parallelism that we present in this chapter.

We extend the dominance constraint solver of the previous chapter to a saturation-based semi-decision procedure for $\mathcal{C}_{\mathrm{p}}$. As before, the procedure works on a set of clauses (constraints). Whenever a clause contains the left-hand side of a saturation rule but not the right-hand side, the right-hand side can be added. By applying a deterministic rule we just extend this one clause. By applying a distribution rule we replace the clause by a set of new ones, where each new one consists of the old clause extended by one right-hand side disjunct. This is exactly as in the previous chapter. However now the saturation rules have a slightly more general form: They may introduce additional existentially quantified variables. The saturation rules that we use now have the form

$$\rho : \varphi_0 \rightarrow \bigvee_{i=1}^{n} \exists V_i \varphi_i \text{ if } \mathrm{appc}_\rho$$

for clauses $\varphi_0, \ldots, \varphi_n$, $n \geq 1$, (possibly empty) sets $V_1, \ldots, V_n \subseteq \mathcal{V}ar$ of variables, and $\mathcal{V}ar(\varphi_i) - \mathcal{V}ar(\varphi_0) \subseteq V_i$ for all $1 \leq i \leq n$. appc_ρ is the application condition of the rule ρ. As in the previous chapter, it states that ρ can be applied only to a clause to which it adds something new. But we have to adapt the application condition to the changed shape of the saturation rules. Given a set V of variables and a constraint φ, we call a constraint $\sigma\varphi$ a *V-variant of φ* if $\sigma : V \rightarrow \mathcal{V}ar$ is a renaming of the variables in V. We call this variant *fresh* if $\sigma(V)$ is disjoint from $\mathcal{V}ar(\varphi)$.

Definition 4.1 (Saturation step, application condition). *Let $\mathcal{S}$ be a set of saturation rules. A* saturation step $\rightarrow_{\mathcal{S}}$ *consists of one application of a rule $\rho \in \mathcal{S}$. Let $\rho = \varphi_0 \rightarrow \bigvee_{i=1}^{n} \exists V_i \varphi_i$ for clauses $\varphi_0, \ldots, \varphi_n$. Then*

$$\frac{\varphi_0 \subseteq \varphi}{\varphi \rightarrow_{\mathcal{S}} \varphi \wedge \varphi_i'} \textit{for } i \in \{1, \ldots, n\} \textit{ if } \mathrm{appc}_\rho \textit{ and } \varphi_i' \textit{ is a fresh } V_i\textit{-variant of } \varphi_i.$$

where the application condition is

$$\mathrm{appc}_\rho(\varphi) =_{\mathrm{def}} \textit{ for all } 1 \leq i \leq n \textit{ and all } V_i\textit{-variants } \varphi_i'' \textit{ of } \varphi_i : \varphi_i'' \not\subseteq \varphi$$

To make the saturation rules easier to read, we introduce formulas for some constraints that we will use repeatedly. These formulas contain disjunctions. If a saturation rule contains such a formula on the right-hand side, it is a distribution rule. However if such a formula occurs on the left-hand side of a rule, it abbreviates a set of saturation rules, as illustrated in Fig. 4.1. Note that this unfolding of rule abbreviations may have to be iterated.

$(\varphi_1 \ \vee \ \varphi_2) \ \wedge \ \varphi_3 \to \varsigma$	abbreviates	$\varphi_1 \wedge \varphi_3 \ \to \ \varsigma$ $\varphi_2 \wedge \varphi_3 \ \to \ \varsigma$

Figure 4.1: Using a disjunctive formula on the left-hand side of a rule

4.1.1 Parallelism Literals and Symmetry

In Chapter 2 we have said that we regard inequality and disjointness literals as symmetric; but we do not regard parallelism literals as symmetric. This is because by Def. 2.7 (p. 29) the conditions on lambda binding are symmetric, but the conditions on anaphoric binding are not: Anaphoric binding in the "source segment term" imposes restrictions on anaphoric binding in the "target segment term", but not vice versa.

However, for all purposes except anaphoric binding, we make no difference between the left and the right segment term of a parallelism literal. Let $A = X_0/X_1, \ldots, X_n$, $B = Y_0/Y_1, \ldots, Y_n$ be segment terms. Then we introduce the following formula for "symmetric parallelism":

$$A \sim^{\text{sym}} B =_{\text{def}} A \sim B \ \vee \ B \sim A.$$

In the current chapter we will use this symmetric parallelism formula throughout. It is only in the following chapter, when we extend the procedure to handle binding literals, that we will make use of the asymmetry of parallelism literals.

4.1.2 Correspondence Formulas and Path Parallelism

The procedure for $\mathcal{C}_{\text{p}}$ that we present in this chapter solves parallelism literals by computing a syntactic equivalent of the correspondence functions of Def. 2.3 (p. 27): *correspondence formulas*, which we also call *syntactic correspondence functions*. Two variables are linked by a correspondence formula if they denote corresponding nodes.

We express syntactic correspondence in terms of a new type of literals, *path parallelism* literals. The path parallelism relation states that two tree paths are the same, as well as the labels encountered on the paths. Figure 4.2 illustrates this.

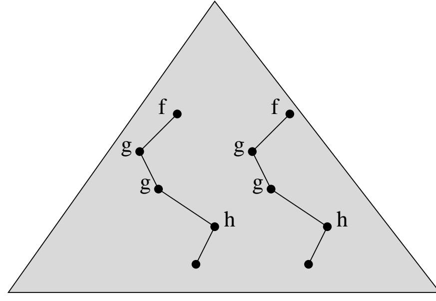


Figure 4.2: Path parallelism

Definition 4.2 (Path parallelism). *Let θ be a tree structure.* Path parallelism *is the largest relation on 4-tuples of paths from D_θ such that $p(\begin{smallmatrix}\pi_1 & \psi_1\\ \pi_2 & \psi_2\end{smallmatrix})$ holds in θ iff*

- *there exists a path π such that $\pi_2 = \pi_1\pi$ and $\psi_2 = \psi_1\pi$, and*
- *for each proper prefix π' of π, $L_\theta(\pi_1\pi') = L_\theta(\psi_1\pi')$.*

Path parallelism is the restriction of parallelism from segments to paths. In particular, note that the labels of π_2 and ψ_2 need not be identical, only the labels of all nodes encountered on the paths before π_2 and ψ_2. Correspondence functions can be expressed in terms of path parallelism:

Proposition 4.3. *Given a tree structure θ with segments α, β such that there exists a correspondence function c between α and β in θ. Then for all nodes π, ψ of θ,*

$$co(\alpha,\beta)(\pi)=\psi \text{ iff } \pi \in \mathsf{b}(\alpha) \text{ and } p(\begin{smallmatrix}r(\alpha) & r(\beta)\\ \pi & \psi\end{smallmatrix})$$

Proof. We proceed by induction on the length of the path from $r(\alpha)$ to π, abbreviating $\text{co}(\alpha,\beta)$ by c for increased readability.

"⇒" Concerning a path π of length 0, we have $\pi = r(\alpha)$, and $c(r(\alpha)) = r(\beta)$. But $r(\alpha) \in \mathsf{b}(\alpha)$ and $\text{p}(\begin{smallmatrix}r(\alpha) & r(\beta)\\ r(\alpha) & r(\beta)\end{smallmatrix})$ hold trivially in θ.

Now let $\pi \in \mathsf{b}^-(\alpha)$ with $\theta \models \pi{:}f(\pi_1,\ldots,\pi_n)$, such that $\text{p}(\begin{smallmatrix}r(\alpha) & r(\beta)\\ \pi & c(\pi)\end{smallmatrix})$ holds in θ. By Def. 2.3, we have $\theta \models c(\pi){:}f(c(\pi_1),\ldots,c(\pi_n))$. But that already means that $\text{p}(\begin{smallmatrix}r(\alpha) & r(\beta)\\ \pi_i & c(\pi_i)\end{smallmatrix})$ must hold in θ for $1 \le i \le n$.

"⇐": For the case of $\pi = r(\alpha)$, if $\text{p}(\begin{smallmatrix}r(\alpha) & r(\beta)\\ \pi & \psi\end{smallmatrix})$ holds then $\psi = r(\beta)$, and by Def. 2.3, $c(r(\alpha)) = r(\beta)$.

For the inductive step, suppose $\text{p}(\begin{smallmatrix}\pi_1 & \psi_1\\ \pi & \psi\end{smallmatrix})$ holds in θ for $\pi \in \mathsf{b}(\alpha)$ and $\pi \neq r(\alpha)$. Then by the definition of path parallelism (Def. 4.2), there are nodes π', ψ' such that $\text{p}(\begin{smallmatrix}\pi_1 & \psi_1\\ \pi' & \psi'\end{smallmatrix})$ holds in θ, π' and ψ' bear the same label, π is the i-th child of π' and ψ the

i-th child of ψ' for some i. The inductive hypothesis applies to π' since π' is strictly shorter than π and $\pi' \in \mathsf{b}(\alpha)$ (since $\pi \in \mathsf{b}(\alpha)$), so $c(\pi') = \psi'$, whence $c(\pi) = \psi$ by Def. 2.3.

□

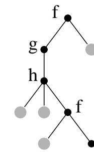
On the syntactic side, we extend our constraint languages by *path parallelism literals* of the form

$$\mathrm{p}(\begin{smallmatrix} X & Y \\ X' & Y' \end{smallmatrix}).$$

For short, we also call them *path literals*. Like inequality and disjointness literals, path literals are symmetric.

We write $\mathcal{C}_{\mathrm{pp}}$ for the language $\mathcal{C}_{\mathrm{p}}$ extended by path literals, and $\mathrm{CLLS}_{\mathrm{p}}$ for CLLS extended by path literals.

Interestingly, it is not clear whether path literals can be expressed in terms of parallelism literals. While it is true that path parallelism is the restriction of parallelism to path-shaped segments, a path literal is not just a restricted form of a parallelism literal. A parallelism literal always specifies the (maximum) number of exceptions in the two segments that it is about, while a path literal does not restrict the number of exceptions in the two path-shaped segments that it describes. See the figure to the right for an example of a path-shaped segment and its exceptions, here drawn as shaded circles.



We use path literals to express syntactic correspondence by some formulas that we introduce now. The fact that some segment term $A = X_0/X_1, \ldots, X_n$ denotes a segment is stated by the formula

$$\mathsf{seg}(A) =_{\mathrm{def}} \bigwedge_{i=1}^{n} X_0 \triangleleft^* X_i \wedge \bigwedge_{1 \le i < j \le n} \big((X_i \perp X_j) \vee (X_i = X_j)\big).$$

(As defined in in Chapter 2, p. 31, two hole variables of a segment term may denote the same hole node, and the order of hole variables need not match the order of the hole nodes.) Given a segment term A and a variable X of a CLLS constraint, we do not always know whether the denotation of X lies inside the denotation of A. The cases where we do know can be described by the following formulas: Let A be as above, then

$$\begin{array}{rcl} X \in \mathsf{b}(A) & =_{\mathrm{def}} & X_0 \triangleleft^* X \;\wedge\; \bigwedge_{i=1}^{n} (X \triangleleft^* X_i \vee X \perp X_i) \\ X \in \mathsf{b}^-(A) & =_{\mathrm{def}} & X \in \mathsf{b}(A) \wedge \bigwedge_{i=1}^{n} (X \neq X_i \vee X \perp X_i) \\ X \notin \mathsf{b}(A) & =_{\mathrm{def}} & X \triangleleft^+ X_0 \vee X \perp X_0 \vee \bigvee_{i=1}^{n} X_i \triangleleft^+ X \\ X \notin \mathsf{b}^-(A) & =_{\mathrm{def}} & X \triangleleft^+ X_0 \vee X \perp X_0 \vee \bigvee_{i=1}^{n} X_i \triangleleft^* X \end{array}$$

So $X \in \mathsf{b}(A)$ is a disjunction of constraints. Still, we write "$X \in \mathsf{b}(A)$ is in φ" to express that one of the disjuncts in $X \in \mathsf{b}(A)$ is contained in the constraint φ, and analogously for

the other formulas that we have just defined. Also, we sometimes write "X is inside A" instead of "$X \in \mathsf{b}(A)$ is in the constraint we currently consider". Note that the negative formula $X \notin \mathsf{b}(A)$ expresses that we know for sure that the denotation of X cannot be inside the denotation of A, and analogously for $X \notin \mathsf{b}^-(A)$.

Now we can define correspondence formulas. Let $A = X_0/X_1, \ldots, X_n$ and $B = Y_0/Y_1, \ldots, Y_n$. Then

$$\mathrm{co}(A,B)(U){=}V \quad =_{\mathrm{def}} \quad A{\sim}^{\mathrm{sym}}B \ \wedge \ \ \mathrm{p}\left(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix}\right) \ \ \wedge \ U \in \mathsf{b}(A).$$

co(A,B) is the syntactic correspondence function for the two parallel segment terms A and B.

4.1.3 The Rules in Detail

Remember that the class $\mathcal{C}_\mathrm{p}$ of parallelism constraints has the following abstract syntax:

$$\begin{array}{lcll} \varphi, \varsigma & ::= & X{\triangleleft^*}Y \mid X{:}f(X_1,\ldots,X_n) \mid X{\perp}Y \mid X{\neq}Y & (\mathsf{ar}(f) = n) \\ & \mid & X_0/X_1,\ldots,X_n{\sim}Y_0/Y_1,\ldots,Y_n & n \geq 0 \\ & \mid & \mathtt{false} \mid \varphi \ \wedge \ \varsigma & \end{array}$$

The semi-decision procedure $\mathcal{P}_\mathrm{p}$ for parallelism constraints is shown in Fig. 4.3. The first block of rules infers a syntactic correspondence function co(A,B) for each parallelism literal $A \sim B$ and copies constraints from variables to their correspondents.[1] The rule (P.init) has the form $A{\sim}B \rightarrow \mathsf{seg}(A) \wedge \mathsf{seg}(B) \wedge \mathrm{co}(A,B)(X_i){=}Y_i$ for segment terms $A = X_0/X_1,\ldots,X_n$ and $B = Y_0/Y_1,\ldots,Y_n$ and $0 \leq i \leq n$. It makes sure that two parallel segment terms A and B denote segments, and it fixes some correspondences in the syntactic correspondence function co(A,B): the root variables of A and B must correspond, and the i-th hole variable of A corresponds to the i-th hole of B for all i. For all other variables inside A or B, (P.new) introduces a new existentially quantified variable as a correspondent: It has the form $A{\sim}^{\mathrm{sym}}B \ \wedge \ U \in \mathsf{b}(A) \rightarrow \exists U'.\mathrm{co}(A,B)(U){=}U'$. The rule (P.copy.dom), which is $U_1RU_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \rightarrow V_1RV_2$ for $R \in \{\triangleleft^*, \perp, \neq\}$, copies dominance, disjointness, and inequality literals from variables U_1, U_2 to variables corresponding to U_1, U_2. Note that this rule only applies if V_1, V_2 are correspondents of U_1, U_2 by the same syntactic correspondence function co(A,B). Likewise, (P.copy.lab), which states $U_0{:}f(U_1,\ldots,U_m) \ \wedge \ \bigwedge_{i=0}^{m} \mathrm{co}(A,B)(U_i){=}V_i \ \wedge \ U_0 \in \mathsf{b}^-(A) \rightarrow V_0{:}f(V_1,\ldots,V_m)$, copies labeling literals from variables $U_0,\ldots,U_m$ to their correspondents. This rule additionally makes sure that U_0 is not a hole of A: By Def. 2.3 and 2.4 (p. 27), two parallel segments are isomorphic only up to their holes, excluding the hole labels.

The procedure $\mathcal{P}_\mathrm{p}$ contains two distribution rules in addition to the ones of $\mathcal{P}_\mathrm{d}$, listed in the second block in Fig. 4.3. (P.distr.seg) has the form $A{\sim}^{\mathrm{sym}}B \ \wedge \ X_0{\triangleleft^*}X \rightarrow X \in$

[1] Note that a variable may have more than one correspondent, even with respect to the same syntactic correspondence function co(A,B). But if a constraint contains $\mathrm{co}(A,B)(U) = V_1$ and $\mathrm{co}(A,B)(U) = V_2$, then its saturation will also contain $V_1{=}V_2$ by (P.trans.h) and (P.path.eq.2).

Let $A = X_0/X_1, \ldots, X_n$ and $B = Y_0/Y_1, \ldots, Y_n$.

Core Rules

(P.init) $A{\sim}B \to \mathsf{seg}(A) \wedge \mathsf{seg}(B) \wedge \mathrm{co}(A,B)(X_i){=}Y_i$ where $0 \le i \le n$

(P.new) $A{\sim}^{\mathrm{sym}}B \wedge U \in \mathsf{b}(A) \to \exists U'.\mathrm{co}(A,B)(U){=}U'$ where U' is a fresh variable

(P.copy.dom) $U_1 R U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \to V_1 R V_2$ where $R \in \{\triangleleft^*, \perp, \neq\}$

(P.copy.lab) $U_0{:}f(U_1,\ldots,U_m) \wedge \bigwedge_{i=0}^{m} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_0 \in \mathsf{b}^-(A) \to V_0{:}f(V_1,\ldots,V_m)$

Additional Distribution Rules

(P.distr.seg) $A{\sim}^{\mathrm{sym}}B \wedge X_0 \triangleleft^* X \to X \in \mathsf{b}(A) \;\vee\; \bigvee_{j=1}^{n} X_j \triangleleft^+ X$

(P.distr.eq) $\varphi \to X{=}Y \;\vee\; X{\neq}Y$ where $X,Y \in \mathcal{V}ar(\varphi)$

Rules concerning Path Parallelism

(P.path.dom) $\mathrm{p}\binom{X\;Y}{U\;V} \to X \triangleleft^* U \wedge Y \triangleleft^* V$

(P.path.eq.1) $\mathrm{p}\binom{X_1\;X_3}{X_2\;X_4} \wedge \bigwedge_{i=1}^{4} X_i{=}Y_i \to \mathrm{p}\binom{Y_1\;Y_3}{Y_2\;Y_4}$

(P.path.eq.2) $\mathrm{p}\binom{X\;X}{U\;V} \to U{=}V$

(P.trans.h) $\mathrm{p}\binom{X\;Y}{U\;V} \wedge \mathrm{p}\binom{Y\;Z}{V\;W} \to \mathrm{p}\binom{X\;Z}{U\;W}$

(P.trans.v) $\mathrm{p}\binom{X_1\;Y_1}{X_2\;Y_2} \wedge \mathrm{p}\binom{X_2\;Y_2}{X_3\;Y_3} \to \mathrm{p}\binom{X_1\;Y_1}{X_3\;Y_3}$

(P.diff.1) $\mathrm{p}\binom{X_1\;Y_1}{X_2\;Y_2} \wedge \mathrm{p}\binom{X_1\;Y_1}{X_3\;Y_3} \wedge X_2 \triangleleft^* X_3 \wedge Y_2 \triangleleft^* Y_3 \to \mathrm{p}\binom{X_2\;Y_2}{X_3\;Y_3}$

(P.diff.2) $\mathrm{p}\binom{X_1\;Y_1}{X_3\;Y_3} \wedge \mathrm{p}\binom{X_2\;Y_2}{X_3\;Y_3} \wedge X_1 \triangleleft^* X_2 \wedge Y_1 \triangleleft^* Y_2 \to \mathrm{p}\binom{X_1\;Y_1}{X_2\;Y_2}$

plus the rules of the dominance constraint solver $\mathcal{P}_{\mathrm{d}}$ in Fig. 3.2, p. 60.

Figure 4.3: Solving $\mathcal{C}_{\mathrm{p}}$ constraints: procedure $\mathcal{P}_{\mathrm{p}}$.

$\mathsf{b}(A) \;\vee\; \bigvee_{j=1}^{n} X_j \triangleleft^+ X$. It deals with situations like the one in Fig. 4.4 (where the two segment terms of the parallelism literal are visualized by brackets from root to hole variables): Here we have to decide whether U is in $\mathsf{b}(X_0/X_1)$ or not, such that we know whether or not to apply (P.new) to U. (P.distr.eq) is a projection rule: Stating $\varphi \rightarrow X{=}Y \;\vee\; X{\neq}Y$ for $X, Y \in \mathcal{V}ar(\varphi)$, it guesses whether two variables should be identified or not.

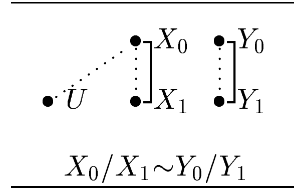
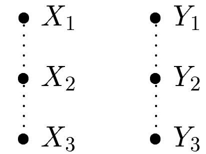


Figure 4.4: Constraint graph illustrating (P.distr.seg)

The saturation rules of the last block state properties of path parallelism. They ensure that correspondence formulas correctly mirror correspondence functions. (P.path.dom), which is $\mathrm{p}\binom{X\;Y}{U\;V} \rightarrow X \triangleleft^* U \;\wedge\; Y \triangleleft^* V$, states that the existence of a path from X to U implies dominance. By (P.path.eq.1), given a path literal we can add another in which equal variables have been substituted: $\mathrm{p}\binom{X_1\;X_3}{X_2\;X_4} \;\wedge\; \bigwedge_{i=1}^{4} X_i{=}Y_i \;\rightarrow\; \mathrm{p}\binom{Y_1\;Y_3}{Y_2\;Y_4}$. At the same time, this rule ensures that syntactic correspondence correctly models the fact that (semantic) correspondence is a function: If two variables are equal, they will be mapped to the same corresponding variable. (P.path.eq.2), which has the form $\mathrm{p}\binom{X\;X}{U\;V} \rightarrow U{=}V$, states that two parallel paths starting at the same point must end at the same point. The rule (P.trans.h), which is $\mathrm{p}\binom{X\;Y}{U\;V} \;\wedge\; \mathrm{p}\binom{Y\;Z}{V\;W} \rightarrow \mathrm{p}\binom{X\;Z}{U\;W}$, expresses horizontal transitivity: if a path is parallel to a second one, which again is parallel to a third one, then the first and third paths are also parallel.



The rules (P.trans.v), (P.diff.1) and (P.diff.2) are all concerned with vertical transitivity. These rules are illustrated in the figure to the right: There are variables X_1, X_2, X_3 with X_1 dominating X_2 and X_2 dominating X_3, and Y_1, Y_2, Y_3 with Y_1 dominating Y_2 and Y_2 dominating Y_3. (P.trans.v) states $\mathrm{p}\binom{X_1\;Y_1}{X_2\;Y_2} \;\wedge\; \mathrm{p}\binom{X_2\;Y_2}{X_3\;Y_3} \rightarrow \mathrm{p}\binom{X_1\;Y_1}{X_3\;Y_3}$: If the "short" path from X_1 to X_2, or $X_1 - X_2$ for short, is parallel to $Y_1 - Y_2$, and the short paths $X_2 - X_3$ and $Y_2 - Y_3$ are parallel as well, then the two long paths $X_1 - X_3$ and $Y_1 - Y_3$ are also parallel. The other two rules make similar statements: if two of the path pairs are parallel, then so is the third. The rule (P.diff.1), which is $\mathrm{p}\binom{X_1\;Y_1}{X_2\;Y_2} \;\wedge\; \mathrm{p}\binom{X_1\;Y_1}{X_3\;Y_3} \;\wedge\; X_2 \triangleleft^* X_3 \;\wedge\; Y_2 \triangleleft^* Y_3 \rightarrow \mathrm{p}\binom{X_2\;Y_2}{X_3\;Y_3}$, says that if $X_1 - X_2$, $Y_1 - Y_2$ are parallel and $X_1 - X_3$, $Y_1 - Y_3$ are parallel too, then so are $X_2 - X_3$ and $Y_2 - Y_3$. And the rule (P.diff.2), which has the form $\mathrm{p}\binom{X_1\;Y_1}{X_3\;Y_3} \;\wedge\; \mathrm{p}\binom{X_2\;Y_2}{X_3\;Y_3} \;\wedge\; X_1 \triangleleft^* X_2 \;\wedge\; Y_1 \triangleleft^* Y_2 \rightarrow \mathrm{p}\binom{X_1\;Y_1}{X_2\;Y_2}$, concludes that $X_1 - X_2$, $Y_1 - Y_2$ must be parallel if first $X_1 - X_3$, $Y_1 - Y_3$ and second $X_2 - X_3$, $Y_2 - Y_3$ are.

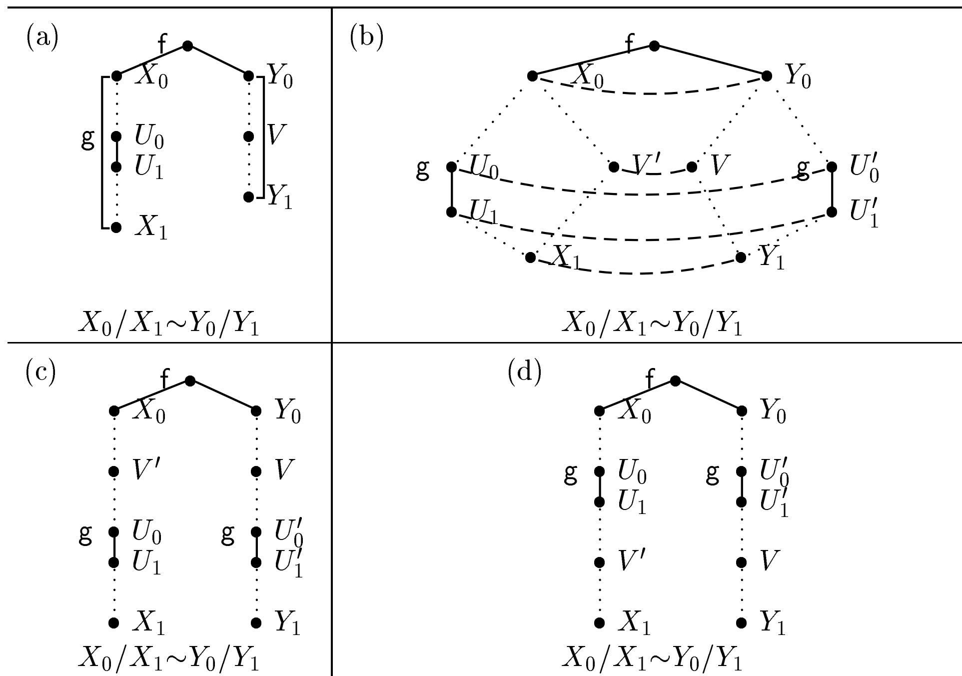


Figure 4.5: (a) A simple constraint with a parallelism literal, (b) a partial saturation with correspondences drawn in, and (c), (d) two further advanced partial saturations

4.1.4 Examples

Example 4.4 (Illustrating the core rules). We first demonstrate the core rules of $\mathcal{P}_\mathrm{p}$ (the first rule block in Fig. 4.3) and their interaction with the $\mathcal{P}_\mathrm{d}$ rules. We start out with the constraint in Fig. 4.5 (a). Then $\mathcal{P}_\mathrm{p}$ can perform the computation shown in Fig. 4.6. When a distribution rule increases the number of clauses, we indicate this by dividing the box in Fig. 4.6 vertically, with one column for each clause.

Rule (P.init) is applied in line (1) of Fig. 4.6: The roots X_0, Y_0 correspond, as do the holes X_1, Y_1. How do we go on from there? The variable U_0 is inside A, and V is inside B. But they are just *dominated* by X_0 and Y_0 respectively, their position is not fixed. Rule (P.new) gives both these variables, as well as U_1, correspondents (in lines (2) – (4) of Fig. 4.6), then leaves it to other rules to determine the positions of these correspondents. In lines (5) – (7), (P.copy.dom) positions the images U'_0, U'_1 within $\mathsf{b}(B)$, and the image V' inside $\mathsf{b}(A)$. Line (10) copies the label of U_0 to U'_0 by (P.copy.lab). The two preceding lines (8) and (9) make sure that (P.copy.lab) is applicable, i.e. U_0 is not the hole of A.

The resulting constraint is the one in Fig. 4.5 (b). In this graph, correspondence is indicated by dashed arcs. At this stage, all variables inside A or B possess a correspondent, but the constraint is not saturated yet. Lines (11) and (12) show what happens when we apply (D.distr.notDisj) now. If we choose the alternative $V' \triangleleft^* U_1$ (line (11a)), then (P.copy.dom) can immediately infer $V \triangleleft^* U'_1$. This gives us the constraint shown in Fig.

(1)	$X_0 \lhd^* X_1 \wedge Y_0 \lhd^* Y_1 \wedge$ $\text{co}(A,B)(X_0)=Y_0 \wedge \text{co}(A,B)(X_1)=Y_1$	(P.init)
(2)	$\exists U'_0.\text{co}(A,B)(U_0)=U'_0$	(P.new)
(3)	$\exists U'_1.\text{co}(A,B)(U_1)=U'_1$	(P.new)
(4)	$\exists V'.\text{co}(A,B)(V')=V$	(P.new)
(5)	$X_0 \lhd^* V',\ V' \lhd^* X_1$	(P.copy.dom)
(6)	$Y_0 \lhd^* U'_0, U'_0 \lhd^* Y_1$	(P.copy.dom)
(7)	$Y_0 \lhd^* U'_1, U'_1 \lhd^* Y_1$	(P.copy.dom)
(8)	$U_0 \lhd^+ U_1$	(D.lab.ineq)
(9)	$U_0 \lhd^+ X_1$	(D.dom.ineq)
(10)	$U'_0 : g(U'_1)$	(P.copy.lab)
(11)	$V' \lhd^* U_1 \vee U_1 \lhd^* V'$	(D.distr.notDisj)

(11a)	$V' \lhd^* U_1$:		(11b)	$U_1 \lhd^* V'$:	
(12)	$V \lhd^* U'_1$	(P.copy.dom)	(13)	$U'_1 \lhd^* V$	(P.copy.dom)

Figure 4.6: Computation of $\mathcal{P}_{\text{p}}$ on Fig. 4.5 (a)

4.5 (c). The alternative in (11b) is analogous, yielding the constraint depicted in Fig. 4.5 (d).

At this point (after lines (12) and (13) of the computation in Fig. 4.6 respectively) we have reached constraints that are almost saturated. It remains to apply (P.distr.eq) to guess which variables should be equal. We consider the constraint we have reached after line (12) of the computation, the one depicted in Fig. 4.5 (c). If (P.distr.eq) now guesses, for example, $X_0 \neq V', V' \neq U_0, U_1 \neq X_1$, then we get $Y_0 \neq V, V \neq U'_0, U'_1 \neq Y_1$ by (P.copy.dom) and a saturation that again is visualized by the constraint graph in Fig. 4.5 (c).

Example 4.5 (Quantifier Parallelism). In Section 2.3 we have discussed the phenomenon of *quantifier parallelism*: If a scope ambiguity occurs in the source sentence of an ellipsis, then this ambiguity has to be resolved the same way in the source and the target sentence, as witnessed e.g. by sentence (2.6) (p. 37), repeated here as (4.1).

(4.1) Every linguist attended a workshop. Every computer scientist did, too.

The constraint representing the meaning of this sentence is shown in Fig. 2.12, p. 37.

In the previous example we have discussed the mechanism that ensures that the "ambiguity" between V' and U_1 in Fig. 4.5 (b) is resolved the same way as the "ambiguity" between V and U'_1. The same mechanism sees to it that quantifier parallelism is handled correctly: When the ambiguity between the two scope-bearing expressions in the source sentence is resolved, the ambiguity between their copies in the target sentence has to be resolved *in the same way* because corresponding nodes must have the same positions within their respective segments.

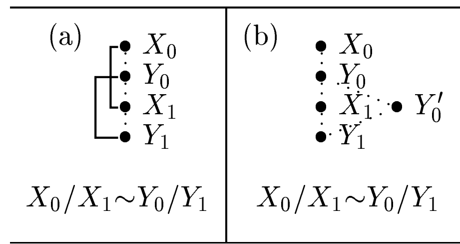
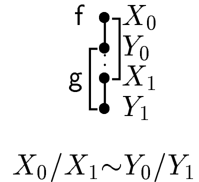


Figure 4.7: (a) Overlapping parallelism literal and (b) a partial saturation

Example 4.6 (Overlap and infinitely many saturations). Figure 4.7 (a) shows a very simple constraint in which the parallelism literal "overlaps itself". For this constraint the procedure $\mathcal{P}_\mathrm{p}$ computes infinitely many different saturations. One saturation contains $X_0{=}X_1{=}Y_0{=}Y_1$. Another contains $X_0 \triangleleft^+ X_1{=}Y_0 \triangleleft^+ Y_1$. If the constraint contains $X_0 \triangleleft^+ Y_0 \triangleleft^+ X_1 \triangleleft^+ Y_1$, then the variable $Y_0 \in \mathsf{b}(X_0/X_1)$ needs a correspondent by (P.new), e.g. $\mathrm{co}(X_0/X_1, Y_0/Y_1)(Y_0){=}Y_0'$, and (P.copy.dom) gives us the constraint depicted in Fig. 4.7 (b). Now (D.distr.notDisj) is applicable to Y_0' and X_1. It can either place Y_0' between X_1 and Y_1. Then Y_0' is inside Y_0/Y_1 but not inside X_0/X_1. Or (D.distr.notDisj) can place Y_0' between Y_0 and X_1, where it is in the "overlap region" belonging to both segment terms. Then Y_0' is inside X_0/X_1 and needs a correspondent in Y_0/Y_1, and so on.

Example 4.7 (Nontermination). The figure to the right shows an unsatisfiable constraint. For this constraint, $\mathcal{P}_\mathrm{p}$ does not terminate. It copies the f-label from X_0 to Y_0. But as Y_0 is inside X_0/X_1, it copies the f-label from Y_0 to Y_0's child. This child of Y_0 is again inside X_0/X_1, so the f-label gets copied from that variable to its child, and so on ad infinitum.



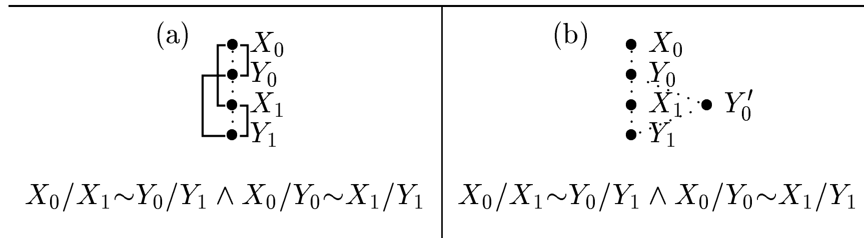


Figure 4.8: (a) Guessing equalities: For this constraint, (P.distr.eq) is needed. (b) A partial saturation.

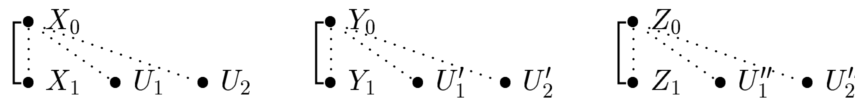
Example 4.8 (Guessing equalities). If we introduce a correspondent using (P.new), there are only two possibilities: Either the new variable will denote the same node as some variable that we already had in the constraint, or it will denote a node that did not interpret any variable of the constraint up to then. We definitely make *progress* in our computation when we decide which of the two cases applies. If we do not decide between

the two cases, we might end up inventing more and more new variables that in the end all denote the same node.

The rule (P.distr.eq) decides between the two cases by guessing equalities and inequalities between nodes. However, it is surprisingly hard to construct a constraint where this rule is actually needed – i.e. a constraint that is satisfiable but for which $\mathcal{P}_{\mathrm{p}}$ would never terminate without guessing equalities. Figure 4.8 (a) shows what seems to be the simplest such constraint. Remember that in discussing the "overlapping parallelism literal" constraint in Fig. 4.7 (a), we have called the part of the constraint between Y_0 and X_1 the "overlap region", which belongs to both segment terms of the parallelism literal, and we have distinguished it from the segment terms above Y_0 and below X_1, which both belong to only one of the two parallelism segment terms. Now in the constraint in Fig. 4.8 (a) , we have two parallelism literals and three "overlap regions".

Now suppose that again we want to find a correspondent for $Y_0 \in \mathsf{b}(X_0/X_1)$. We introduce a variable Y_0' with $\mathrm{co}(X_0/X_1, Y_0/Y_1)(Y_0){=}Y_0'$, and (P.copy.dom) adds $Y_0 \triangleleft^* Y_0', Y_0' \triangleleft^* Y_1$ as shown in Fig. 4.8 (b). Again (D.distr.notDisj) is applicable to Y_0' and X_1. If Y_0' is placed between Y_0 and X_1, it is inside Y_0/Y_1 and will need a correspondent inside X_0/X_1. And if Y_0' is placed between X_1 and Y_1, it is inside X_1/Y_1 and will need a correspondent inside X_0/Y_0. So wherever Y_0' is placed, (P.new) is again applicable to it.

The constraint in Fig. 4.8 (a) is satisfiable, and $\mathcal{P}_{\mathrm{p}}$ should be able to compute saturations for it. And indeed $\mathcal{P}_{\mathrm{p}}$ can compute saturations (infinitely many different ones, as for the constraint in Fig. 4.7 (a)), but only because (P.distr.eq) guarantees that progress is made in the computation.



$U_1 \perp X_1 \wedge U_2 \perp X_1 \wedge U_1 \neq U_2 \wedge$
$A \sim B \wedge B \sim C \wedge C \sim A \wedge$
$\mathrm{co}(A,B)(U_1){=}U_1' \wedge \mathrm{co}(A,B)(U_2){=}U_2' \wedge$
$\mathrm{co}(B,C)(U_1'){=}U_1'' \wedge \mathrm{co}(B,C)(U_2'){=}U_2''$
where $A = X_0/X_1$ and $B = Y_0/Y_1$ and $C = Z_0/Z_1$.

Figure 4.9: A constraint illustrating the path parallelism transitivity rules

Example 4.9 (Path parallelism transitivity). The path parallelism rules see to it that corresponding variables are assigned *in a consistent way* over the different syntactic correspondence functions. We now discuss an example which without the horizontal transitivity rule (P.trans.h) could receive spurious saturations. Consider Fig. 4.9. There are three parallelism literals. In any model, their three syntactic correspondence functions must match the actual (semantic) correspondence functions (because their path literals

must be satisfied and because of the correlation between path parallelism and correspondence functions stated in Prop. 4.3, p. 78). If for example U_1 denotes the first child of the node interpreting X_0, then U'_1 must denote the first child of the node interpreting Y_0, and analogously U_2 and U'_2, according to the definition of correspondence functions (Def. 2.3).

The constraint in Fig. 4.9 has already been partially saturated, as the correspondence formulas show. Figure 4.10 shows a possible further computation. Line (3) and (4) attach the correspondents (with respect to $A \sim C$) of U_1, U_2 in C "the wrong way round": At that point the constraint states that U_1, U'_1 denote corresponding nodes, and U'_1, U''_1 denote corresponding nodes, and U_1, U''_2 denote corresponding nodes. That constraint is unsatisfiable. This can be detected, as the rest of the computation in Fig. 4.10 shows, but only using (P.trans.h).

(1)	$\exists \overline{U_1}.\mathrm{co}(A,C)(U_1)=\overline{U_1}$	(P.new)
(2)	$\exists \overline{U_2}.\mathrm{co}(A,C)(U_2)=\overline{U_2}$	(P.new)
(3)	$\overline{U_1}=U''_2$	(P.distr.eq)
(4)	$\overline{U_2}=U''_1$	(P.distr.eq)
(5)	$\mathrm{p}(\substack{X_0 \ Z_0 \\ U_1 \ U''_1}), \quad \mathrm{p}(\substack{X_0 \ Z_0 \\ U_2 \ U''_2})$	(P.trans.h)
(6)	$\mathrm{p}(\substack{Z_0 \ Z_0 \\ U''_1 \ \overline{U_1}}), \quad \mathrm{p}(\substack{Z_0 \ Z_0 \\ U''_2 \ \overline{U_2}})$	(P.trans.h)
(7)	$U''_1=\overline{U_1}, U''_2=\overline{U_2}$	(P.path.eq.2)
(8)	$\overline{U_1}=\overline{U_2}$	(D.dom.trans)
(9)	$\overline{U_1}\neq\overline{U_2}$	(P.copy.dom)
(10)	`false`	(D.clash.ineq)

Figure 4.10: A further computation on Fig. 4.9

4.1.5 A Note on Saturations and Readings

At the beginning of the current chapter we have argued that one of the good properties of the procedure $\mathcal{P}_\mathrm{p}$ is that, for the linguistically relevant constraints, it can compute constraints that directly match the correct readings. Now it is time to take a closer look at this statement.

How many saturations does the constraint for our standard quantifier parallelism sentence (4.1) have? We repeat the constraint here (without the lambda binding literals, which we cover in the following chapter) as Fig. 4.11. The sentence has three readings. And indeed the constraint in Fig. 4.11 has three different partial saturations sketched in Fig. 4.12. But the constraint has more saturations than three, because of (P.distr.eq): For each of the dominance edges shown in the sketches in Fig. 4.12, (P.distr.eq) guesses whether to identify the upper and the lower variable that the dominance edge connects.

But as we have said above, it is difficult to find a constraint for which (P.distr.eq) is

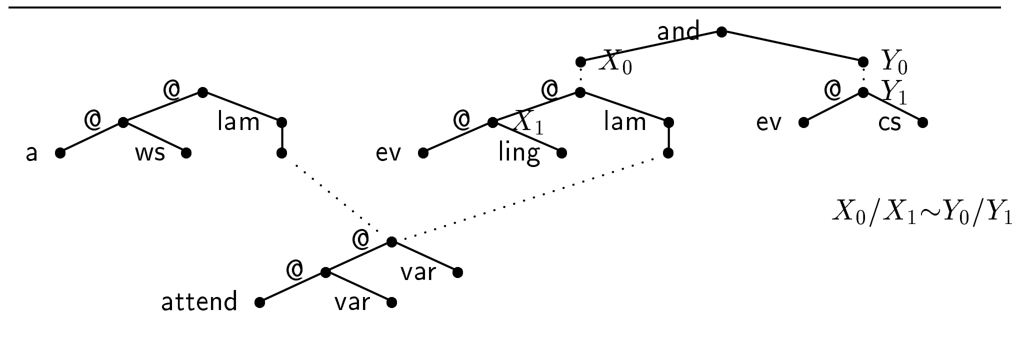


Figure 4.11: Constraint for *Every linguist attended a workshop. Every computer scientist did, too.* (minus lambda binding)

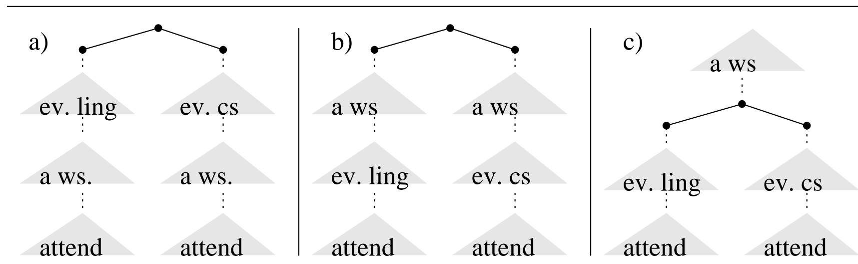


Figure 4.12: Sketch of three partial saturations of the constraint in Fig. 4.11

actually needed. The one constraint that we have discussed above involves multiple self-overlap of parallelism literals. However, in linguistically relevant constraints the parallelism literals are much simpler – it seems that "self-overlap" does not occur at all. In a first implementation of the procedure $\mathcal{P}_\mathrm{p}$ [21] the rule (P.distr.eq) is disabled, which means that the saturations computed e.g. for the constraint in Fig. 4.11 are actually the three constraints sketched in Fig. 4.12.

In Chapter 9, where we give an outlook over further work, we discuss a restriction of the language of parallelism constraints that excludes "self-overlap": This language fragment is decidable and could have interesting processing properties (and it should suffice for handling parallelism phenomena).

4.2 Some Properties of the Procedure: Soundness, Nontermination, Control, Saturations

In this and the following sections we examine properties of the semi-decision procedure $\mathcal{P}_\mathrm{p}$. All results are collected in a theorem in Sec. 4.6.

4.2.1 Soundness

In Def. 3.3 (p. 62), we have stated the notion of soundness that we use: We call a saturation procedure sound if all its rules are equivalence transformations. We have noted that because we are working in a saturation framework it suffices to show that in all rules the premise entails the conclusion.

It is easy to show that in all rules of the procedure $\mathcal{P}_{\mathrm{p}}$, the left-hand side entails the right-hand side: The two additional distribution rules are obviously sound, and the rules about path literals describe valid properties of path parallelism. Concerning the core rules, Prop. 4.3 (p. 78) implies that satisfaction for correspondence formulas works the same way as satisfaction of a literal: A correspondence formula $\mathrm{co}(A,B)(U){=}V$ is satisfied by a lambda structure iff the correspondence function for the segments interpreting A, B maps the denotation of U to the denotation of V. And that means that the core rules of $\mathcal{P}_{\mathrm{p}}$ are obviously sound by the definition of parallelism (Def. 2.4, p. 28).

Lemma 4.10 (Soundness). *The semi-decision procedure $\mathcal{P}_{\mathrm{p}}$ for $\mathcal{C}_{\mathrm{p}}$ is sound for lambda structures.*

4.2.2 Nontermination

As we have seen in Ex. 4.7, there are unsatisfiable constraints for which $\mathcal{P}_{\mathrm{p}}$ does not terminate. But if a $\mathcal{C}_{\mathrm{p}}$ constraint is satisfiable, then the procedure $\mathcal{P}_{\mathrm{p}}$ will compute all its *minimal saturations*, as we show below in Sec. 4.5.

4.2.3 Fairness

Basically, we will call a sequence of saturation steps *fair* if whenever a rule is applicable, one of the disjuncts in its conclusion will ultimately be added. That is, our notion of fairness is one of exhaustiveness.

Definition 4.11 (Fairness). *Let S be a set of saturation rules and let $\varphi_0, \varphi_1, \ldots$ be clauses. We call a sequence $\varphi_0 \rightarrow_S \varphi_1 \rightarrow_S \ldots$* fair *iff either there exists some $i \geq 0$ such that φ_i is failed, or the following holds:*

For all $i \geq 0$ such that some rule $\rho = \varsigma_0 \rightarrow \bigvee_{k=1}^{n} \exists V_k \varsigma_k$ in S is applicable to φ_i, there exists some $j > i$, and some k such that ς'_k is in φ_j for some V_k-variant ς'_k of ς_k.

For the $\mathcal{C}_{\mathrm{d}}$-solver $\mathcal{P}_{\mathrm{d}}$ of the previous chapter, fairness is not a problem. Each sequence of $\mathcal{P}_{\mathrm{d}}$-saturation steps is finite because there are only finitely many different literals that that algorithm can add for each variable. And if we reach a saturated constraint, the above fairness condition must hold or the constraint would not be saturated: By Def. 4.1 (p. 76) a saturation rule is applicable only if it can add something new to the constraint (by the application condition appc_ρ of a rule ρ). So it can happen that two different rules ρ_1, ρ_2 are applicable to a constraint φ, and the rule ρ_1 is chosen to produce the

constraint φ' by a saturation step $\varphi \rightarrow_{\{\rho_1\}} \varphi'$, but ρ_2 is not applicable to φ' anymore. Suppose that is the case, and suppose $\rho_2 = \varphi \rightarrow \bigvee_{i=1}^{n} \exists V_i \varphi_i$ is the rule that did not get chosen. Then a V_i-variant of some φ_i must already be in φ', this is the way that the application condition appc_{ρ_2} is defined. That is, one of the disjuncts in ρ_2's conclusion has been added, and if not by ρ_2, then by some other rule (in the case we have just sketched it must have been ρ_1).

For the procedure $\mathcal{P}_\text{p}$, however, things are different. Because of (P.new), there are infinite sequences of $\mathcal{P}_\text{p}$-saturation steps. So we introduce the following condition to ensure fairness:

> **Fairness condition.** (P.new) is applied only to constraints saturated under $\mathcal{P}_\text{p} - \{(\text{P.new})\}$. (P.new) is applied to variables in the order of their introduction into the constraint.

It is easy to verify that this condition guarantees fairness as defined above. But why this particular condition? With this condition, we make progress after each application of (P.new), in the sense that we have discussed in Ex. 4.8: After each application of (P.new) we determine whether the newly introduced corresponding variable denotes the same node as some variable we already had, or if it denotes a node that up to then did not interpret any variable of the constraint. And it is this fairness condition (or rather its first half) that we will use in our argument for the completeness of the procedure $\mathcal{P}_\text{p}$.

4.2.4 Saturated Constraints

For a satisfiable parallelism constraint, the procedure computes a set of saturations, constraints to which no rule of $\mathcal{P}_\text{p}$ is applicable anymore.

Lemma 4.12. *There are satisfiable parallelism constraints for which $\mathcal{P}_\text{p}$ computes infinitely many saturations.*

Proof. This is the case e.g. for the constraints discussed in Ex. 4.6 and 4.8. □

In Sec. 4.4, where we discuss *minimal saturated constraints*, we introduce a partial order on CLLS_p constraints. There we will show that whenever $\mathcal{P}_\text{p}$ computes more than one saturation for a constraint, the saturations will be incomparable by that order.

$\mathcal{P}_\text{p}$-saturated constraints basically look like the $\mathcal{P}_\text{d}$-saturated constraints of the previous chapter, with a few additional restrictions. Remember that we have informally described $\mathcal{P}_\text{d}$-saturations as follows: The constraint graph of a saturated constraint is a forest with two different kinds of tree nodes, labeled and unlabeled ones, and two different kinds of edges, dominance and labeling edges. A node is either labeled, all its outgoing edges are labeling edges, and its children are ordered; or it is unlabeled, all its outgoing edges are dominance edges, and its children are unordered.

What is different in a $\mathcal{P}_{\mathrm{p}}$-saturated constraint? It may contain parallelism literals. In that case, if we draw the correspondence formulas into the constraint graph in the shape of correspondence arcs, then given a parallelism literal, every constraint graph node "inside" one of its two segment terms will be linked, via a correspondence arc, to exactly one node "inside" the other segment term. Arcs always link labeled with labeled nodes, and unlabeled with unlabeled nodes, except for the holes of a segment term. Arcs linking labeled nodes respect the label and the order of the children. And we can order the "dominance children" of all unlabeled nodes in the graph in such a way that a correspondence arc always links an i-th "dominance child" to an i-th "dominance child".

This last point, that correspondence arcs must in some way also respect the "order" of "dominance children", concerns cases such as the one in Fig. 4.9 (p. 86). In that constraint we have 3 parallelism literals. The constraint is partially saturated, already containing some correspondence formulas. The variables X_0 and Y_0 correspond via the first parallelism literal, Y_0 and Z_0 via the second, Z_0 and X_0 via the third. Each of X_0, Y_0 and Z_0 have 3 "dominance children". In the further saturation of the constraint in Fig. 4.10 we have used (P.trans.h) to ensure that there exists an order on the "dominance children" of X_0, Y_0 and Z_0 that is respected by all correspondence formulas. As the saturation in Fig. 4.10 contains correspondence formulas that violate any possible order on the "dominance children", the result is a clash.

4.3 Satisfiability of Saturated Constraints

In this section we show that from any saturated constraint that $\mathcal{P}_{\mathrm{p}}$ computes we can read off a model. We proceed as in the previous chapter: We first consider simple constraints, for which the constraint graph already looks like a tree. Then we lift the result to arbitrary $\mathcal{P}_{\mathrm{p}}$-saturated constraints.

4.3.1 Valuations and Segment Terms

However, there are two technical issues that we have to address first. The first is an additional piece of notation: we lift valuation functions canonically from variables to segment terms.

Let $A = X_0/X_1, \ldots, X_n$. We write $\sigma(A) = \alpha$ iff the following holds: $\alpha = \pi_0/\pi_1, \ldots, \pi_m$ such that $\sigma(X_0) = \pi_0$, and $\sigma(\{X_1, \ldots, X_n\}) = \{\pi_1, \ldots, \pi_m\}$.

4.3.2 Generatedness

The second issue that we have to consider is this: $\mathcal{P}_{\mathrm{p}}$-saturated constraints are not $\mathcal{C}_{\mathrm{p}}$ constraints but $\mathcal{C}_{\mathrm{pp}}$ constraints – they may contain path literals. The procedure $\mathcal{P}_{\mathrm{p}}$ does not accept path literals in the input, and it does not check whether path literals in *arbitrary* places in a constraint are satisfiable; but it checks the satisfiability of path literals

that it has introduced for recording correspondence. To formalize this, we introduce *generated* constraints, where each path literal either establishes a correspondence for some parallelism literal or is the result of combining several such correspondence statements by a path parallelism rule.

Definition 4.13 (Correspondence-generated). *Let φ be a $\mathcal{C}_{\mathrm{pp}}$-constraint. A path literal $p\binom{U_1\ V_1}{U_2\ V_2} \in \varphi$ is* correspondence-generated *in φ iff there exists some literal $A \sim B \in \varphi$ with $A = U_1/\ldots$ and $B = V_1/\ldots$ such that either $U_2 \in \mathsf{b}(A)$ or $V_2 \in \mathsf{b}(B)$ is in φ.*

Intuitively, a path literal is correspondence-generated if it has been introduced as part of a correspondence formula and thus expresses a correspondence. Now we define what it means for a path literal to be generated: It must be entailed by the non-path literals together with the correspondence-generated path literals.

Definition 4.14 (Generated). *Let φ be a $\mathcal{C}_{\mathrm{pp}}$-constraint, let φ_0 be φ without all its path literals, and let φ_1 be the set of correspondence-generated path literals in φ.*

Then a path literal $p\binom{U_1\ V_1}{U_2\ V_2} \in \varphi$ is generated *in φ iff*

$$\varphi_0 \wedge \varphi_1 \models \ p\binom{U_1\ V_1}{U_2\ V_2} \ .$$

Whenever $\mathcal{P}_{\mathrm{p}}$ computes a saturation of a constraint, that saturation is generated.

Lemma 4.15 (Generatedness). *Let φ be a $\mathcal{C}_{\mathrm{p}}$ constraint with $\varphi \rightarrow^*_{\mathcal{P}_{\mathrm{p}}} \varphi'$. Then φ' is generated.*

Proof. Any path literal in φ' must have been introduced by either (P.init), (P.new), (P.path.eq.1), (P.trans.h), (P.trans.v), (P.diff.1) or (P.diff.2). We proceed by induction on the length of the saturation from φ to φ'. The constraint φ does not contain any path literals, so it is generated by definition. Now suppose $\varphi \rightarrow^*_{\mathcal{P}_{\mathrm{p}}} \varphi'' \rightarrow_{\{\rho\}} \varphi'$, where ρ is an instance of either (P.init), (P.new), (P.path.eq.1), (P.trans.h), (P.trans.v), (P.diff.1), or (P.diff.2), and the inductive hypothesis holds for φ''.

If ρ is an instance of (P.init), then any path literal in $\varphi' - \varphi''$ must be correspondence-generated because (P.init) also infers $U_1 \in \mathsf{b}(A)$ as well as $U_2 \in \mathsf{b}(B)$. If ρ is an instance of (P.new), then any additional path literal in φ' is again correspondence-generated because (P.new) has $U \in \mathsf{b}(A)$ in its premise.

The rules (P.path.eq.1), (P.trans.h), (P.trans.v), (P.diff.1) and (P.diff.2) only infer new path literals from existing ones. They are equivalence transformations by Lemma 4.10. This means that if φ'' is generated, then so is φ'. □

The aim of this section is to show that whenever $\mathcal{P}_{\mathrm{p}}$ computes a saturation, we can construct a model from it. And since we have just shown that anything that $\mathcal{P}_{\mathrm{p}}$ computes from a $\mathcal{C}_{\mathrm{p}}$ constraint is generated, we can safely restrict ourselves to generated constraints for the rest of this section.

4.3.3 Simple Constraints

In Def. 3.7 (p. 64) we have introduced *simple* constraints: they possess a root variable dominating all others, and every variable is labeled. This definition can be lifted canonically from dominance constraints to $\mathcal{C}_{\mathrm{pp}}$ constraints: A $\mathcal{C}_{\mathrm{pp}}$ constraint φ is called simple iff the maximal subset of φ that is a $\mathcal{C}_{\mathrm{d}}$ constraint is simple.

Lemma 4.16 (Satisfiability of simple generated saturations). *A simple generated $\mathcal{P}_{\mathrm{p}}$-saturated $\mathcal{C}_{\mathrm{pp}}$-constraint is satisfiable.*

Proof. Let φ be a simple generated $\mathcal{P}_{\mathrm{p}}$-saturated $\mathcal{C}_{\mathrm{pp}}$-constraint. In Chapter 3 we have shown that any simple $\mathcal{P}_{\mathrm{d}}$-saturated $\mathcal{C}_{\mathrm{d}}$-constraint is satisfiable (Lemma 3.8, p. 65). Now we proceed as follows: We construct a model for the maximal subset of φ that is a dominance constraint, in the same way as in Lemma 3.8, and we show that this model also satisfies the path literals and parallelism literals of φ. So let φ_{dom} be the maximal subset of φ that is a $\mathcal{C}_{\mathrm{d}}$ constraint, and let (θ, σ) be a model for φ_{dom} constructed as in the proof of Lemma 3.8. Note that the model has been constructed in such a way that for any node $\pi \in D_\theta$ there exists some $X \in \mathcal{V}ar(\varphi)$ with $\sigma(X) = \pi$. It remains to show that all path literals and parallelism literals of φ are satisfied in that model.

Path literals. A simple constraint already has a tree-shaped constraint graph. For path literals, we make use of this as follows: Whenever $X_0 \triangleleft^* U$ is in φ – as for example when we have $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix})$ in φ–, there exists a path from $\sigma(X_0)$ to $\sigma(U)$ with the following property: For any node π such that $\sigma(X_0) \triangleleft^* \pi \triangleleft^* \sigma(U)$ holds in θ, there exists a variable $U' \in \mathcal{V}ar(\varphi)$ such that first $\sigma(U') = \pi$, and second, if $\theta \models \pi{:}f(\ldots)$, then $U'{:}f(\ldots)$ is in φ.

We only need to show that all correspondence-generated path literals of φ are satisfied by θ, all others are entailed anyway by the definition of generatedness (Def. 4.14). So let $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix})$ be a correspondence-generated path literal in φ, which by Def. 4.13 means that there exists some parallelism literal $A \sim B \in \varphi$ with $A = X_0/X_1, \ldots, X_n$, $B = Y_0/Y_1, \ldots, Y_n$, and either $U \in \mathsf{b}(A)$ or $V \in \mathsf{b}(B)$ is in φ; but if $U \in \mathsf{b}(A)$ is in φ, then $V \in \mathsf{b}(B)$ is in φ as well by closure under (P.copy.dom).

We proceed by induction on the length of the path from $\sigma(X_0)$ to $\sigma(U)$. If $X_0{=}U$ is in φ, then we must also have $Y_0{=}V$ in φ by saturation under (P.copy.dom). As (θ, σ) satisfies φ_{dom}, σ must map X_0 and U to the same node, and likewise Y_0 and V. So (θ, σ) also satisfies the path literal $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix})$.

Now suppose that the path from $\sigma(X_0)$ to $\sigma(U)$ has length $m+1$. Let $\pi \in D_\theta$ be such that $\sigma(X_0) \triangleleft^* \pi \triangleleft^* \sigma(U)$ holds in θ and the path from $\sigma(X_0)$ to π has length m. Figure 4.13 shows this situation. Then we must have $\theta \models \pi{:}f(\pi 1, \ldots, \pi \ell)$ for some f of arity ℓ, with $\pi i = \sigma(U)$ for some $i \leq \ell$.

As noted above there exists some $U' \in \mathcal{V}ar(\varphi)$ with $\sigma(U') = \pi$ such that $U'{:}f(U_1, \ldots, U_\ell)$

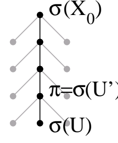


Figure 4.13: Induction step in the model construction for path literals

is in φ for some $U_1, \ldots, U_\ell \in \mathcal{V}ar(\varphi)$. As θ is a model of φ_{dom}, we must have $U_i{=}U$ in φ for the same i for which $\pi i = \sigma(U)$.

Next we show that $U', U_1, \ldots, U_\ell$ must all be inside A. Since $X_0 \triangleleft^* U'$ is in φ, (P.distr.seg) must have been applied to yield either $U' \in \mathsf{b}(A)$ or $X_j \triangleleft^+ U'$ for some j, $1 \leq j \leq n$. Suppose the latter is the case. Then, since φ contains $U' \triangleleft^* U$ by (D.lab.dom) and (D.dom.trans), it also contains $X_j \triangleleft^* U$ by (D.dom.trans). By (P.distr.eq) we must have either $X_j {\neq} U$ or $X_j{=}U$. We regard the first case first. Above we have assumed that φ contains $U \in \mathsf{b}(A)$, so it must contain either $U \triangleleft^* X_j$ and thus $X_j{=}U$, which is impossible by (D.clash.ineq) and the fact that φ is clash-free, or $U \perp X_j$ and thus $U \perp U$ by (D.disj), which is impossible by (D.clash.disj). So only the second case, $X_j{=}U$, remains. In this case φ must contain $U \triangleleft^* U'$ and hence also $U{=}U'{=}U_i$ by (D.dom.trans), but also $U' {\neq} U_i$ by (D.lab.ineq), which is impossible by (D.clash.ineq) and the fact that φ is clash-free. So both cases are impossible, which means that φ must contain $U' \in \mathsf{b}(A)$. Furthermore by closure under (P.distr.eq) φ contains either $U' \in \mathsf{b}^-(A)$ or $U'{=}X_j$ for some $j, 1 \leq j \leq n$. But the latter can be excluded in a similar fashion as we just excluded $X_j \triangleleft^+ U'$, by closure of φ under (D.lab.dom), (D.dom.trans), (P.distr.eq), and (D.clash.ineq).

Now we turn to $U_1, \ldots, U_\ell$. By closure under (D.lab.dom) and (D.dom.trans) we have $X_0 \triangleleft^* U_k$ in φ for $1 \leq k \leq \ell$, so (P.distr.seg) has been applied to U_k and $\mathsf{b}(A)$. If it has not chosen $U_k \in \mathsf{b}(A)$, there must be some $X_j, 1 \leq j \leq n$, with $X_j \triangleleft^+ U_k$ and thus $X_j \triangleleft^* U'$ by (D.distr.notDisj), (D.distr.child), (D.dom.refl), (D.dom.trans), (D.disj) and the fact that φ is clash-free. But that is impossible since φ contains $U' \in \mathsf{b}^-(A)$. Thus, to sum up, $U' \in \mathsf{b}^-(A)$ must be in φ, as well as $U_k \in \mathsf{b}(A)$ for $1 \leq k \leq \ell$.

This means that by closure under (P.new), there must be some $V' \in \mathcal{V}ar(\varphi)$ such that φ contains $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U' & V' \end{smallmatrix})$. As the path from $\sigma(X_0)$ to $\sigma(U')$ has only length m, we can use the inductive hypothesis and conclude that $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U' & V' \end{smallmatrix})$ is satisfied by (θ, σ). Again by (P.new) there must be $V_1, \ldots, V_\ell \in \mathcal{V}ar(\varphi)$ such that $\mathrm{co}(A,B)(U_i){=}V_i$ is in φ for $1 \leq i \leq \ell$. And as $U' \in \mathsf{b}^-(A)$ is in φ, φ contains $V'{:}f(V_1, \ldots, V_\ell)$ by (P.copy.lab).

Above we have said that $U{=}U_i$ is in φ. We also have $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix})$, so $V{=}V_i$ is in φ too by (P.copy.dom) and (P.path.eq.1). So (θ, σ) must satisfy the literal $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix})$: It satisfies $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U' & V' \end{smallmatrix})$, $\sigma(U')$ and $\sigma(V')$ bear the same label, and $\sigma(U)$ is the i-th child of $\sigma(U')$ just as $\sigma(V)$ is the i-th child of $\sigma(V')$.

Parallelism literals. Let $A \sim B \in \varphi$ with $A = X_0/X_1, \ldots, X_n$, $B = Y_0/Y_1, \ldots, Y_n$. Let $\sigma(A) = \alpha$ and $\sigma(B) = \beta$ (as defined in Sec. 4.3.1 above), with $\alpha = \pi_0/\pi_1, \ldots, \pi_m$ and $\beta = \psi_0/\psi_1, \ldots, \psi_m$. We have to show that there exists a correspondence function between $\mathsf{b}(\alpha)$ and $\mathsf{b}(\beta)$. So we define a function $c : \mathsf{b}(\alpha) \to \mathsf{b}(\beta)$ by

$$c(\pi) = \psi \text{ iff } \pi = \sigma(X), \psi = \sigma(Y) \text{ such that } X \in \mathsf{b}(A) \text{ in } \varphi \text{ and } \mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ X & Y \end{smallmatrix}) \text{ in } \varphi.$$

It remains to show that c is the correspondence function for $\alpha \sim \beta$.

***c* is well-defined:** Assume $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ X & Y \end{smallmatrix}), \mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ X' & Y' \end{smallmatrix}) \in \varphi$ with $\sigma(X){=}\sigma(X')$. Then $X{=}X'$ is in φ by the construction of θ in the proof of Lemma 3.8, so by (P.trans.h) and (P.path.eq.1) we have $Y{=}Y'$ in φ.

The domain of *c* is $\mathsf{b}(\alpha)$: We first show that the domain of c is a subset of $\mathsf{b}(\alpha)$. Let $X \in \mathsf{b}(A)$ be in φ. As θ is a model of φ_{dom}, $\pi_0 \triangleleft^* \sigma(X)$ holds in θ, along with either $\sigma(X) \triangleleft^* \pi_i$ or $\sigma(X) \perp \pi_i$ for each $1 \le i \le m$. So $\sigma(X) \in \mathsf{b}(\alpha)$. We now show that $\mathsf{b}(\alpha)$ is a subset of the domain of c. Let $\pi \in \mathsf{b}(\alpha)$, then, as noted above, there exists an X with $\sigma(X) = \pi$. We need to show that $X \in \mathsf{b}(A)$ is in φ. φ possesses a root variable, call it Z, and we have $Z \triangleleft^* X_0, Z \triangleleft^* X$ in φ. Let Z' be a $\triangleleft^+$-maximal variable such that $Z' \triangleleft^* X_0, Z' \triangleleft^* X \in \varphi$. If $Z'{=}X$ is in φ, then $X \triangleleft^* X_0$ is also contained by saturation under (D.dom.trans), and φ must contain $X{=}X_0$ by (P.distr.eq) because $\pi_0 \triangleleft^* \pi$. If $Z'{=}Z'', Z''{:}f(Z_1, \ldots, Z_m)$ are in φ, then we cannot have $Z_i \triangleleft^* X_0, Z_j \triangleleft^* X \in \varphi$ for $1 \le i \ne j \le m$, since then $X \perp X_0 \in \varphi$ by (D.dom.trans) and (D.distr). We cannot have $Z_i \triangleleft^* X_0, Z_i \triangleleft^* X \in \varphi$ for any $i \in \{1, \ldots, m\}$ since we have chosen Z' to be maximal. The only remaining possibility is $Z'{=}X_0$ in φ and $Z_i \triangleleft^* X$ in φ for some $i \in \{1, \ldots, m\}$. In any case, $X_0 \triangleleft^* X$ is in φ. By (P.distr.seg), we must have chosen either $X \triangleleft^* X_i$ or $X \perp X_i$ for all $1 \le i \le n$. By an analogous argument, one can see that the range of c is $\mathsf{b}(\beta)$.

***c* is one-to-one (injective)** because if $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ X & Z \end{smallmatrix}), \mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ Y & Z \end{smallmatrix}) \in \varphi$ for $X, Y \in \mathsf{b}(A)$, then $X{=}Y$ is in φ by (P.copy.dom). **It is onto (surjective)** by (P.new).

$\boldsymbol{c(\pi_i) = \psi_i}$ for $0 \le i \le n$ by (P.init).

***c* is structure-preserving:** Suppose $\pi'_0 \in \mathsf{b}^-(\alpha)$, and $\theta \models \pi'_0{:}f(\pi'_1, \ldots, \pi'_\ell)$. Then there exists a $U_0 \in \mathcal{V}ar(\varphi)$ with $\sigma(U_0) = \pi'_0$ and, as shown above, $U_0 \in \mathsf{b}(A)$ is in φ. As φ is simple, U_0 must be labeled: φ must contain $U_0{=}U'_0, U'_0{:}f(U_1, \ldots, U_\ell)$ for some $U'_0, U_1, \ldots, U_\ell$. (P.distr.seg) and (P.distr.eq) must have chosen $U_0 \in \mathsf{b}^-(A)$ since $\pi'_0 \in \mathsf{b}^-(\alpha)$. Thus $U_i \in \mathsf{b}(A)$ is in φ for $1 \le i \le \ell$. By (P.new), φ contains $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ U_i & V_i \end{smallmatrix})$, $0 \le i \le \ell$, for some $V_0, \ldots, V_\ell$, and by (P.path.eq.1) and (P.copy.lab), it contains $V_0{:}f(V_1, \ldots, V_\ell)$. By the construction of c, we have $c(\pi'_i) = c(\sigma(U_i)) = \sigma(V_i)$ for $0 \le i \le \ell$, so we must have $\theta \models \sigma(V_0){:}f(\sigma(V_1), \ldots, \sigma(V_\ell)) = c(\pi'_0){:}f(c(\pi'_1), \ldots, c(\pi'_\ell))$. The opposite direction, starting from $\theta \models c(\pi'_0){:}f(c(\pi'_1), \ldots, c(\pi'_\ell))$, is proved by an analogous argument. □

4.3.4 Non-simple Constrains

Now suppose we have a $\mathcal{P}_{\mathrm{p}}$-saturated constraint that is not simple. As we have done in the case of dominance constraints (in the previous chapter), we extend a non-simple

saturation by labeling previously unlabeled variables while keeping the constraint saturated, until we finally reach a simple saturated constraint. We reuse the definitions of the partial order $\prec_\varphi$ (Def. 3.9, p. 66), $\mathrm{con}_\varphi(X)$ for a node's minimal dominance children (Def. 3.9, p. 66), and φ-disjointness sets, which we use to determine minimal dominance children that may denote different nodes (Def. 3.10, p. 67).

X_0, X, X_1 Y_0, Y, Y_1

$$X_0/X_1{\sim}Y_0/Y_1 \wedge$$
$$\mathrm{co}(X_0/X_1, Y_0/Y_1)(X){=}Y$$

Figure 4.14: Extension by labeling

However, to keep the constraint saturated during extension, we now have to take parallelism literals into account. Consider Fig. 4.14, where we have $X_0/X_1{\sim}Y_0/Y_1$ and $\mathrm{co}(X_0/X_1, Y_0/Y_1)(X){=}Y$. We have to be careful when labeling X_0: X_0 is in $\mathsf{b}(X_0/X_1)$, and when we add $X_0{:}g(X)$ for some unary g, we also have to add $Y_0{:}g(Y)$, otherwise (P.copy.lab) would be applicable. In general, we have to label all corresponding variables in the same way at the same time. We formalize this in the notion of the *copy set.*

Definition 4.17 (Copy set). *Let φ be a $\mathcal{C}_{\mathrm{pp}}$ constraint. Then the relation $\hookrightarrow_\varphi$ on tuples of variables from $\mathcal{V}ar(\varphi)$ is defined by*

$$(U_0, U_1, \ldots, U_m) \hookrightarrow_\varphi (V_0, V_1, \ldots, V_m)$$

iff there exists segment terms A, B of φ such that $A \sim^{\mathrm{sym}} B$ is in φ and $co(A,B)(U_i){=}V_i$ is in φ for all $0 \leq i \leq m$, and $U_i \in \mathsf{b}(A)$ is in φ for $1 \leq i \leq m$, and $U_0 \in \mathsf{b}^-(A)$ is in φ.

Furthermore, we define copy sets *of variable tuples by*

$$\mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m) =_{\mathrm{def}} \{(V_0, V_1, \ldots, V_m) \mid (U_0, U_1, \ldots, U_m) \hookrightarrow^*_\varphi (V_0, V_1, \ldots, V_m)\}$$

where as usual $\hookrightarrow^*_\varphi$ is the reflexive and transitive closure of $\hookrightarrow_\varphi$.

Note that the relation $\hookrightarrow_\varphi$ is symmetric because $A \sim^{\mathrm{sym}} B$ is symmetric. Members of the same copy set share some properties:

Lemma 4.18. *Let φ be a $\mathcal{P}_{\mathrm{p}}$-saturated $\mathcal{C}_{\mathrm{pp}}$-constraint with $U_0, \ldots, U_m \in \mathcal{V}ar(\varphi)$, and let $(V_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$.*

- *If U_0 is unlabeled in φ, then so is V_0.*
- *If $\{U_1, \ldots, U_m\} \subseteq con_\varphi(U_0)$, then $\{V_1, \ldots, V_m\} \subseteq con_\varphi(V_0)$.*

- *If* $\{U_1, \ldots, U_m\}$ *is a maximal* φ*-disjointness set in* $con_\varphi(U_0)$*, then* $\{V_1, \ldots, V_m\}$ *is a maximal* φ*-disjointness set in* $con_\varphi(V_0)$*.*

Proof. By well-founded induction on the strict partial order $\subset$ on $\{\mathcal{S} \mid \{U_0{:}f(U_1, \ldots, U_m)\} \subseteq \mathcal{S} \subseteq \text{copy}_\varphi(U_0, U_1, \ldots, U_m)\}$.

The case of $\mathcal{S} = \{(U_0, U_1, \ldots, U_m)\}$ is trivial. Otherwise, $\mathcal{S}$ has the form $\mathcal{S}' \cup \{(V_0, V_1, \ldots, V_m)\}$ and there exists some $(W_0, W_1, \ldots, W_m) \in \mathcal{S}'$ with $(W_0, W_1, \ldots, W_m) \hookrightarrow_\varphi (V_0, V_1, \ldots, V_m)$: we have $(U_0, U_1, \ldots, U_m) \in \mathcal{S}$, so if there were no such $(W_0, W_1, \ldots, W_m) \in \mathcal{S}'$, then $\mathcal{S} \not\subseteq \text{copy}_\varphi(U_0, U_1, \ldots, U_m)$. Let $A \sim^{\text{sym}} B$ be in φ with $A = X_0/X_1, \ldots, X_n$, $B = Y_0/Y_1, \ldots, Y_n$, $W_i \in \mathsf{b}(A)$ in φ, $W_0 \in \mathsf{b}^-(A)$ in φ, and $\text{co}(A, B)(W_i){=}V_i \in \varphi$ for $0 \leq i \leq m$. Then $V_i \in \mathsf{b}(B)$ is in φ for $1 \leq i \leq m$ and $V_0 \in \mathsf{b}^-(B)$ is in φ by closure of φ under (P.copy.dom).

- Suppose W_0 is unlabeled. Then V_0 must be unlabeled too, as any labeling literal would have been copied by (P.copy.lab).
- Suppose $\{W_1, \ldots, W_m\} \subseteq \text{con}_\varphi(W_0)$. Then by closure under (P.copy.dom), $V_0 \triangleleft^* V_i \in \varphi$ but $V_i \triangleleft^* V_0 \notin \varphi$ for $1 \leq i \leq m$. Assume that V_i is not minimal with $V_0 \prec_\varphi V_i$, i.e. there exists some Z with $V_0 \prec_\varphi Z \prec_\varphi V_i$. Then $Z \in \mathsf{b}(B)$ is in φ by closure under (D.dom.trans), (D.prop.disj), (P.distr.seg). So by (P.new) there exists some Z' with $Z' \in \mathsf{b}(A)$ in φ as well as $\text{co}(A, B)(Z'){=}Z$. But then $W_0 \triangleleft^* Z', Z' \triangleleft^* W_i \in \varphi$ by (P.copy.dom), but neither $Z' \triangleleft^* W_0$ nor $W_i \triangleleft^* Z'$ is in φ, so W_i is not minimal either, a contradiction.
- Suppose $\{W_1, \ldots, W_m\}$ is a maximal φ-disjointness set in $\text{con}_\varphi(W_0)$. Assume that $\{V_i, V_j\}$ is not a disjointness set for some $1 \leq i < j \leq n$. So either $V_i \triangleleft^* V_j$ or $V_j \triangleleft^* V_i$ is in φ. But then by (P.copy.dom), W_i and W_j do not form a disjointness set either, a contradiction.

  Assume $\{V_1, \ldots, V_m\}$ is not maximal, i.e. there exists some $V' \notin \{V_1, \ldots, V_m\}$ such that $\{V_1, \ldots, V_m, V'\} \subseteq \text{con}_\varphi(V_0)$ is a disjointness set. We must have $Y_0 \triangleleft^* V'$ by (D.dom.trans), and by (P.distr.seg) either $V' \triangleleft^* Y_i$ or $V' \perp Y_i$ or $Y_i \triangleleft^+ V'$ for each $1 \leq i \leq n$. But if $Y_i \triangleleft^+ V'$ for some i, then $V' \notin \text{con}_\varphi(V_0)$ because $V_0 \in \mathsf{b}^-(B)$ is in φ. So $V' \in \mathsf{b}(B)$ is contained as well. By closure under (P.new) and (P.copy.dom), there exists a W' with $W' \in \mathsf{b}(A)$ in φ as well as $\text{co}(A, B)(W'){=}V'$, and $W' \in \text{con}_\varphi(W_0)$. W' cannot be in $\{W_1, \ldots, W_m\}$: If $W'{=}W_i$ is in φ for some $i \in \{1, \ldots, m\}$, then $\text{p}(\begin{smallmatrix} X_0 & Y_0 \\ W_i & V' \end{smallmatrix}), \text{p}(\begin{smallmatrix} X_0 & Y_0 \\ W_i & V_i \end{smallmatrix})$ is in φ by (P.path.eq.1), so $V'{=}V_i$ is in φ by (P.path.eq.2). Hence, $\{W_1, \ldots, W_m, W'\}$ is a φ-disjointness set in $\text{con}_\varphi(W_0)$ that is bigger than $\{W_1, \ldots, W_m\}$, a contradiction.

□

Now we proceed like in Lemma 3.12 (p. 67) of the previous chapter: we extend a saturated non-simple constraint by labeling at least one previously unlabeled variable.

Lemma 4.19 (Extension by labeling). *Every $\mathcal{P}_\mathrm{p}$-saturated $\mathcal{C}_\mathrm{pp}$-constraint with an unlabeled variable U_0 can be extended to a $\mathcal{P}_\mathrm{p}$-saturated constraint in which U_0 is labeled.*

Proof. Let $\{U_1, \ldots, U_m\}$ be a maximal φ-disjointness set in $\mathrm{con}_\varphi(U_0)$. Assume that Σ contains a function symbol f of arity m. (If it does not, then we can encode it using a nullary function symbol and a symbol of arity 2, as in Lemma 3.12.) We define the following extension $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ of $\varphi \wedge U_0{:}f(U_1, \ldots, U_m)$:

$$\mathsf{ext}_{U_0,\ldots,U_m}(\varphi) =_{\mathrm{def}} \varphi \;\wedge \bigwedge_{\substack{(V_0,V_1,\ldots,V_m)\in \\ \mathrm{copy}_\varphi\left(U_0,U_1,\ldots,U_m\right)}} \Big(V_0{:}f(V_1,\ldots,V_m) \;\wedge\; \textstyle\bigwedge_{i=1}^m V_0{\neq}V_i \;\wedge\; \displaystyle\bigwedge_{\substack{V_i \lhd^* Z, V_j \lhd^* W \in \varphi, \\ 1\leq i<j\leq n}} Z{\perp}W \;\wedge\; \bigwedge_{\substack{Z:g(\ldots)\in\varphi, \\ g\neq f}} Z{\neq}V_0 \Big)$$

This definition of extensions is the same as in Lemma 3.12, except that all members of a copy set are labeled at the same time. We now show that no $\mathcal{P}_\mathrm{p}$-rule is applicable to $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$. For better readability, we abbreviate $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ by $\mathsf{ext}(\varphi)$.

We first consider the most interesting rules:

(D.lab.decom): This rule has the form $X{:}f(X_1, \ldots, X_n) \wedge Y{:}f(Y_1, \ldots, Y_n) \wedge X{=}Y \to \bigwedge_{i=1}^n X_i{=}Y_i$. If this rule has become applicable in $\mathsf{ext}(\varphi)$, then it must concern a newly labeled variable V_0 (since we have not added any dominance literals): So suppose $(V_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi\big(U_0, U_1, \ldots, U_m\big)$, and $V_0 = W_0$ is in φ. Then V_0 must be unlabeled in φ by Lemma 4.18, so W_0 must be unlabeled in φ too. Hence, for (D.lab.decom) to be applicable, both $V_0{:}f(V_1, \ldots, V_m)$ and W_0: $f(W_1, \ldots, W_m)$ must be in $\mathsf{ext}(\varphi) - \varphi$, which means that $(W_0, W_1, \ldots, W_m)$ must be in $\mathrm{copy}_\varphi\big(U_0, U_1, \ldots, U_m\big)$ too.

If $\mathrm{copy}_\varphi\big(U_0, U_1, \ldots, U_m\big)$ is a singleton, then we must have $U_i{=}V_i{=}W_i$ for $1 \leq i \leq m$. So suppose otherwise. We show the following auxiliary lemma:

Lemma 4.20. *Let $(Z_0, Z_1, \ldots, Z_m) \in \mathrm{copy}_\varphi\big(U_0, U_1, \ldots, U_m\big)$. Then $p(\begin{smallmatrix}U_0 & Z_0\\ U_i & Z_i\end{smallmatrix}) \in \varphi$ for $1 \leq i \leq m$.*

Proof. We use induction on the length of a $\hookrightarrow_\varphi$ sequence starting in $(U_0, U_1, \ldots, U_m)$ and ending in $(Z_0, Z_1, \ldots, Z_m)$. We start with a sequence of length zero, i.e. we show that $\mathrm{p}(\begin{smallmatrix}U_0 & U_0\\ U_i & U_i\end{smallmatrix})$ is in φ for $1 \leq i \leq m$.

$\mathrm{copy}_\varphi\big(U_0, U_1, \ldots, U_m\big)$ is not a singleton

$\Longrightarrow$ there exists some $A \sim^{\mathrm{sym}} B$ in φ, $A = X_0/X_1, \ldots, X_n$, $B = Y_0/Y_1, \ldots, Y_n$, with $U_i \in \mathsf{b}(A)$ in φ for $0 \leq i \leq m$

$\Longrightarrow$ by (P.new), there exist $U_0', \ldots, U_m'$ such that $\mathrm{co}(A, B)(U_i){=}U_i'$ in φ for $0 \leq i \leq m$

$\Longrightarrow$ by (P.trans.h), $\mathrm{p}(\begin{smallmatrix}X_0 & X_0\\ U_i & U_i\end{smallmatrix})$ in φ for $0 \leq i \leq m$

$\Longrightarrow$ by (P.diff.1) and the fact that $U_0 \lhd^* U_i \in \varphi$, $\mathrm{p}(\begin{smallmatrix}U_0 & U_0\\ U_i & U_i\end{smallmatrix}) \in \varphi$ for $1 \leq i \leq m$

Now suppose $(Z'_0, Z'_1, \ldots, Z'_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$ with $\mathrm{p}(\begin{smallmatrix}U_0 & Z'_0\\ U_i & Z'_i\end{smallmatrix}) \in \varphi$ for $1 \leq i \leq m$, and $(Z'_0, Z'_1, \ldots, Z'_m) \hookrightarrow_\varphi (Z_0, Z_1, \ldots, Z_m)$. Then φ contains some $A \sim^{\mathrm{sym}} B$ defined as above with $Z'_i \in \mathsf{b}(A)$ $\mathrm{p}(\begin{smallmatrix}X_0 & Y_0\\ Z'_i & Z_i\end{smallmatrix})$ in φ for $0 \leq i \leq m$. Then by closure under (P.diff.1), $\mathrm{p}(\begin{smallmatrix}Z'_0 & Z_0\\ Z'_i & Z_i\end{smallmatrix})$ is in φ for $1 \leq i \leq m$, and so, by (P.trans.h), is $\mathrm{p}(\begin{smallmatrix}U_0 & Z_0\\ U_i & Z_i\end{smallmatrix})$. □

This concludes the proof of the auxiliary lemma. By that lemma, $\mathrm{p}(\begin{smallmatrix}U_0 & V_0\\ U_i & V_i\end{smallmatrix})$, $\mathrm{p}(\begin{smallmatrix}U_0 & W_0\\ U_i & W_i\end{smallmatrix}) \in \varphi$ for $1 \leq i \leq m$. By closure under (P.trans.h), φ contains $\mathrm{p}(\begin{smallmatrix}V_0 & W_0\\ V_i & W_i\end{smallmatrix})$, and as $V_0{=}W_0$ is in φ, $\mathrm{p}(\begin{smallmatrix}W_0 & W_0\\ V_i & W_i\end{smallmatrix})$ in φ by (P.path.eq.1), whence by (P.path.eq.2), $V_i{=}W_i$ in φ already (all for $1 \leq i \leq m$).

(P.copy.dom): This rule has the form $U_1 R U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \rightarrow V_1 R V_2$ for $R \in \{\triangleleft^*, \perp, \neq\}$. We consider all possible cases of R. Any dominance literal in $\mathsf{ext}(\varphi)$ is in φ already, so the case of R being $\triangleleft^*$ does not apply.

Now suppose R is $\perp$. Let $Z \perp W$ be in $\mathsf{ext}(\varphi) - \varphi$, where $V_{i_1} \triangleleft^* Z, V_{i_2} \triangleleft^* W$ in φ for some $(V_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$ and some $1 \leq i_1 < i_2 \leq m$. (Thus, $\{V_1, \ldots, V_m\} \neq \emptyset$.) Suppose φ contains $A \sim^{\mathrm{sym}} B$, with A, B defined as usual, with $Z \in \mathsf{b}(A)$, $W \in \mathsf{b}(A)$ in φ.

$\Longrightarrow$ by (P.new), there exist Z', W' such that $\mathrm{p}(\begin{smallmatrix}X_0 & Y_0\\ Z & Z'\end{smallmatrix}), \mathrm{p}(\begin{smallmatrix}X_0 & Y_0\\ W & W'\end{smallmatrix}) \in \varphi$
$\Longrightarrow$ $X_0 \triangleleft^* Z, X_0 \triangleleft^* W \in \varphi$ and by (D.dom.trans), $V_0 \triangleleft^* Z, V_0 \triangleleft^* W \in \varphi$
$\Longrightarrow$ by (D.distr.notDisj), φ contains either $V_0 \triangleleft^* X_0$ or $X_0 \triangleleft^* V_0$

Suppose φ contains $V_0 \triangleleft^* X_0$ but not $V_0{=}X_0$, i.e. $V_0 \prec_\varphi X_0$.

Suppose $X_0 \in \mathrm{con}_\varphi(V_0)$.
$\Longrightarrow$ $X_0{=}V_k$ is in φ for some $k \in \{1, \ldots, m\}$ since $\{V_1, \ldots, V_m\}$ is a maximal φ-disjointness set in $\mathrm{con}_\varphi(V_0)$ by Lemma 4.18.

Suppose $X_0 \notin \mathrm{con}_\varphi(V_0)$.
$\Longrightarrow$ there exists some $V' \in \mathrm{con}_\varphi(V_0)$ such that $V' \prec_\varphi X_0$
$\Longrightarrow$ by Lemma 3.11, φ contains $V'{=}V_k$ for some $k \in \{1, \ldots, m\}$
$\Longrightarrow$ by (D.dom.trans), $V_k \triangleleft^* X_0 \in \varphi$

but at least one of $V_{i_1} \perp V_k$ and $V_{i_2} \perp V_k$ is in φ, and φ is clash-free
$\Longrightarrow$ we cannot have both $X_0 \triangleleft^* U_0$ and $X_0 \triangleleft^* V_0$ in φ
$\Longrightarrow$ (D.distr.notDisj) must have made the choice $X_0 \triangleleft^* V_0$

Now suppose $X_0 \triangleleft^* V_0$ is in φ.

We have $V_0 \triangleleft^+ Z, V_0 \triangleleft^+ W$ in φ by (D.dom.trans), (D.lab.dom), (P.distr.eq)
$\Longrightarrow$ either $V_0 \triangleleft^+ X_i$ in φ or $V_0 \perp X_i$ in φ for each $1 \leq i \leq n$ by (P.distr.seg) and (P.distr.eq) since $Z \in \mathsf{b}(A)$, $W \in \mathsf{b}(A)$ in φ
$\Longrightarrow$ $V_i \in \mathsf{b}(A)$ in φ for $0 \leq i \leq m$ by (P.distr.eq), (D.disj) and the fact that all V_i are minimal with $V_0 \prec_\varphi V_i$
$\Longrightarrow$ by (P.new), there are $V'_0, \ldots, V'_m$ such that $\mathrm{p}(\begin{smallmatrix}X_0 & Y_0\\ V_i & V'_i\end{smallmatrix}) \in \varphi$ for $0 \leq i \leq m$
$\Longrightarrow$ $(V'_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$ since $V_0{=}X_i$ is not in φ for any $1 \leq i \leq n$
$\Longrightarrow$ by (P.copy.dom), $V'_{i_1} \triangleleft^* Z', V'_{i_2} \triangleleft^* W'$ in φ

$\Longrightarrow$ $Z' \perp W' \in \mathsf{ext}(\varphi)$ by definition

Now suppose R is $\neq$. Let $(V_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$. Suppose $V_0 \neq V_k \in \mathsf{ext}(\varphi) - \varphi$ for some $k \in \{1, \ldots, m\}$. (Again, $\{V_1, \ldots, V_m\} \neq \emptyset$.) Suppose further that $A \sim^{\mathrm{sym}} B$ is in φ with A, B defined as usual, and $V_0 \in \mathsf{b}(A)$, $V_k \in \mathsf{b}(A)$ in φ.

$\Longrightarrow$ by (P.new), there exist V_0', V_k' such that $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ V_0 & V_0' \end{smallmatrix}), \mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ V_k & V_k' \end{smallmatrix}) \in \varphi$
by (P.distr.seg), (P.distr.eq) and the fact that $V_k \in \mathsf{b}(A)$ is in φ, we must have $V_0 \lhd^+ X_i$ in φ or $V_0 \perp X_i$ in φ for $1 \leq i \leq n$

$\Longrightarrow$ by (P.distr.seg), $V_i \in \mathsf{b}(A)$ in φ for $1 \leq i \leq m$
$X_i \lhd^+ V_j$ cannot have been chosen for any $1 \leq i \leq n$, $1 \leq j \leq m$ because $V_0 \in \mathsf{b}^-(A)$ in φ and each V_j is minimal with $V_0 \prec_\varphi V_j$

$\Longrightarrow$ there are $V_1', \ldots, V_m'$ such that $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ V_j & V_j' \end{smallmatrix}) \in \varphi$ for $1 \leq j \leq m$

$\Longrightarrow$ $(V_0', V_1', \ldots, V_m') \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$ since $V_0 \in \mathsf{b}^-(A)$ is in φ

$\Longrightarrow$ $V_0' \neq V_k'$ is in $\mathsf{ext}(\varphi)$ by definition

Now suppose $Z \neq V_0 \in \mathsf{ext}(\varphi) - \varphi$, where $Z{:}g(\ldots)$ is in φ for some g with either $g \neq f$ or $\mathsf{ar}(g) \neq \mathsf{ar}(f)$. Suppose further that $A \sim^{\mathrm{sym}} B$ is in φ, with A, B defined as usual, with $V_0 \in \mathsf{b}(A)$, $Z \in \mathsf{b}(A)$ in φ. By closure under (P.distr.eq), we have either $Z = V_0$ in φ or $Z \neq V_0$ in φ. $Z = V_0$ in φ is impossible since V_0 is unlabeled by Lemma 4.18. So $Z \neq V_0$ must be in φ already.

(P.copy.lab): This rule has the form $U_0{:}f(U_1, \ldots, U_m) \wedge \bigwedge_{i=0}^m \mathrm{co}(A, B)(U_i) = V_i \wedge U_0 \in \mathsf{b}^-(A) \rightarrow V_0{:}f(V_1, \ldots, V_m)$. Let $(V_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$ with $V_0{:}f(V_1, \ldots, V_m) \in \mathsf{ext}(\varphi) - \varphi$. Suppose $A \sim^{\mathrm{sym}} B$ is in φ, with A, B defined as usual, with $V_i \in \mathsf{b}(A)$ in φ for $0 \leq i \leq m$. Then there exist $V_0', \ldots, V_m'$ such that $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ V_i & V_i' \end{smallmatrix}) \in \varphi$ for $0 \leq i \leq m$.

By closure under (P.distr.seg), either $V_0 \neq X_i$ is in φ or $V_0 = X_i$ is in φ for all $1 \leq i \leq n$. If $V_0 \neq X_i$ is in φ for all i, then $(V_0', V_1', \ldots, V_m') \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$, so the labeling literal $V_0'{:}f(V_1', \ldots V_m')$ has been added to $\mathsf{ext}(\varphi)$. If $V_0 = X_i$ is in φ for some i, then (P.copy.lab) is not applicable since it does not copy the label of the exception.

(P.new): This rule has the form $A \sim^{\mathrm{sym}} B \wedge U \in \mathsf{b}(A) \rightarrow \exists U'.\mathrm{co}(A, B)(U) = U'$, where U' is a fresh variable. We have not added any parallelism or dominance literals to the constraint, so the only possibility is that a correspondence formula is new in $\mathsf{ext}(\varphi)$ by the new inequality and disjointness literals. So suppose $A \sim^{\mathrm{sym}} B$ and $X_0 \lhd^* V$ are in φ and $V \in \mathsf{b}(A)$ is in $\mathsf{ext}(\varphi) - \varphi$. But then by closure under (P.distr.seg), one of $V \lhd^* X_i$, $V \perp X_i$, $X_i \lhd^+ V$ must already be in φ for each i.

For the rules (D.clash.ineq), (D.clash.disj), (D.lab.dom), (D.distr.child), we just lift the proofs from Lemma 3.12 using Lemma 4.18, which transfers all the necessary properties of $(U_0, U_1, \ldots, U_m)$ to any $(V_0, V_1, \ldots, V_m) \in \mathrm{copy}_\varphi(U_0, U_1, \ldots, U_m)$:

(D.clash.ineq): $\mathsf{ext}(\varphi)$ contains no new dominance literals. If a new inequality literal $V_0{\neq}V_i$ were to make (D.Clash.Ineq) applicable, then φ must contain $V_0{=}Y_i$, but $V_0{:}f(V_1,\ldots,V_m) \in \mathrm{copy}_\varphi\big(U_0,U_1,\ldots,U_m\big)$, so $V_i \in \mathrm{con}_\varphi(V_0)$ by Lemma 4.18.

If a new inequality $Z{\neq}V_0$ were to make the clash rule applicable, then $Z{:}g(\ldots)$ and $V_0{=}Z$ must be in φ, but by Lemma 4.18, V_0 is unlabeled because U_0 is.

(D.clash.disj): The only new disjointness literals in $\mathsf{ext}(\varphi)$ have the form $Z{\perp}W$ for $V_i{\triangleleft^*}Z, V_j{\triangleleft^*}W$ in φ with $i \neq j$. Assume Z and W are the same variable. Then by (D.distr.notDisj), either $V_i{\triangleleft^*}V_j$ or $V_j{\triangleleft^*}V_i$ must be in φ. But $\{U_i, U_j\}$ is a disjointness set, and so, by Lemma 4.18, is $\{V_i, V_j\}$.

(D.lab.dom): Suppose $V_0{:}f(V_1,\ldots,V_m) \in \mathsf{ext}(\varphi) - \varphi$. We have $V_0{\triangleleft^*}V_i \in \varphi$ by Lemma 4.18. $V_0{\neq}V_i \in \mathsf{ext}(\varphi)$ by definition.

(D.distr.child): Suppose $V_0{:}f(V_1,\ldots,V_m) \in \mathsf{ext}(\varphi) - \varphi$ and $V_0{\triangleleft^*}Z \in \varphi$.

If $Z{\triangleleft^*}V_0 \in \varphi$, then (D.distr.child) is not applicable in $\mathsf{ext}(\varphi)$. Otherwise $V_0 \prec_\varphi Z$. If Z is minimal with $V_0 \prec_\varphi Z$, then $Z \in \mathrm{con}_\varphi(V_0)$, and as $\{V_1,\ldots,V_m\}$ is a maximal φ-disjointness set in $\mathrm{con}_\varphi(V)_0$, we have $Z{=}V_i$ in φ for some $i \in \{1,\ldots,m\}$. If Z is not minimal, there exists some $V' \in \mathrm{con}_\varphi(V_0)$ such that $V'{\triangleleft^*}Z$ is in φ. But then again, $V_i{=}V'$ for some $i \in \{1,\ldots,m\}$, so $V_i{\triangleleft^*}Z$.

The rules (P.init), (P.path.dom), (P.path.eq.1), (P.path.eq.1), (P.distr.eq),(P.distr.seg), (P.trans.h), (P.trans.v), (P.diff.1), (P.diff.2) cannot become applicable because no new variables or new dominance, parallelism, or path literals have been added.

For the proofs concerning (D.dom.refl), (D.dom.trans), (D.lab.ineq), (D.lab.disj), (D.prop.disj), (D.distr.notDisj), the change from labeling single variables to labeling copy sets of variables does not make any difference.

□

By extending a non-simple constraint sufficiently often, we can finally obtain a simple $\mathcal{P}_\mathrm{p}$-saturated constraint:

Proposition 4.21. *Every generated $\mathcal{P}_\mathrm{p}$-saturated $\mathcal{C}_\mathrm{pp}$-constraint can be extended to a simple generated $\mathcal{P}_\mathrm{p}$-saturated $\mathcal{C}_\mathrm{pp}$-constraint.*

Proof. The proof is the same as for Prop. 3.13 (p. 69). Generatedness is preserved because no further path literals are added during the extension. □

Lemma 4.22 (Satisfiability of generated saturations). *A generated $\mathcal{P}_\mathrm{p}$-saturated $\mathcal{C}_\mathrm{pp}$-constraint is satisfiable.*

Proof. We get this result by combining Lemma 4.16 and Prop. 4.21. □

4.4 A Partial Order on $\mathcal{C}_{\text{p}}$ Constraints

In this and the following section we show that $\mathcal{P}_{\text{p}}$ is complete. In Chapter 3 we have defined completeness as computing all minimal saturated constraints for a given constraint, dependent on a partial order $\preceq$ on constraints (Def. 3.15, p. 69). The partial order we have used for $\mathcal{C}_{\text{d}}$ is subset inclusion. But for $\mathcal{C}_{\text{pp}}$ constraints we would like to have a partial order that generalizes over variables introduced during saturation. Intuitively, we want to consider two constraints equal if their constraint graphs look the same and if they agree on the variables that were already present in the input constraint.[2]

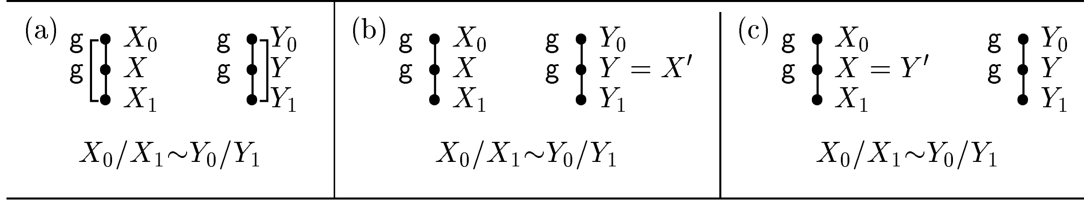


Figure 4.15: Illustrating the problem of existentially quantified variables.

Consider the constraint in Fig. 4.15 (a). If (P.new) is applied to X first, this yields the constraint $\exists X'.\text{co}(X_0/X_1, Y_0/Y_1)(X){=}X'$ for a fresh variable X', plus $Y_0{:}g(X')$ and $X'{=}Y$ by (P.copy.lab) and (D.lab.decom) – the result is shown in picture (b). Accordingly, if (P.new) is applied to Y first, we get $\exists Y'.\text{co}(X_0/X_1, Y_0/Y_1)(Y'){=}Y \wedge X_0{:}g(Y') \wedge Y'{=}X$ for a fresh variable Y' – this constraint is shown in picture (c). The indeterminism in applying (P.new) eventually leads to two $\mathcal{P}_{\text{p}}$-saturated constraints incomparable by $\subseteq$ which, however, we do not want to distinguish.

(1) Eliminating/introducing a variable

$$X{=}Z \;\wedge\; \varphi =^{ex}_{\mathcal{G}} \varphi \quad \text{if } X \notin \mathcal{G},\ X \notin \mathcal{V}ar(\varphi),\ Z \in \mathcal{V}ar(\varphi)$$

(2) Renaming a variable

$$\varphi =^{ex}_{\mathcal{G}} \varphi[Y/X] \quad \text{if } X \notin \mathcal{G},\ Y \notin \mathcal{V}ar(\varphi) \cup \mathcal{G}$$

(3) Exchanging representatives of an equivalence class in a constraint

$$X{=}Y \;\wedge\; \varphi =^{ex}_{\mathcal{G}} X{=}Y \;\wedge\; \varphi[Y/X]$$

(4) Set equivalence (associativity, commutativity, idempotency)

$$\varphi =^{ex}_{\mathcal{G}} \varsigma \quad \text{if } \varphi = \varsigma$$

Figure 4.16: The equivalence relation $=^{ex}_{\mathcal{G}}$ on constraints.

Definition 4.23 ($=^{ex}_{\mathcal{G}}$). *Let $\mathcal{G} \subseteq \mathcal{V}ar$, then $=^{ex}_{\mathcal{G}}$ is the smallest equivalence relation on $\mathcal{C}_{\text{pp}}$ constraints satisfying the axioms in Fig. 4.16.*

[2] The definition of the partial order is more complex here than in the paper by Erk and Niehren [46]. For the previous definition, some proofs required well-foundedness, which the partial order did not possess. For the current definition of the partial order, the proofs do not require well-foundedness.

The idea in distinguishing a set $\mathcal{G} \subseteq \mathcal{V}ar$ of variables is as follows: We use an equivalence relation $=^{ex}_{\mathcal{G}}$ to compare minimal saturated constraints for an input constraint containing only variables from $\mathcal{G}$, while all variables introduced during saturation are from $\mathcal{V}ar$ - $\mathcal{G}$.

Consider again the constraint in Fig. 4.15 and the two different constraints that we have obtained depending on where we applied (P.new). Let $\mathcal{G} = \{X_0, X_1, Y_0, Y_1, X, Y\}$. Then we get $X_0{:}g(X) \wedge Y_0{:}g(Y) \wedge Y_0{:}g(X') \wedge X'{=}Y =^{ex}_{\mathcal{G}} X_0{:}g(X) \wedge Y_0{:}g(Y) \wedge X'{=}Y$ by axioms (3) and (4). This, in turn, is $=^{ex}_{\mathcal{G}}$ equivalent to $X_0{:}g(X) \wedge Y_0{:}g(Y)$ by axiom (1). Again by axiom (1), this is $=^{ex}_{\mathcal{G}}$ equivalent to $X_0{:}g(X) \wedge Y_0{:}g(Y) \wedge Y'{=}X$, which equals $X_0{:}g(X) \wedge X_0{:}g(Y') \wedge Y_0{:}g(Y) \wedge Y'{=}X$ by axioms (4) and (3). So the equivalence relation $=^{ex}_{\mathcal{G}}$ identifies the two constraints that only differ in a (superfluous) additional existentially quantified variable not in $\mathcal{G}$.

In the rest of this section we combine the equivalence relation $=^{ex}_{\mathcal{G}}$ with set inclusion to obtain a partial order on $\mathcal{C}_{\text{pp}}$ constraints, and we show properties of this partial order.

We first define a normal form for $\mathcal{C}_{\text{pp}}$ constraints depending on the set $\mathcal{G}$. A normal form has exactly one variable $\notin \mathcal{G}$ in each $=^{ex}_{\mathcal{G}}$ equivalence class, and the constraint has the form $\varphi \wedge \varphi_{eq}$, where $\mathcal{V}ar(\varphi) \cap \mathcal{G} = \emptyset$, and φ_{eq} is a set of equations that link the variables of each $=^{ex}_{\mathcal{G}}$ equivalence class to their representative.

For a constraint φ and $X \in \mathcal{V}ar(\varphi)$, let $\text{Eq}_\varphi(X)$ be the reflexive and transitive closure of $=$ in φ, i.e. $X \in \text{Eq}_\varphi(X)$, and if $Y \in \text{Eq}_\varphi(X)$ and $Y{=}Z$ in φ, then $Z \in \text{Eq}_\varphi(X)$.

Definition 4.24 ($\mathcal{G}$-normal form). *Let φ be a $\mathcal{C}_{\text{pp}}$ constraint, and let $\mathcal{G} \subseteq \mathcal{V}ar$. Then the $\mathcal{C}_{\text{pp}}$ constraint ς is a* $\mathcal{G}$-normal form *for φ iff the following condition holds: There exists a function $\nu : \mathcal{V}ar(\varphi) \to (\mathcal{V}ar - \mathcal{G})$ such that $\nu(X) = \nu(X')$ iff $X' \in Eq_\varphi(X)$, and with $\mathcal{V}ar(\varphi) = \{X_1, \ldots, X_n\}$, it holds that*

$$\varsigma = \varphi[\nu(X_1)/X_1, \ldots, \nu(X_n)/X_n] \wedge \bigwedge_{X \in \mathcal{V}ar(\varphi) \cap \mathcal{G}} X{=}\nu(X).$$

This normal form has the following properties:

Lemma 4.25 (Properties of $\mathcal{G}$-normal forms). *Let φ be a $\mathcal{C}_{\text{pp}}$ constraint, $\mathcal{G} \subseteq \mathcal{V}ar$, and ς a $\mathcal{G}$-normal form for φ. Then*

1. *$\varsigma =^{ex}_{\mathcal{G}} \varphi$;*
2. *ς is a $\mathcal{G}$-normal form for all $\varphi' =^{ex}_{\mathcal{G}} \varphi$;*
3. *ς is unique modulo α-renaming of the variables in $\mathcal{V}ar(\varsigma) - \mathcal{G}$.*

Proof. 1. Let $\mathcal{V}ar(\varphi) = \{X_1, \ldots, X_n\}$ with $X_1, \ldots, X_k \in \mathcal{G}$ and $X_{k+1}, \ldots, X_n \notin \mathcal{G}$ for some $k \leq n$. Let $\nu : \mathcal{V}ar(\varphi) \to (\mathcal{V}ar - \mathcal{G})$ be as in Def. 4.24. Let $Y_1, \ldots, Y_n \in \mathcal{V}ar$ with $Y_1, \ldots, Y_n \notin \mathcal{G} \cup \mathcal{V}ar(\varphi) \cup \{\nu(X_i) \mid 1 \leq i \leq n\}$. Then

$$\begin{array}{lll} \varphi & =^{ex}_{\mathcal{G}} & X_1{=}Y_1 \wedge \ldots \wedge X_n{=}Y_n \wedge \varphi & \text{by axiom (1)} \\ & =^{ex}_{\mathcal{G}} & X_1{=}Y_1 \wedge \ldots \wedge X_n{=}Y_n \wedge \varphi[Y_1/X_1, \ldots, Y_n/X_n] & \text{by axiom (3)} \\ & =^{ex}_{\mathcal{G}} & X_1{=}Y_1 \wedge \ldots \wedge X_k{=}Y_k \wedge \varphi[Y_1/X_1, \ldots, Y_n/X_n] & \text{by axiom (1)} \\ & =^{ex}_{\mathcal{G}} & Y_1{=}\nu(X_1) \wedge \ldots \wedge Y_n{=}\nu(X_n) \wedge X_1{=}Y_1 \wedge \ldots \wedge X_k{=}Y_k & \\ & & \wedge \varphi[Y_1/X_1, \ldots, Y_n/X_n] & \text{by axiom (1)} \\ & =^{ex}_{\mathcal{G}} & Y_1{=}\nu(X_1) \wedge \ldots \wedge Y_n{=}\nu(X_n) \wedge X_1{=}\nu(X_1) \wedge \ldots \wedge X_k{=}\nu(X_k) & \\ & & \wedge \varphi[\nu(X_1)/X_1, \ldots, \nu(X_n)/X_n] & \text{by axiom (3)} \\ & =^{ex}_{\mathcal{G}} & X_1{=}\nu(X_1) \wedge \ldots \wedge X_k{=}\nu(X_k) \wedge \varphi[\nu(X_1)/X_1, \ldots, \nu(X_n)/X_n] & \text{by axiom (1)} \\ & = & \varsigma & \end{array}$$

The $Y_1, \ldots, Y_n$ are necessary to first name variables apart in case $\mathcal{V}ar(\varphi) - \mathcal{G}$ and the range of ν are not disjoint.

2. It suffices to show that if ς is a $\mathcal{G}$-normal form of φ_1, and $\varphi_2 =^{ex}_{\mathcal{G}} \varphi_1$ by a single axiom from Fig. 4.16, then ς is a $\mathcal{G}$-normal form of φ_2.

   Suppose the axiom used is (1), and $\varphi_1 = X{=}Z \wedge \varphi_2$ for $X \notin \mathcal{G} \cup \mathcal{V}ar(\varphi_2)$, $Z \in \mathcal{V}ar(\varphi_2)$. Then ς is a $\mathcal{G}$-normal form of φ_2 with the function $\nu' : \mathcal{V}ar(\varphi_2) \rightarrow (\mathcal{V}ar - \mathcal{G})$ defined as $\nu' = \nu|_{\mathcal{V}ar(\varphi_2)}$.

   If the axiom used is (1), and $\varphi_2 = X{=}Z \wedge \varphi_1$ for $X \notin \mathcal{G} \cup \mathcal{V}ar(\varphi_1)$, $Z \in \mathcal{V}ar(\varphi_1)$, then we can use the function $\nu' : \mathcal{V}ar(\varphi_2) \rightarrow (\mathcal{V}ar - \mathcal{G})$ defined by $\nu'(Y) = \nu(Y)$ for all $Y \in \mathcal{V}ar(\varphi_1)$ and $\nu'(X) = \nu(Z)$.

   The cases of axioms (2), (3), and (4) are obvious.

3. We define a reduction $\rightarrow_{\mathcal{G}}$ that is convergent modulo α-renaming of variables not in $\mathcal{G}$. This reduction reduces each constraint to its $\mathcal{G}$-normal form. Let $Names \subseteq \mathcal{V}ar$ with $Names \cap \mathcal{G} = \emptyset$. Then we define the relation $\rightarrow_{\mathcal{G}}$ on $\mathcal{C}_{\mathrm{pp}}$ constraints by two rules.

   (a) $\varphi \rightarrow_{\mathcal{G}} name(\mathrm{Eq}_\varphi(U), X) \wedge \varphi$
   if $X \in Names$, $U \in \mathcal{V}ar(\varphi)$, $name(\mathrm{Eq}_\varphi(U), Y) \notin \varphi$ for any $Y \in Names$, $name(S, X) \notin \varphi$ for any $S \subseteq \mathcal{V}ar(\varphi)$, and there exists no $Y \in Names$ with $Y \in \mathrm{Eq}_\varphi(U)$, $\mathrm{Eq}_\varphi(U) = \{U_1, \ldots, U_n\}$ such that $\varphi = \bigwedge_{U' \in \mathcal{G} \cap \mathrm{Eq}_\varphi(U)} U'{=}Y \wedge \varphi[Y/U_1, \ldots, Y/U_n]$

   (b) $name(S, X) \wedge \varphi \rightarrow_{\mathcal{G}} \bigwedge_{U \in S \cap \mathcal{G}} U{=}X \wedge \varphi[X/U_1, \ldots, X/U_n]$
   for $S = \{U_1, \ldots, U_n\}$

   The reduction $\rightarrow_{\mathcal{G}}$ is terminating: Let φ' be the "input constraint". The rule (a) is applicable at most $|\mathrm{Eq}_{\varphi'}|$ times, once to each $\mathrm{Eq}_{\varphi'}$-equivalence class, unless it already has a representative in the sense of Def. 4.24. Likewise, rule (b) is applicable at most $|\mathrm{Eq}_{\varphi'}|$ times, its applicability is bounded by the number of $name(S, U)$-atoms in the constraint. Note that the right-hand side of (b) exactly matches the last condition on the applicability of rule (a), hence rule (a) is inapplicable to an equivalence class to which rule (b) has already been applied.

The reduction $\to_{\mathcal{G}}$ is confluent modulo α-renaming of variables in $Names$: The reduction is terminating, so it suffices to show local confluence. If $\varphi \to_{\mathcal{G}} name(S_1, X_1) \wedge \varphi$ and $\varphi \to_{\mathcal{G}} name(S_2, X_2) \wedge \varphi$ by (a), then either the two resulting constraints are equal modulo α-renaming of $Names$-variables, or both can be reduced in one further application of (a) to $name(S_1, X_1) \wedge name(S_2, X_2) \wedge \varphi$ (in this case, S_1, S_2 are not the same Eq_φ-equivalence class). If $\varphi \to_{\mathcal{G}} \varphi_1$ and $\varphi \to_{\mathcal{G}} \varphi_2$ by rule (b), then these two rule applications must concern different Eq_φ-equivalence classes, hence there exists a constraint φ_3 with $\varphi_1 \to_{\mathcal{G}} \varphi_3$ and $\varphi_2 \to_{\mathcal{G}} \varphi_3$. The same is true if $\varphi \to_{\mathcal{G}} \varphi_1$ by (a) and $\varphi \to_{\mathcal{G}} \varphi_2$ by (b).

If ς is a $\to_{\mathcal{G}}$-normal form for φ, then ς is a $\mathcal{G}$-normal form for φ: Rule (a) sees to it that each equivalence class either already has the form demanded by Def. 4.24 or has a "designated representative". Rule (b) then transforms an equivalence class to the form that Def. 4.24 prescribes for $\mathcal{G}$-normal forms.

□

We combine set inclusion and the equivalence relation $=^{ex}_{\mathcal{G}}$ into a partial order on $\mathcal{C}_{\mathrm{pp}}$ constraints.

Definition 4.26 ($\leq_{\mathcal{G}}$). *Let φ_1, φ_2 be $\mathcal{C}_{\mathrm{pp}}$ constraints and let $\mathcal{G} \subseteq \mathcal{V}ar$. Then*

$$\varphi_1 \leq_{\mathcal{G}} \varphi_2$$

iff there exist $\mathcal{G}$-normal forms ς_i of φ_i, $i = 1, 2$, such that $\varsigma_1 \subseteq \varsigma_2$.

The relation $\leq_{\mathcal{G}}$ is actually a partial order:

Lemma 4.27 (Partial order). *Let $\mathcal{G} \subseteq \mathcal{V}ar$. Then $\leq_{\mathcal{G}}$ is a partial order on $\mathcal{C}_{\mathrm{pp}}$ constraints.*

Proof. Reflexivity is obvious. Now for transitivity: let $\varphi_1 \leq_{\mathcal{G}} \varphi_2 \leq_{\mathcal{G}} \varphi_3$, i.e. there are $\mathcal{G}$-normal forms ς_i for φ_i, $i = 1, 2$, with $\varsigma_1 \subseteq \varsigma_2$, and $\mathcal{G}$-normal forms ς'_i for φ_i, $i = 2, 3$, with $\varsigma'_2 \subseteq \varsigma'_3$. Since the normal forms are unique up to α-renaming of non-$\mathcal{G}$-variables, there exists a renamed normal form ς'_1 of φ_1 with $\varsigma'_1 \subseteq \varsigma'_3$. □

We write $=_{\mathcal{G}}$ for $\leq_{\mathcal{G}} \cap \geq_{\mathcal{G}}$, and $<_{\mathcal{G}}$ for $\leq_{\mathcal{G}} - =_{\mathcal{G}}$.

We will compare $\mathcal{C}_{\mathrm{pp}}$ constraints and also $\mathrm{CLLS_p}$ constraints using the family of partial orders $\leq_{\mathcal{G}}$. We specify the partial order we use by specifying the set $\mathcal{G}$.

Definition 4.28 (Saturation for a constraint with respect to $\mathcal{G}$). *Let φ, ς be $CLLS_{\mathrm{p}}$ constraints and let $\mathcal{G} \subseteq \mathcal{V}ar$. Then ς is a $\mathcal{P}_{\mathrm{p}}$*-saturated constraint for φ with respect to $\mathcal{G}$ *iff ς is a $\mathcal{P}_{\mathrm{p}}$-saturation with $\varphi \leq_{\mathcal{G}} \varsigma$.*

Unfortunately, this partial order is not well-founded on $\mathcal{C}_{\mathrm{pp}}$ constraints in general. While the empty constraint is smaller than all other constraints, there may be an infinite chain $\varphi_1 \leq_{\mathcal{G}} \ldots \leq_{\mathcal{G}} \varphi_2$ for two given constraints φ_1, φ_2. For example, let $\mathcal{G} =_{\mathrm{def}} \{X_1, X_2, Y_1, Y_2\}$. Let $\varphi_1 =_{\mathrm{def}} \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})$ and $\varphi_2 =_{\mathrm{def}} \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge X_1{=}X_2{=}Y_1{=}Y_2$. Then $\varphi_1 <_{\mathcal{G}} \varphi_2$ with

$$
\begin{array}{l}
\mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \\
<_{\mathcal{G}} \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_1' \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1' \\ X_2 & Y_1'' \end{smallmatrix}) \\
<_{\mathcal{G}} \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_1' \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1' \\ X_2 & Y_1'' \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1'' \\ X_2 & Y_1''' \end{smallmatrix}) \\
<_{\mathcal{G}} \ldots \\
<_{\mathcal{G}} \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge X_1{=}X_2{=}Y_1{=}Y_2{=}Y_1'{=}Y_1''{=}\ldots \\
=^{ex}_{\mathcal{G}} \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge X_1{=}X_2{=}Y_1{=}Y_2
\end{array}
$$

The problem is that we can add equalities that may collapse an arbitrary number of $=$ equivalence classes. If we eliminate that possibility, $\leq_{\mathcal{G}}$ is indeed well-founded: we consider constraints in which for each X, Y either $X{=}Y$ or $X{\neq}Y$ is contained.

Definition 4.29 (Projected). *A $\mathcal{C}_{\mathrm{pp}}$ constraint φ is called* projected *iff for all $X, Y, Z \in \mathcal{V}ar(\varphi)$,*

- *either $X{=}Y$ in φ or $X{\neq}Y$ in φ, and*
- *$X{=}Y, X{=}Z$ in $\varphi \Longrightarrow Y{\neq}Z$ not in φ.*

Lemma 4.30 (Well-foundedness of $>_{\mathcal{G}}$ for projected constraints). *Let φ_0 be a projected $\mathcal{C}_{\mathrm{pp}}$ constraint and let $\mathcal{G} \subseteq \mathcal{V}ar$. Then there exists no infinite sequence of projected constraints $\varphi_1, \varphi_2, \ldots$ such that $\varphi_0 >_{\mathcal{G}} \varphi_1 >_{\mathcal{G}} \varphi_2 >_{\mathcal{G}} \ldots$*

Proof. For $\mathcal{C}_{\mathrm{pp}}$ constraints φ_1, φ_2 we have $\varphi_1 >_{\mathcal{G}} \varphi_2$ iff φ_i has a normal form ς_i, $1 = 1, 2$, such that $\varsigma_1 \supset \varsigma_2$ and $\varphi_1 \neq^{ex}_{\mathcal{G}} \varphi_2$. To show that there cannot be an infinite sequence of projected constraints smaller than a given projected constraint φ_0, we embed $(\{\varphi_0, \varphi_1, \ldots\}, >_{\mathcal{G}})$ into $(\mathbb{N}, >)$ by a monotone measure function μ, i.e. for all projected constraints φ' with $\varphi >_{\mathcal{G}} \varphi'$ it holds that $\mu(\varphi) > \mu(\varphi')$.

To embed $(\{\varphi_0, \varphi_1, \ldots\}, >_{\mathcal{G}})$ into $(\mathbb{N}, >)$, we use the following measure function: For any constraint φ, let ς be a $\mathcal{G}$-normal form for φ, then

$$
\begin{array}{lcl}
\mu_1(\varphi) & = & |\mathcal{V}ar(\varsigma) - \mathcal{G}| \\
\mu_2(\varphi) & = & |\varsigma|_{\mathcal{V}ar(\varsigma) - \mathcal{G}}| \\
\mu_3(\varphi) & = & |\mathcal{V}ar(\varsigma) \cap \mathcal{G}| \\
\mu(\varphi) & = & \mu_1(\varphi) + \mu_2(\varphi) + \mu_3(\varphi)
\end{array}
$$

The value $\mu_1(\varphi)$ is the number of $=$ equivalence classes in φ, $\mu_2(\varphi)$ is the number of literals in $\varphi[\nu(X_1)/X_1, \ldots, \nu(X_n)/X_n]$ (where ν is the function that defines ς as in Def.

4.24, and $\mathcal{V}ar(\varphi) = \{X_1, \ldots, X_n\}$), and $\mu_3(\varphi)$ is the number of $\mathcal{G}$-variables in φ. For all $\varphi' =^{ex}_{\mathcal{G}} \varphi$, we have $\mu(\varphi') = \mu(\varphi)$, as only ς, not φ itself, is used to compute $\mu(\varphi)$.

It remains to show that μ is monotone, i.e. it assigns a strictly bigger number to a strictly bigger projected constraint. W.l.o.g. we consider the normal forms themselves, i.e. the case that $\varphi \supset \varphi'$ and $\varphi \neq^{ex}_{\mathcal{G}} \varphi'$. We must have $\mu_3(\varphi) \geq \mu_3(\varphi')$ and $\mu_2(\varphi) \geq \mu_2(\varphi')$ because $\varphi \supset \varphi'$ and the two constraints are both in normal form. Can $\mu_1(\varphi')$ be larger than $\mu_1(\varphi)$, i.e. can φ' possess more variable equivalence classes w.r.t. =? As $\varphi \supset \varphi'$, φ could contain equalities that φ' lacks. But φ and φ' are both projected, so if φ' was lacking some equalities of φ, it would have to contain additional inequalities, which is impossible. □

As all $\mathcal{P}_{\mathrm{p}}$-saturated constraints are projected, we can use $\leq_{\mathcal{G}}$ to compare the saturated constraints that $\mathcal{P}_{\mathrm{p}}$ computes for a given input constraint φ, setting $\mathcal{G} = \mathcal{V}ar(\varphi)$.

Note that there are constraints for which the procedure $\mathcal{P}_{\mathrm{p}}$ computes infinitely many saturated constraints that are incomparable by $\leq_{\mathcal{G}}$. A case in point is the constraint in Fig. 4.7 (p. 85).

Next we show a lemma that will be quite useful in later proofs: we can factor the partial order $\leq_{\mathcal{G}}$ into the relational composition of its components, i.e. $\leq_{\mathcal{G}}$ is $\subseteq \circ =^{ex}_{\mathcal{G}}$.

Lemma 4.31 (Factoring $\leq_{\mathcal{G}}$ into $\subseteq$ and $=^{ex}_{\mathcal{G}}$). *Let φ_1, φ_2 be $\mathcal{C}_{\mathrm{pp}}$ constraints and $\mathcal{G} \subseteq \mathcal{V}ar$. If $\varphi_1 \leq_{\mathcal{G}} \varphi_2$, then there exists a $\mathcal{C}_{\mathrm{pp}}$ constraint φ_2' such that*

$$\varphi_1 \subseteq \varphi_2' =^{ex}_{\mathcal{G}} \varphi_2.$$

Proof. By the definition of $\leq_{\mathcal{G}}$ there exist $\mathcal{G}$-normal forms φ_1' of φ_1 and φ_2' of φ_2 such that $\varphi_1 =^{ex}_{\mathcal{G}} \varphi_1' \subseteq \varphi_2' =^{ex}_{\mathcal{G}} \varphi_2$. In the proof of Lemma 4.25, part 1, we have shown a transformation from a $\mathcal{C}_{\mathrm{pp}}$ constraint to a $\mathcal{G}$-normal form. This transformation only uses a finite number of single axiom applications, as can easily be checked. So there exists a sequence $\varsigma_0, \ldots, \varsigma_n$ of constraints such that $\varphi_1 = \varsigma_0$, $\varsigma_n = \varphi_1'$, and for $0 \leq i \leq n-1$, $\varsigma_i =^{ex}_{\mathcal{G}} \varsigma_{i+1}$ by a single axiom from Fig. 4.16.

Now we transform this sequence $\varsigma_0, \ldots, \varsigma_n$ step by step, moving the $\subseteq$ "to the left beyond the $=^{ex}_{\mathcal{G}}$". We use induction on the length n of the sequence.

If $n = 0$, then φ_1 is in $\mathcal{G}$-normal form and we are done. So suppose $n \geq 1$. We show that there exists a constraint ς such that $\varsigma_0 =^{ex}_{\mathcal{G}} \ldots =^{ex}_{\mathcal{G}} \varsigma_{n-1} \subseteq \varsigma =^{ex}_{\mathcal{G}} \varphi_2'$ holds, i.e. we shift the $\subseteq$ one $=^{ex}_{\mathcal{G}}$ to the left.

Suppose we currently have $\varsigma_0 =^{ex}_{\mathcal{G}} \ldots =^{ex}_{\mathcal{G}} \varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n \subseteq \varsigma'$ for some constraint ς'. We consider all possible ways in which we might have $\varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n$ by a single axiom.

- Suppose $\varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n$ by axiom (1) of Fig. 4.16, and ς_{n-1} has the form $X{=}Z \wedge \varsigma_n$ where $X \notin \mathcal{G} \cup \mathcal{V}ar(\varsigma_n)$ and $Z \in \mathcal{V}ar(\varsigma_n)$.

The constraint ς' has the form $X{=}Z \wedge \varsigma_n \wedge \varsigma''$, where X may occur in ς''. We set $\varsigma = X{=}Z \wedge \varsigma'[X'/X]$ where $X' \notin \mathcal{G}$ does not occur in $\bigcup_{i=1}^{n} \varsigma_i$:

$$\begin{aligned}&\varsigma_{n-1} = X{=}Z \wedge \varsigma_n \subseteq \varsigma = X{=}Z \wedge \varsigma_n \wedge \varsigma''[X'/X]\\ &=^{ex}_{\mathcal{G}} \varsigma'[X'/X] \ =^{ex}_{\mathcal{G}} \ (\varsigma'[X'/X])[X/X'] = \varsigma'.\end{aligned}$$

- Suppose $\varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n$ by axiom (1) of Fig. 4.16, and ς_n has the form $X{=}Z \wedge \varsigma_{n-1}$ where $X \notin \mathcal{G} \cup \mathcal{V}ar(\varsigma_{n-1})$ and $Z \in \mathcal{V}ar(\varsigma_{n-1})$. But then we already have $\varsigma_{n-1} \subseteq \varsigma_n \subseteq \varsigma'$.
- Suppose $\varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n$ by axiom (2) of Fig. 4.16. Then ς_n has the form $\varsigma_{n-1}[Y/X]$ for $X \notin \mathcal{G}$ and $Y \notin \mathcal{V}ar(\varsigma_{n-1}) \cup \mathcal{G}$. This time, we define ς in two steps, making it depend on a constraint ς'' that we define first.

  – ς' has the form $\varsigma_n \wedge \varsigma'''$, where X may only occur in ς'''. So let $\varsigma'' = \varsigma'[X'/X] = \varsigma_n \wedge \varsigma'''[X'/X]$, where $X' \notin \mathcal{G}$ does not occur in $\bigcup_{i=1}^{n} \varsigma_i$.
  – If $Y \in \mathcal{V}ar(\varsigma')$, then it has to be replaced by X while ς' is moved to the left of ς_n. Let $\varsigma = \varsigma''[X/Y]$.

  We have
  $$\varsigma_{n-1} \subseteq \varsigma =^{ex}_{\mathcal{G}} \varsigma[Y/X] =^{ex}_{\mathcal{G}} (\varsigma[Y/X])[X/X'] = \varsigma'.$$

- Suppose $\varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n$ by axiom (3) of Fig. 4.16, and suppose ς_{n-1} has the form $X{=}Y \wedge \varsigma'_{n-1}$, ς_n has the form $X{=}Y \wedge \varsigma'_{n-1}[Y/X]$, and ς' has the form $X{=}Y \wedge \varsigma'_{n-1}[Y/X] \wedge \varsigma''$. We set $\varsigma = X{=}Y \wedge \varsigma'_{n-1} \wedge \varsigma''$. Then
  $$\varsigma_{n-1} \subseteq \varsigma =^{ex}_{\mathcal{G}} X{=}Y \wedge (\varsigma'_{n-1} \wedge \varsigma'')[Y/X] =^{ex}_{\mathcal{G}} \varsigma'.$$
- Suppose $\varsigma_{n-1} =^{ex}_{\mathcal{G}} \varsigma_n$ by axiom (3) of Fig. 4.16, and suppose ς_n has the form $X{=}Y \wedge \varsigma'_n$, while ς_{n-1} has the form $X{=}Y \wedge \varsigma'_n[Y/X]$ and ς' is $X{=}Y \wedge \varsigma'_n \wedge \varsigma''$. We set $\varsigma = X{=}Y \wedge \varsigma'_n[Y/X] \wedge \varsigma''$, then
  $$\varsigma_{n-1} \subseteq \varsigma =^{ex}_{\mathcal{G}} X{=}Y \wedge (\varsigma'_n[Y/X] \wedge \varsigma'')[Y/X] =^{ex}_{\mathcal{G}} \varsigma'.$$

□

We can make the result of the previous lemma even stronger: given a constraint φ and a saturated constraint for it, we can always find another equivalent saturated constraint that is a superset of φ.

Lemma 4.32. *Let φ_1, φ_2 be $\mathcal{C}_{\mathrm{pp}}$ constraints and $\mathcal{G} \subseteq \mathcal{V}ar$ such that $\varphi_1 \leq_{\mathcal{G}} \varphi_2$ and φ_2 is $\mathcal{P}_{\mathrm{p}}$-saturated. Then there exists a $\mathcal{P}_{\mathrm{p}}$-saturated constraint φ'_2 such that $\varphi_1 \subseteq \varphi'_2 =^{ex}_{\mathcal{G}} \varphi_2$.*

Proof. Suppose $\varphi_1 \leq_{\mathcal{G}} \varphi_2$ where φ_2 is $\mathcal{P}_{\mathrm{p}}$-saturated. By Lemma 4.31, there exists a constraint φ_2' with $\varphi_1 \subseteq \varphi_2' =_{\mathcal{G}}^{ex} \varphi_2$. φ_2' need not be $\mathcal{P}_{\mathrm{p}}$-saturated, but we show that a constraint $\mathrm{Pad}(\varphi_2')$, the *padded constraint of* φ_2', is. We proceed as follows: We define the concept of padded constraints, then we show that for any $\mathcal{C}_{\mathrm{pp}}$ constraint φ and its padded version $\mathrm{Pad}(\varphi)$, it holds that $\mathrm{Pad}(\varphi) =_{\mathcal{G}}^{ex} \varphi$. Next, we show that $\mathrm{Pad}(\varphi_2')$ is saturated. Finally we prove that $\varphi_1 \subseteq \mathrm{Pad}(\varphi_2')$.

Let φ be a $\mathcal{C}_{\mathrm{pp}}$ constraint, then

$$\mathrm{Pad}(\varphi) =_{\mathrm{def}} \{\varphi_2'[Z_1/X_1, \ldots, Z_\ell/X_\ell] \mid \begin{array}{l} \varphi_2' \text{ literal in } \varphi, \mathcal{V}ar(\varphi_2') = \{X_1, \ldots, X_\ell\}, \\ Z_i \in \mathrm{Eq}_\varphi(X_i) \text{ for } 1 \leq i \leq \ell\} \end{array}$$

That is, the padded constraint $\mathrm{Pad}(\varphi)$ of a $\mathcal{C}_{\mathrm{pp}}$ constraint φ contains the same literal for all members of an Eq_φ equivalence class.

Now we show that $\mathrm{Pad}(\varphi) =_{\mathcal{G}}^{ex} \varphi$. Let $|\mathrm{Eq}_\varphi| = n$, i.e. there are n equivalence classes of the equivalence relation Eq_φ on $\mathcal{V}ar(\varphi)$. For $1 \leq i \leq n$, let the i-th equivalence class be $\{Z_1^i, \ldots Z_{m_i}^i\}$. Let $Y_1, \ldots, Y_n \notin \mathcal{G} \cup \mathcal{V}ar(\varphi)$. Then

$$\begin{array}{l} \varphi =_{\mathcal{G}}^{ex} Z_1^1{=}Y_1 \ \wedge \ \ldots \ \wedge \ Z_{m_1}^1{=}Y_1 \ \wedge \ \ldots \ \wedge \\ \quad Z_1^n{=}Y_n \ \wedge \ \ldots \ \wedge \ Z_{m_n}^n{=}Y_n \ \wedge \\ \qquad \varphi[Y_1/Z_1^1, \ldots, Y_1/Z_{m_1}^1, \ldots Y_n/Z_1^n, \ldots Y_n/Z_{m_n}^n] \end{array}$$

This holds by axiom (1) for the introduction of the Y_i, $1 \leq i \leq n$, and axiom (3) for replacing Z_j^i by Y_i for $1 \leq j \leq m_i$, $1 \leq i \leq n$. Now by duplicating $\varphi[Y_1/Z_1^1, \ldots, Y_1/Z_{m_1}^1, \ldots Y_n/Z_1^n, \ldots Y_n/Z_{m_n}^n]$ a suitable number of times, using axiom (4), replacing Y_i by each Z_j^i according to axiom (3), and then dropping $Y_1, \ldots, Y_n$ according to axiom (1), we arrive at $\mathrm{Pad}(\varphi)$.

So we know $\mathrm{Pad}(\varphi) =_{\mathcal{G}}^{ex} \varphi$ for any $\mathcal{C}_{\mathrm{pp}}$ constraint φ. Now we show that $\mathrm{Pad}(\varphi_2')$ is saturated. Let $\varphi_2 = \varsigma_0 =_{\mathcal{G}}^{ex} \varsigma_1 =_{\mathcal{G}}^{ex} \ldots =_{\mathcal{G}}^{ex} \varsigma_m = \varphi_2'$ where for all $1 \leq i \leq m-1$, we have $\varsigma_i =_{\mathcal{G}}^{ex} \varsigma_{i+1}$ by a single axiom from Fig. 4.16. As remarked in the previous lemma, this finite sequence $\varsigma_0, \ldots, \varsigma_m$ exists by the proof of Lemma 4.25, part 1. We use induction on m to show that $\mathrm{Pad}(\varsigma_i)$ is $\mathcal{P}_{\mathrm{p}}$-saturated for all $i \leq m$. For $\varsigma_0 = \varphi_2$, this is trivial.

Suppose $\varsigma_i =_{\mathcal{G}}^{ex} \varsigma_{i+1}$ by axiom (1) of Fig. 4.16, and ς_i has the form $X{=}Z \wedge \varsigma_{i+1}$, where $X \notin \mathcal{G} \cup \mathcal{V}ar(\varsigma_{i+1})$ and $Z \in \mathcal{V}ar(\varsigma_{i+1})$. Then X is a superfluous non-$\mathcal{G}$ variable in ς_i, and $\mathrm{Eq}_{\varsigma_i}(X) \ \cap \ \mathcal{V}ar(\varsigma_{i+1}) \neq \emptyset$. So the constraint $\mathrm{Pad}(\varsigma_i)|_{\mathcal{V}ar(\varsigma_i)-\{X\}} = \mathrm{Pad}(\varsigma_{i+1})$ must be saturated, too.

Suppose $\varsigma_i =_{\mathcal{G}}^{ex} \varsigma_{i+1}$ by axiom (1), and ς_{i+1} has the form $X{=}Z \wedge \varsigma_i$ for variables $X \notin \mathcal{G} \cup \mathcal{V}ar(\varsigma_i)$ and $Z \in \mathcal{V}ar(\varsigma_i)$. Then $\mathrm{Pad}(\varsigma_{i+1}) = \mathrm{Pad}(X{=}Z \wedge \varsigma_i)$. $\mathrm{Pad}(\varsigma_{i+1})$ is a saturated constraint: For all saturation rules that would become applicable because of the added dominance literals $X{=}Z$, the consequent has already been added by Pad.

Suppose $\varsigma_i =_{\mathcal{G}}^{ex} \varsigma_{i+1}$ by axiom (2) of Fig. 4.16, and ς_{i+1} has the form $\varsigma_i[Y/X]$ where $X \notin \mathcal{G}$ and $Y \notin \mathcal{V}ar(\varsigma_i) \cup \mathcal{G}$. So all occurrences of a variable $X \notin \mathcal{G}$ have been replaced by a new variable $Y \notin \mathcal{G}$, and if $\mathrm{Pad}(\varsigma_i)$ is saturated, then so is $\mathrm{Pad}(\varsigma_i)[Y/X] = \mathrm{Pad}(\varphi_2')$.

In both cases where $\varsigma_i =^{ex}_{\mathcal{G}} \varsigma_{i+1}$ by axiom (3), we obviously have $\mathrm{Pad}(\varsigma_i) = \mathrm{Pad}(\varsigma_{i+1})$.

So we have $\varphi_2 =^{ex}_{\mathcal{G}} \varphi'_2 =^{ex}_{\mathcal{G}} \mathrm{Pad}(\varphi'_2)$, and $\mathrm{Pad}(\varphi'_2)$ is $\mathcal{P}_\mathrm{p}$-saturated. It remains to show that $\varphi_1 \subseteq \mathrm{Pad}(\varphi'_2)$. This last step is easy: We have $\varphi_1 \subseteq \varphi'_2$ since that is what we have assumed in the beginning, and $\varphi'_2 \subseteq \mathrm{Pad}(\varphi'_2)$ by the definition of padded constraints. □

So, having set up the family $\leq_{\mathcal{G}}$ of partial orders that we are going to use to compare $\mathrm{CLLS_p}$ constraints, we now make use of these orders as follows:

Definition 4.33 (Computation with respect to $\mathcal{G}$). *Let φ, ς be $\mathcal{C}_\mathrm{pp}$ constraints, and let $\mathcal{G} \subseteq \mathcal{V}ar$. Then $\mathcal{P}_\mathrm{p}$* can compute ς from φ with respect to $\mathcal{G}$ *iff there exists a $\mathcal{P}_\mathrm{p}$-saturation ς' for φ with respect to $\mathcal{G}$ such that $\varphi \to^*_{\mathcal{P}_\mathrm{p}} \varsigma'$, and $\varsigma =_{\mathcal{G}} \varsigma'$.*

4.5 Completeness of Procedure $\mathcal{P}_\mathrm{p}$

In this section we show that $\mathcal{P}_\mathrm{p}$ is complete, i.e. that it computes all $\leq_{\mathcal{V}ar(\varphi)}$-minimal saturated constraints for a given constraint φ. We proceed in two steps. The first step is as in the previous chapter (Lemma 3.17, p. 70): Given a constraint and a minimal saturated constraint for it, we show that we can apply each applicable rule in such a way that we move closer to this saturated constraint. However whereas this one step sufficed in the previous chapter, we now have to add a second step: We have to show additionally that after a finite number of steps we actually reach the saturated constraint towards which we are moving.

Lemma 4.34 (Approaching a saturation). *Let φ be a $\mathcal{C}_\mathrm{pp}$ constraint, $\mathcal{G} \subseteq \mathcal{V}ar$, and ς a $\mathcal{P}_\mathrm{p}$-saturated $\mathcal{C}_\mathrm{pp}$-constraint with $\varphi \leq_{\mathcal{G}} \varsigma$. If a rule $\rho \in \mathcal{P}_\mathrm{p}$ is applicable to φ, then there exists a constraint φ' satisfying $\varphi \to_{\{\rho\}} \varphi'$ and $\varphi' \leq_{\mathcal{G}} \varsigma$.*

Proof. By Lemma 4.32 there exists a $\mathcal{P}_\mathrm{p}$-saturated constraint ς' with $\varphi \subseteq \varsigma' =^{ex}_{\mathcal{G}} \varsigma$. Suppose ρ is a rule $\overline{\varphi} \to \bigvee_{i=1}^n \overline{\varphi}_i$ that is not an instance of (P.new). Then by the same argument as in the proof of Lemma 3.17 there exists an i such that $\overline{\varphi}_i \subseteq \varsigma'$, hence $\varphi \wedge \overline{\varphi}_i \subseteq \varsigma'$. Now suppose that ρ is an instance of (P.new). Let ρ be $\overline{\varphi} \to \exists X'.A \sim^{\mathrm{sym}} B \wedge \mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ X & X' \end{smallmatrix}) \wedge X \in \mathsf{b}(A)$ with $X' \notin \mathcal{G} \cup \mathcal{V}ar(\varphi)$. We must have $\mathrm{p}(\begin{smallmatrix} X_0 & Y_0 \\ X & Y \end{smallmatrix}) \in \varsigma$ for some variable Y. But then by axiom (2) of Fig. 4.16, we have $\varsigma' =^{ex}_{\mathcal{G}} \varsigma'[Z'/X']$ for some $Z' \notin \mathcal{G} \cup \mathcal{V}ar(\varsigma') \cup \mathcal{V}ar(\varphi)$, which by axiom (1) is $=^{ex}_{\mathcal{G}}$ equivalent to $\varsigma'[Z'/X'] \wedge Y{=}X'$, which in turn equals $\varsigma'[Z'/X'] \wedge Y{=}X' \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X & X' \end{smallmatrix})$ by axiom (3). Call this last constraint ς'', then $\varphi \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X & X' \end{smallmatrix}) \subseteq \varsigma'' =^{ex}_{\mathcal{G}} \varsigma$. □

For the algorithm $\mathcal{P}_\mathrm{d}$ that we have discussed in the previous chapter, we have argued that the saturation rules never introduce additional variables, so there are only finitely many literals that the algorithm can possibly add to a constraint. Hence after a finite number of steps we must reach the minimal saturated constraint we are moving towards.

However, things are different with $\mathcal{P}_\mathrm{p}$, since (P.new) introduces additional variables into the constraint. To prove completeness of $\mathcal{P}_\mathrm{p}$, we use a *distance measure* between a

constraint φ and a minimal saturated constraint ς for it. The two elements of the measure are: the number of nodes in the constraint graph for ς that are not present in the constraint graph for φ; and the number of correspondences still to be computed for the variables that are present in φ. Then, to show that $\mathcal{P}_\mathrm{p}$ can actually *reach* any given minimal saturated constraint of a constraint, we show that after applying one instance of (P.new) and then saturating the constraint under all other rules, we have made the distance from the minimal saturated constraint strictly smaller.

Definition 4.35 (Lacking correspondents). *Let φ be a $\mathcal{C}_\mathrm{pp}$ constraint and $\mathcal{S} \subseteq \mathcal{V}ar(\varphi)$. Then we define the number $\mathsf{lc}(\mathcal{S},\varphi)$ of* lacking correspondents *in φ by*

$$\mathsf{lc}(\mathcal{S},\varphi) = \sum \left\{ \mathsf{lc}^{A,B}(X,\varphi) + \mathsf{lc}^{B,A}(X,\varphi) \mid X \in S \textit{ and } A \sim^{\mathrm{sym}} B \textit{ in } \varphi \right\}$$

where we fix the values of the auxiliary terms by setting for all segment terms A with root variable X_0 and B with root variable Y_0 and for all $X \in \mathcal{V}ar(\varphi)$:

$$\mathsf{lc}^{A,B}(X,\varphi) = \begin{cases} 1 & \textit{if } X \in \mathsf{b}(A) \textit{ in } \varphi \textit{ but } p\binom{X_0\ Y_0}{X\ X'} \textit{ is not in } \varphi \textit{ for any } X' \\ 0 & \textit{otherwise} \end{cases}$$

Definition 4.36 (Inequality set). *For $\mathcal{C}_\mathrm{pp}$ constraints $\varphi_1 \subseteq \varphi_2$, let $\mathsf{diff}(\varphi_1,\varphi_2)$ be the size of the set $\{X \in \mathcal{V}ar(\varphi_2) \mid X \neq Y \in \varphi_2 \textit{ for all } Y \in \mathcal{V}ar(\varphi_1)\}$.*

We call a set $\mathcal{S} \subseteq \mathcal{V}ar(\varphi)$ of variables an inequality set *for φ iff $X \neq Y \in \varphi$ for any distinct $X, Y \in \mathcal{S}$.*

For constraints φ_2 that are saturated with respect to (P.distr.eq), $\mathsf{diff}(\varphi_1,\varphi_2)$ is the number of variables X in φ_2 such that $X{=}Y$ not in φ_2 for all $Y \in \mathcal{V}ar(\varphi_1)$.

Definition 4.37 ($\mathcal{G}$-measure). *Let φ, ς be $\mathcal{C}_\mathrm{pp}$ constraints and $\mathcal{G} \subseteq \mathcal{V}ar$ with $\varphi \leq_\mathcal{G} \varsigma$. Then the $\mathcal{G}$-*measure *$\mu_\mathcal{G}(\varphi,\varsigma)$ for φ and ς is the pair $\left(\mu^1_\mathcal{G}(\varphi,\varsigma), \mu^2(\varphi)\right)$, where:*

- $\mu^1_\mathcal{G}(\varphi,\varsigma) = min\{\mathsf{diff}(\varphi,\varsigma') \mid \varphi \subseteq \varsigma' =^{ex}_\mathcal{G} \varsigma \textit{ and } \varsigma' \textit{ is } \mathcal{P}_\mathrm{p}\textit{-saturated}\}$
- $\mu^2(\varphi) = min\{\mathsf{lc}(\mathcal{S},\varphi) \mid \mathcal{S} \textit{ is a maximal inequality set for } \varphi\}$.

We order $\mathcal{G}$-measures by the lexicographic ordering $<$ on sequences of natural numbers, which is well-founded.

Let $\mathcal{P}_\mathrm{new}$ be the set of all instances of (P.new), and let $\mathcal{P}_\mathrm{o}$ be $\mathcal{P}_\mathrm{p} - \mathcal{P}_\mathrm{new}$. The main idea of the following proof is that after each $\rightarrow_{\mathcal{P}_\mathrm{new}}$ step and subsequent $\mathcal{P}_\mathrm{o}$ saturation, the $\mathcal{G}$-measure between a constraint and the minimal saturation that we are moving towards has strictly decreased: Either we have introduced a new node in the constraint graph, which decreases μ^1, even though the new variable may need more correspondents, thus increasing μ^2. Or we have made a correspondent-lacking variable correspond to a variable already present in the constraint. This leaves μ^1 unchanged but decreases μ^2. In formulating the following lemma, we make use of Def. 4.33:

Lemma 4.38 (Completeness). *Let φ be a $\mathcal{C}_{\mathrm{pp}}$ constraint and $\mathcal{G} \subseteq \mathcal{V}ar(\varphi)$. Then $\mathcal{P}_{\mathrm{p}}$ can compute from φ, in a finite number of steps, any minimal $\mathcal{P}_{\mathrm{p}}$-saturation for φ with respect to $\mathcal{G}$.*

Proof. Let ς be a minimal $\mathcal{P}_{\mathrm{p}}$-saturation for φ with respect to $\mathcal{G}$. What we want to show is that there is a $\mathcal{P}_{\mathrm{p}}$-saturation $\varsigma' =_{\mathcal{G}} \varsigma$ such that $\varphi \rightarrow^*_{\mathcal{P}_{\mathrm{p}}} \varsigma'$ in a finite number of steps.

W.l.o.g. let φ be $\mathcal{P}_{\mathrm{o}}$-saturated. If no rule from $\mathcal{P}_{\mathrm{new}}$ is applicable to φ then $\varphi =_{\mathcal{G}} \varsigma$ by the minimality of ς. If a rule $\rho \in \mathcal{P}_{\mathrm{new}}$ is applicable to φ, then by Lemma 4.34 there exist φ', φ'' such that $\varphi \rightarrow_{\{\rho\}} \varphi'' \rightarrow^*_{\mathcal{P}_{\mathrm{o}}} \varphi' \leq_{\mathcal{G}} \varsigma$, and φ' is $\mathcal{P}_{\mathrm{o}}$-saturated.

We show that for the $\mathcal{P}_{\mathrm{o}}$-saturated φ, the constraint φ' that results from one application of (P.new) and subsequent $\mathcal{P}_{\mathrm{o}}$-saturation, and the minimal $\mathcal{P}_{\mathrm{p}}$-saturation ς for φ, we have $\mu_{\mathcal{G}}(\varphi', \varsigma) < \mu_{\mathcal{G}}(\varphi, \varsigma)$. Note that because φ is $\mathcal{P}_{\mathrm{o}}$-closed, a maximal inequality set within φ contains exactly one variable from each syntactic variable equivalence class represented in φ; and $\mathsf{lc}(\{X\}, \varphi) = \mathsf{lc}(\{Y\}, \varphi)$ whenever $X{=}Y$ is in φ because of saturation under (P.path.eq.1). For any $\mathcal{P}_{\mathrm{p}}$-saturation ς' with $\varphi \subseteq \varsigma' =^{ex}_{\mathcal{G}} \varsigma$, the value of $\mathsf{diff}(\varphi, \varsigma')$ is minimal (i.e. equal to $\mu^1_{\mathcal{G}}(\varphi, \varsigma)$) if for any $Y \in \mathcal{V}ar(\varsigma')$ such that $Y{\neq}X$ in ς' for all $X \in \mathcal{V}ar(\varphi)$ the following holds: Y is not in $\mathcal{G}$ (since $\mathcal{V}ar(\varsigma') \cap \mathcal{G} = \mathcal{V}ar(\varsigma) \cap \mathcal{G} = \mathcal{V}ar(\varphi) \cap \mathcal{G}$, otherwise ς would not be a *minimal* saturation for φ with respect to $\mathcal{G}$) and there is no variable $Z \in \mathcal{V}ar(\varsigma')$ distinct from Y with $Y{=}Z$ in ς'.

Let the rule ρ be $\exists X'.\mathrm{co}(A,B)(X'){=}X$, i.e. $X' \notin \mathcal{G}$ is the variable newly introduced by ρ. In φ', (P.distr.eq) has been applied to X' and all variables in $\mathcal{V}ar(\varphi)$. Let $\varsigma' =^{ex}_{\mathcal{G}} \varsigma$ with $\varphi \subseteq \varsigma'$ and minimal $\mathsf{diff}(\varphi, \varsigma')$. The constraint ς' contains $\mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X & Z \end{smallmatrix})$ for some Z. W.l.o.g. we pick a ς' that does not contain X'.

- If $X'{=}Y$ is in φ' for some $Y \in \mathcal{V}ar(\varphi)$, then $\mu^2(\varphi') < \mu^2(\varphi)$ and $\mu^1_{\mathcal{G}}(\varphi', \varsigma) = \mu^1_{\mathcal{G}}(\varphi', \varsigma)$: We first show that $\mu^2(\varphi') < \mu^2(\varphi)$. We have $\mathsf{lc}(\{V\}, \varphi') < \mathsf{lc}(\{X\}, \varphi)$ for all variables $V \in \mathcal{V}ar(\varphi)$ such that $V{=}X$ is in φ', and either X or some other member of its equivalence class must be in each maximal inequality set. At the same time, a maximal inequality set within φ' can contain only one of X' and Y, so X' contributes nothing additional to $\mu^2(\varphi')$.

  Now we show that $\mu^1_{\mathcal{G}}(\varphi', \varsigma) = \mu^1_{\mathcal{G}}(\varphi', \varsigma)$. Let ς'' be $\mathrm{Pad}(\varsigma' \wedge X'{=}Z)$. Then ς'' is $\mathcal{P}_{\mathrm{p}}$-saturated by the proof of Lemma 4.32, and $\varphi' \subseteq \varsigma' \subseteq \mathrm{Pad}(\varsigma' \wedge X'{=}Z)$. We have $\mathsf{diff}(\varphi', \varsigma'') = \mathsf{diff}(\varphi, \varsigma')$ because $\mathcal{V}ar(\varphi') - \mathcal{V}ar(\varphi) = \mathcal{V}ar(\varsigma'') - \mathcal{V}ar(\varsigma') = \{X'\}$, and X' belongs to the same equivalence class as Y, which occurs in φ and ς too. Furthermore $\mathsf{diff}(\varphi', \varsigma'')$ is minimal because $\mathsf{diff}(\varphi, \varsigma')$ is, and the only variable in $\mathcal{V}ar(\varsigma'') - \mathcal{V}ar(\varsigma')$ is X', which is not different from all variables in φ' and thus does not contribute to $\mathsf{diff}(\varphi', \varsigma'')$.

- If $X'{\neq}Y$ is in φ' for all $Y \in \mathcal{V}ar(\varphi)$, then $\mu^1_{\mathcal{G}}(\varphi', \varsigma) < \mu^1_{\mathcal{G}}(\varphi, \varsigma)$: We must have $Z{\neq}Y \in \varsigma'$ for all $Y \in \mathcal{V}ar(\varphi')$ because (P.distr.eq) has contributed $X'{\neq}Y$ to φ' for all $Y \in \mathcal{V}ar(\varphi)$, and we have assumed that all rules are applied in such a way that the resulting clause is still $\leq_{\mathcal{G}} \varsigma$. And this means that by the minimality of

$\mathsf{diff}(\varphi, constrII')$, $Z \notin \mathcal{G}$ and that $Z{=}Z'$ is not in ς' for any variable Z' distinct from Z, as pointed out above.

Now let ς'' be $\varsigma'[X'/Z]$. Then we have $\varsigma' =^{ex}_{\mathcal{G}} \varsigma''$ by axiom (2) since $Z \notin \mathcal{G}$. For the same reason ς'' is a $\mathcal{P}_{\mathrm{p}}$-saturated constraint, and we have $\varphi' \subseteq \varsigma''$. Furthermore, $\mathsf{diff}(\varphi', \varsigma'') = \mathsf{diff}(\varphi, \varsigma) - 1$ because we must have had $Z{\neq}V$ in ς' for all $V \in \mathcal{V}ar(\varphi)$.

So in any case, the μ-distance between φ' and ς is strictly smaller than the μ-distance between φ and ς. Since we can always decrease the distance from ς in a finite number of $\mathcal{P}_{\mathrm{p}}$-computation steps, the procedure $\mathcal{P}_{\mathrm{p}}$ can compute a saturation for φ that is $=_{\mathcal{G}}$-equal to ς in a finite number of steps. □

So $\mathcal{P}_{\mathrm{p}}$ can compute all minimal saturations for a given constraint. We can say even more: $\mathcal{P}_{\mathrm{p}}$ computes *exactly* the minimal saturations. In Chapter 3, p. 70, we have argued that the only rules that can lead to the computation of nonminimal saturations are distribution rules, and only those where the right-hand side disjuncts are not mutually exclusive. In $\mathcal{P}_{\mathrm{d}}$, and also in $\mathcal{P}_{\mathrm{p}}$, there is exactly one such rule, (D.distr.notDisj). With $\mathcal{P}_{\mathrm{d}}$, (D.distr.notDisj) can indeed lead to the computation of nonminimal saturations, where one saturated constraints contains an equality $X{=}Y$ and another contains neither $X{=}Y$ nor $X{\neq}Y$. But $\mathcal{P}_{\mathrm{p}}$ contains the rule (P.distr.eq), which guesses either $X{=}Y$ or $X{\neq}Y$ for each pair of variables. So $\mathcal{P}_{\mathrm{p}}$ only computes minimal saturations.

Each model of a constraint is also a model of one of its minimal saturations.

Proposition 4.39. *Let φ be a $\mathcal{C}_{\mathrm{pp}}$-constraint for which (θ, σ) is a model, and let $\mathcal{G} \subseteq \mathcal{V}ar(\varphi)$. Then φ possesses a $\leq_{\mathcal{G}}$-minimal $\mathcal{P}_{\mathrm{p}}$-saturated constraint that is also satisfied by (θ, σ).*

Proof. The proof of this proposition is the same as for Prop. 3.18 (p. 70). □

4.6 Recapitulation: Properties of the Procedure $\mathcal{P}_{\mathrm{p}}$

In the previous sections, we have shown a number of properties of the procedure $\mathcal{P}_{\mathrm{p}}$, which we now sum up.

Theorem 4.40. *The semi-decision procedure $\mathcal{P}_{\mathrm{p}}$ for parallelism constraints has the following properties:*

1. *It is sound for lambda structures.*
2. *There are unsatisfiable parallelism constraints for which it does not terminate.*
3. *A generated $\mathcal{P}_{\mathrm{p}}$-saturated $\mathcal{C}_{\mathrm{pp}}$-constraint is satisfiable.*

4. *$\mathcal{P}_\mathrm{p}$ is complete: Given a $\mathcal{C}_\mathrm{pp}$ constraint φ and a set $\mathcal{G} \subseteq \mathcal{V}ar(\varphi)$, $\mathcal{P}_\mathrm{p}$ can compute from φ, in a finite number of steps, any minimal $\mathcal{P}_\mathrm{p}$-saturation for φ with respect to $\mathcal{G}$.*

5. *This set of minimal $\mathcal{P}_\mathrm{p}$-saturations for a parallelism constraint may be infinite.*

Proof. 1. by Lemma 4.10, 2. by Ex. 4.7, 3. by Lemma 4.22, 4. by Lemma 4.38, 5. by Lemma 4.12. □

4.7 Related Work

The closest relative of parallelism constraints is context unification (CU), which we have sketched in Chapter 2. In this section we compare the parallelism constraint procedure that we have just introduced to a context unification procedure. There are different rule-based procedures for CU [84, 93]. We choose the simpler version as a basis for our comparison. This CU procedure is from a paper by Niehren, Pinkal and Ruhrberg [93]. It is shown in Fig. 4.17.

Decomposition:	$f(t_1,\ldots,t_n) = f(t'_1,\ldots,t'_n) \longrightarrow \bigwedge_{i=1}^n t_i = t'_i \mid Id$
Substitution:	$x = t \longrightarrow true$ if $x \notin \mathcal{V}ar(t) \mid x \mapsto t$
Orient:	$t = X \longrightarrow X = t \mid Id$ for $X \in \mathcal{V}_1 \cup \mathcal{V}_2$
Projection:	$f(t_1,\ldots,t_n) = C(t') \longrightarrow f(t_1,\ldots,t_n) = t' \mid C \mapsto \lambda x.x$
Imitation	$f(t_1,\ldots,t_n) = C(t') \longrightarrow t_i = C'(t')$ $\mid C \mapsto \lambda x.f(t_1,\ldots,t_{i-1}, C'(x), t_{i+1},\ldots,t_n)$
Simplification:	$C(t) = C(t') \longrightarrow t = t' \mid Id$
Flex-Flex1:	$C(t) = C'(t') \longrightarrow t = C''(t') \mid C' \mapsto \lambda x.C(C''(x))$
Flex-Flex2:	$C(t) = C'(t') \longrightarrow true$ $\mid C \mapsto \lambda y.C_1(f(\varpi(\overline{x}, C_2(y), C_3(t'))))$, $C' \mapsto \lambda z.C_1(f(\varpi(\overline{x}, C_2(t), C_3(z))))$, where ϖ is a permutation

Figure 4.17: A procedure for context unification

The procedure has the form of a *state transformer*. A state is a pair $\langle E, \sigma\rangle$ of a set E of equations and a substitution σ (which we lift canonically from terms t to equation systems E over terms). For a given equation system E the procedure starts in the state $\langle E, Id\rangle$. The equation system is solved if a final state of the form $\langle\emptyset, \sigma\rangle$ can be reached by an (indeterministic) application of transformation rules, where a transformation rule has the form $t = t' \longrightarrow E \mid \sigma$. When applied to the state $\langle\{t = t'\} \cup E', \sigma'\rangle$ it yields the new state $\langle\sigma(E \cup E'), \sigma \circ \sigma'\rangle$. As above, we view contexts as context functions, linear second order lambda terms of the form $\lambda x.t$, where t is a second order term in which the

first order variable x occurs exactly once. We assume that in performing a substitution $\sigma(E)$ we also beta reduce the terms.

The state transformation rules of the procedure for context unification are given in Fig. 4.17. The two most interesting cases are the Flex-Flex rules. Given a context γ, let us call the path π with $L_\gamma(\pi) = \bullet$ its *exception path.* Then we can describe the two rules as follows. Given an equation $C(t) = C'(t')$, there are two possibilities: Either the hole of C lies on the exception path of C', or the exception paths of C and C' branch at some point. Flex-Flex1 covers the first case. Here t' is a subtree of t. Flex-Flex2 covers the second case. It has to guess a function symbol f (of some arity n) at which the two exception paths branch and a permutation ϖ of an n-ary sequence consisting of contexts C_2 and C_3 "leading to" the holes of C and C', respectively, and $n-2$ fresh variables $\overline{x}$.

As we have remarked in Chapter 2, CLLS has evolved from a CU approach to underspecification [95]. In that analysis, both scope and ellipsis were modeled by context variables. To process the ensuing CU equation systems, the procedure in Fig. 4.17 was used, but the two Flex-Flex rules were omitted to keep the computation tractable [74], which makes the procedure incomplete.

So what are the advantages of using a dedicated procedure for parallelism constraints instead of falling back on the CU procedure? The biggest advantage is that the parallelism constraint procedure incorporates a dominance constraint solver: It can use a dedicated, faster sub-procedure for handling dominance constraints, while a CU procedure cannot discriminate between dominance and parallelism – dominance constraints do not seem to correspond to any clear-cut fragment of context unification. Furthermore the CU procedure determines the shape of a context in a top-down fashion, starting at the root of the context and working downward. In the process, it sometimes has to guess labels. In contrast, the parallelism constraint procedure handles parallelism literals without any preferred direction, and it never has to guess labels.

4.8 The Search Tree that $\mathcal{P}_\mathrm{p}$ Explores

In the search tree that $\mathcal{P}_\mathrm{p}$ explores, the nodes are constraints, and if a saturation rule $\varphi_0 \rightarrow \bigvee_{i=1}^{n} \exists V_i \varphi_i$ is applied to a constraint φ, then the node φ has n children $\varphi \wedge \varphi'_i$ for $1 \leq i \leq n$ and fresh V_i-variants φ'_i of φ_i. A leaf of the search tree can be either succeeded, in which case it is a $\mathcal{P}_\mathrm{p}$-saturated constraint that does not contain `false`, or failed, in which case it contains `false`.

What does completeness of $\mathcal{P}_\mathrm{p}$ mean stated in terms of search trees? The first thing to note is that there may be more than one search tree starting with the same constraint at the root, since the application of saturation rules is don't care indeterministic. So completeness means that in all search trees with the same constraint φ at the root and for each model θ of φ there must be a succeeded leaf at finite depth satisfied by θ. There are search trees with infinite branches, for example any search tree for the unsatisfiable constraint of Ex. 4.7. For the constraint of Ex. 4.8 the fairness condition – (P.new) is

applied only to constraints saturated under $\mathcal{P}_\mathrm{o}$ – is necessary to ensure that there are success nodes at finite depth.

Another interesting question to study in connection with the search tree of $\mathcal{P}_\mathrm{p}$ is *finite failure*. Suppose that for a given constraint φ there exists a search tree with root φ that is finite, and all its leaves are failed. Then, is $\mathcal{P}_\mathrm{p}$ guaranteed to find that search tree, or can it still diverge into an infinite branch of some other search tree with root φ? The question of finite failure has been extensively studied in connection with Negation as Failure [4]. A literal A is in the finite failure set of a program P if there exists a finitely failed SLD-tree with $\leftarrow A$ as root. Lassez and Maher [83] show that for programs P and ground atoms A, A is in the finite failure set of P iff every *fair* SLD-tree with $\leftarrow A$ as root is finitely failed. Here, fairness means that the tree is either finite, or every atom appearing in it is eventually selected.

Is the two-level control we use with the saturation rules of $\mathcal{P}_\mathrm{p}$ sufficient to guarantee that we find each finitely failed search tree? The answer to this question is not known yet. The failed search tree might include some edges that are instances of (P.new). Can we guarantee that we can always find the "right" instances, the ones that do not lead to infinite search tree branches?

4.9 Summary

In this central chapter we have introduced the procedure $\mathcal{P}_\mathrm{p}$ for parallelism constraints. It extends the solver $\mathcal{P}_\mathrm{d}$ for dominance constraints, which we have discussed in the previous chapter. Like $\mathcal{P}_\mathrm{d}$, $\mathcal{P}_\mathrm{p}$ is a saturation procedure. It keeps on extending a set of clauses until a state of saturation is reached, and it never eliminates any information it has gathered.

The main idea in solving parallelism literals is to compute *syntactic correspondence functions*. The procedure makes sure that each variable occurring in one of the two parallel segment terms has a correspondent in the other segment term, and copies all material from one parallel segment term to the other.

The *correspondence formulas* that make up a syntactic correspondence function are expressed by *path parallelism literals*. Path parallelism states that the tree path between two nodes is the same as the path between two other nodes, including the labels encountered on the way. The properties of path parallelism literals, expressed as saturation rules, enforce the right interaction between different syntactic correspondence functions.

Fairness is ensured by a control on the order of rule applications: the rules that introduce new variables are applied only to constraints saturated under all other rules.

The procedure is sound: All rules are equivalence transformations. Also, each $\mathcal{P}_\mathrm{p}$-saturated constraint is satisfiable. We have shown how to construct a model from a given saturated constraint. For saturated constraints that already look like trees, we can name a satisfying tree directly, using the models for $\mathcal{P}_\mathrm{d}$-saturated constraints that we

have constructed in the previous chapter. For other saturated constraints we again label unlabeled variables until a tree-shaped constraint is reached, but this time we have to label all variables linked by correspondence at the same time.

The procedure $\mathcal{P}_{\mathrm{p}}$ is complete: It computes all minimal saturated constraints for a given input constraint. While in the previous chapter we have considered minimality with respect to subset inclusion, we now use the family of partial orders $\leq_{\mathcal{G}}$, parametrized by a set $\mathcal{G} \subseteq \mathcal{V}ar$ of variables, that can be described as subset inclusion modulo α-renaming of variables introduced during computation with $\mathcal{P}_{\mathrm{p}}$.

We have shown that $\mathcal{P}_{\mathrm{p}}$ is complete: First, given a constraint to which a rule is applicable, and a minimal saturation for it, we can apply the rule in such a way that we move closer to the minimal saturation in question. Second, when we move towards such a minimal saturation by consecutive rule applications, the saturationis actually reached in a finite number of steps. To show this, we have used a distance measure between constraint and saturation, a measure that counts the number of variables that still need to be introduced in the constraint and the number of correspondences still to be fixed. The fairness condition plays a critical role in the completeness proof.

Chapter 5

Solving CLLS Constraints

In this chapter we complete the semi-decision procedure for CLLS. It incorporates the procedures we have been discussing in the previous two chapters: the semi-decision procedure $\mathcal{P}_\mathrm{p}$ for parallelism constraints, and hence also the solver $\mathcal{P}_\mathrm{d}$ for dominance constraints (which forms part of the procedure $\mathcal{P}_\mathrm{p}$). What we add to the procedure $\mathcal{P}_\mathrm{p}$ in the current chapter are rules for handling lambda and anaphoric binding. We again formulate the procedure within the framework of *saturation*: It keeps on adding material to a set of clauses until a state of saturation is reached (i.e. a set of saturated constraints), in which nothing new can be added anymore.

The new saturation rules that we introduce in the current chapter are rather simple: They just implement the conditions on the interaction of parallelism and binding in lambda structures that we have laid down in Chapter 2.

We prove the semi-decision procedure for CLLS sound and complete, using again the same proof schemata as in the previous chapter. Finally, we sum up all saturation rules of the procedure, the new rules as well as those that we have discussed in the previous two chapters.

5.1 A Semi-Decision Procedure for CLLS: $\mathcal{P}$

In this section we present a semi-decision procedure for the constraint language CLLS. Again, the procedure tries to find out, for a given constraint, whether there exists a *model*, a lambda structure that satisfies the constraint. Given a satisfiable constraint, the procedure computes a set of *saturations*. From each of those a model can be read off.

We again formulate this procedure as a saturation procedure: It operates on a set of clauses, adding more and more material to them until nothing new can be added anymore. A saturation rule ρ has the form $\varphi_0 \to \bigvee_{i=1}^{n} \exists V_i \varphi_i$, where for $0 \le i \le n$, φ_i is a clause and for $1 \le i \le n$, V_i is a set of variables. The rule is applicable to a clause ς if the application condition appc_ρ holds. This condition, laid down in Def. 4.1 (p. 76), basically states that ρ is applicable only if none of its consequences φ_i, $1 \le i \le n$, is already in ς (modulo renaming of the existentially quantified variables V_i). See the beginning of Chapters 3 and 4 for formal definitions and more detailed explanations of saturation procedures.

In the previous chapter we have introduced a few formulas that we reuse here. Remember that these formulas may contain disjunctions: A disjunction on the right-hand side of a saturation rule just makes it a distribution rule, but if a disjunction occurs on the left-hand side of a rule, then this rule is actually an abbreviation for a *set* of rules, in the way explained at the beginning of Chapter 4. We reuse the following formulas: Let $A = X_0/X_1, \ldots, X_n$, $B = Y_0/Y_1, \ldots, Y_n$ be segment terms. Then

$$\begin{array}{rcl} A \sim^{\text{sym}} B & =_{\text{def}} & A \sim B \;\vee\; B \sim A \\ \mathsf{seg}(A) & =_{\text{def}} & \bigwedge_{i=1}^{n} X_0 \triangleleft^* X_i \wedge \bigwedge_{1 \le i < j \le n} \big((X_i \bot X_j) \vee (X_i{=}X_j)\big) \\ X \in \mathsf{b}(A) & =_{\text{def}} & X_0 \triangleleft^* X \;\wedge\; \bigwedge_{i=1}^{n} (X \triangleleft^* X_i \vee X \bot X_i) \\ X \in \mathsf{b}^-(A) & =_{\text{def}} & X \in \mathsf{b}(A) \wedge \bigwedge_{i=1}^{n} (X {\neq} X_i \vee X \bot X_i) \\ X {\notin} \mathsf{b}(A) & =_{\text{def}} & X \triangleleft^+ X_0 \vee X \bot X_0 \vee \bigvee_{i=1}^{n} X_i \triangleleft^+ X \\ X {\notin} \mathsf{b}^-(A) & =_{\text{def}} & X \triangleleft^+ X_0 \vee X \bot X_0 \vee \bigvee_{i=1}^{n} X_i \triangleleft^* X \\ \mathrm{co}(A,B)(U){=}V & =_{\text{def}} & A {\sim^{\text{sym}}} B \;\wedge\; \mathrm{p}\big(\begin{smallmatrix} X_0 & Y_0 \\ U & V \end{smallmatrix}\big) \;\wedge\; U \in \mathsf{b}(A). \end{array}$$

5.1.1 The Rules in Detail

Remember that the language CLLS has the following abstract syntax:

$$\begin{array}{rcll} \varphi, \varsigma & ::= & X \triangleleft^* Y \mid X{:}f(X_1, \ldots, X_n) \mid X \bot Y \mid X {\neq} Y & (\mathsf{ar}(f) = n) \quad (1) \\ & \mid & X_0/X_1, \ldots, X_n {\sim} Y_0/Y_1, \ldots, Y_n & n \ge 0 \quad (2) \\ & \mid & \lambda(X){=}Y \mid \mathsf{ante}(X){=}Y & (3) \\ & \mid & \mathtt{false} \mid \varphi \;\wedge\; \varsigma & (4) \end{array}$$

The semi-decision procedure $\mathcal{P}$ for CLLS is shown in Fig. 5.1. The first block of rules in Fig. 5.1 implements the conditions laid down in Def. 2.6 (p. 29). (D.λ.func) states that λ is a function: It has the form $\lambda(X){=}Y \wedge \lambda(U){=}V \wedge X{=}U \to Y{=}V$, i.e. each node of a lambda structure has at most one image under λ. The rule (D.ante.func) does the same for ante. (D.λ.dom), which is $\lambda(X){=}Y \to Y \triangleleft^* X$, implements the condition that a lambda binder always dominates its bound nodes. (D.λ.var) fixes the label of bound nodes: By stating $\lambda(X){=}Y \to X{:}\mathsf{var}$, it says that node with a lambda binder must be labeled var. Likewise, (D.ante.ana) states that a node with an anaphoric binder must be labeled ana. The rule (D.λ.lam), which says $\lambda(X){=}Y \to \exists Y'.(Y{:}\mathsf{lam}(Y') \vee Y{:}\forall(Y') \vee Y{:}\exists(Y'))$, makes sure that a lambda binder is labeled either lam, $\forall$, or $\exists$.

The second block of rules realizes the conditions of Def. 2.7 (p. 29). The rule (P.λ.same) matches the condition (λ.same) by stating $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \to \lambda(V_1){=}V_2$: For a variable bound within the same segment term, the corresponding variable is bound correspondingly. (P.λ.out) implements the condition (λ.out): It says $\lambda(U_1){=}Y \wedge \mathrm{co}(A,B)(U_1){=}V_1 \wedge U_1 \in \mathsf{b}^-(A) \wedge Y \triangleleft^+ X_0 \to \lambda(V_1){=}Y$ for $A = X_0/\ldots$, i.e., if a variable is bound above its segment term, then its correspondent must be bound at the same binder. For the condition (λ.hang) we have (P.λ.hang), an additional clash rule, which is $\lambda(U_1){=}U_2 \wedge A \sim B \wedge U_2 \in \mathsf{b}^-(A) \wedge U_1 {\notin} \mathsf{b}^-(A) \to \mathtt{false}$. It declares a constraint unsatisfiable if a lambda binder inside a segment term binds a variable that

Let $A = X_0/X_1, \ldots, X_n$ and $B = Y_0/Y_1, \ldots, Y_n$.

(D.λ.func)	$\lambda(X){=}Y \wedge \lambda(U){=}V \wedge X{=}U \rightarrow Y{=}V$
(D.λ.dom)	$\lambda(X){=}Y \rightarrow Y \triangleleft^* X$
(D.λ.var)	$\lambda(X){=}Y \rightarrow X{:}\mathsf{var}$
(D.λ.lam)	$\lambda(X){=}Y \rightarrow \exists Y'.(Y{:}\mathsf{lam}(Y') \vee Y{:}\forall(Y') \vee Y{:}\exists(Y'))$
(D.ante.func)	$\mathsf{ante}(X){=}Y \wedge \mathsf{ante}(U){=}V \wedge X{=}U \rightarrow Y{=}V$
(D.ante.ana)	$\mathsf{ante}(X){=}Y \rightarrow X{:}\mathsf{ana}$
(P.λ.same)	$\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \rightarrow \lambda(V_1){=}V_2$
(P.λ.out)	$\lambda(U_1){=}Y \wedge \mathrm{co}(A,B)(U_1){=}V_1 \wedge U_1 \in \mathsf{b}^-(A) \wedge Y \triangleleft^+ X_0 \rightarrow \lambda(V_1){=}Y$
(P.λ.hang)	$\lambda(U_1){=}U_2 \wedge A \sim B \wedge U_2 \in \mathsf{b}^-(A) \wedge U_1 \notin \mathsf{b}^-(A) \rightarrow \mathtt{false}$
(P.ante.same)	$\mathsf{ante}(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \rightarrow$ $\mathsf{ante}(V_1){=}U_1 \;\vee\; \mathsf{ante}(V_1){=}V_2$
(P.ante.out)	$\mathsf{ante}(U_1){=}U_2 \wedge \mathrm{co}(A,B)(U_1){=}V_1 \wedge U_2 \notin \mathsf{b}(A) \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \rightarrow$ $\mathsf{ante}(V_1){=}U_2$
(P.ante.distr)	$\mathsf{ante}(U_1){=}U_2 \wedge A \sim B \wedge U_1 \in \mathsf{b}^-(A) \rightarrow X_0 \triangleleft^* U_2 \;\vee\; U_2 \triangleleft^+ X_0 \;\vee\; U_2 \perp X_0$

plus the rules of the parallelism constraint procedure $\mathcal{P}_{\mathrm{p}}$ in Fig. 4.3, p. 81.

Figure 5.1: Solving CLLS constraints: procedure $\mathcal{P}$.

is below the segment term. The distribution rule (P.ante.same) implements the condition (ante.same): It says $\mathsf{ante}(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \rightarrow \mathsf{ante}(V_1){=}U_1 \;\vee\; \mathsf{ante}(V_1){=}V_2$, i.e. if a variable has its anaphoric binder within the same segment term, there are two possible anaphoric bindings for its correspondent. These two bindings match the strict and the sloppy reading of anaphora occurring within an ellipsis. (This phenomenon is discussed in Sec. 2.3.4, p. 38.) Note the condition $A \sim B$ in the premise of this rule: The correspondence formula only contains $A \sim^{\mathrm{sym}} B$. Here, for the first time, we need to make use of the fact that parallelism literals are not symmetric, the reason being that the conditions for anaphoric binding are not symmetric. The condition (ante.out) is realized by the rule (P.ante.out), which is $\mathsf{ante}(U_1){=}U_2 \wedge \mathrm{co}(A,B)(U_1){=}V_1 \wedge U_2 \notin \mathsf{b}(A) \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \rightarrow \mathsf{ante}(V_1){=}U_2$: If a variable is anaphorically bound outside its segment term, then its correspondent must have the same anaphoric binder. Again, we have $A \sim B$ in the premise because the condition (ante.out) is not symmetric. The distribution rule (P.ante.distr), which states $\mathsf{ante}(U_1){=}U_2 \wedge A \sim B \wedge U_1 \in \mathsf{b}^-(A) \rightarrow X_0 \triangleleft^* U_2 \;\vee\; U_2 \triangleleft^+ X_0 \;\vee\; U_2 \perp X_0$, makes sure that we can always decide whether to apply (P.ante.same) or (P.ante.out). A similar rule is not needed for lambda binding: (D.λ.dom) together with (D.distr.notDisj) already enforces a decision between (P.λ.same) and (P.λ.out).

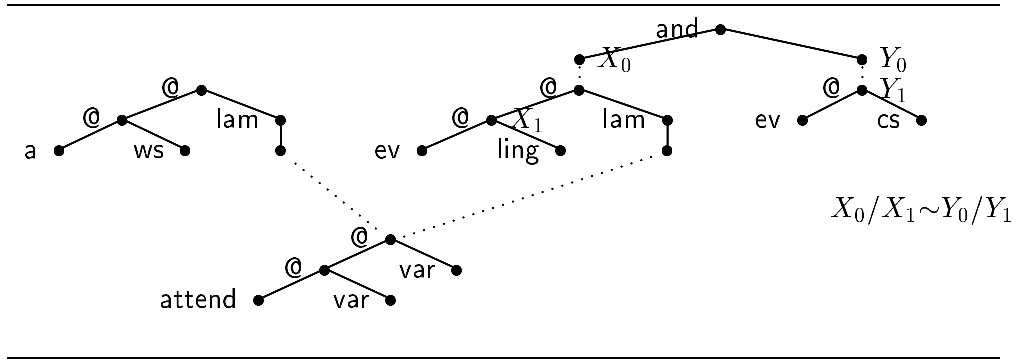


Figure 5.2: Constraint for sentence (2.6): "Every linguist attended a workshop. Every computer scientist did, too."

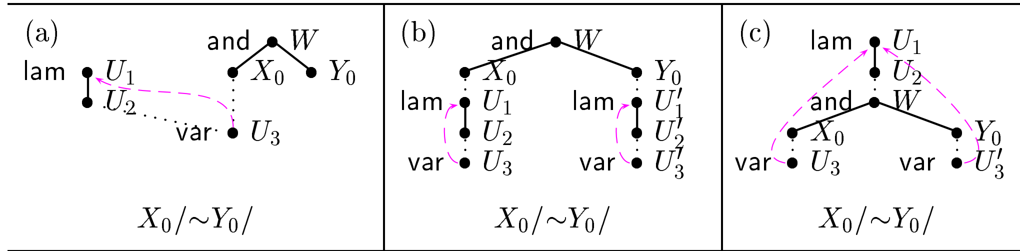


Figure 5.3: Applying lambda binding rules to a simpler version of Fig. 5.2

5.1.2 Examples

Example 5.1 (Lambda binding). The constraint in Fig. 5.2 shows the representation of the sentence "Every linguist attended a workshop. Every computer scientist did, too." What is interesting for our current purpose is the lambda binding from U_1 to U_2, and where its copy in the "target segment term" will be. So we concentrate on the essentials

(1)	$\mathrm{co}(A,B)(X_0)=Y_0$	(P.init)
(2)	$X_0 \lhd^* U_2 \;\vee\; U_2 \lhd^* X_0$	(D.distr.notDisj)

(2a)	$X_0 \lhd^* U_2$:		(2b)	$U_2 \lhd^* X_0$:	
...	$X_0 \lhd^* U_1$		...	$U_2 \lhd^* W$	
(3)	$\exists U'_i.\mathrm{co}(A,B)(U_i)=U'_i$, $1 \le i \le 3$	(P.new)	(8)	$\exists U'_3.\mathrm{co}(A,B)(U_3)=U'_3$	(P.new)
(4)	$Y_0 \lhd^* U'_1, U'_2 \lhd^* U'_3$	(P.copy.dom)	(9)	$Y_0 \lhd^* U'_3$	(P.copy.dom)
(5)	U'_1:$\mathsf{lam}(U'_2)$	(P.copy.lab)	(10)	U'_3:var	(P.copy.lab)
(6)	U'_3:var	(P.copy.lab)	(11)	$U_1 \lhd^+ X_0$	(D.lab.ineq), (P.distr.ineq)
(7)	$\lambda(U'_3)=U'_1$	(P.λ.same)			
			(12)	$\lambda(U'_3)=U_1$	(P.λ.out)

Figure 5.4: Computation of $\mathcal{P}_\mathrm{p}$ on Fig. 5.3 (a)

of the constraint and regard Fig. 5.3 (a) instead. The relevant part of the computation by the procedure $\mathcal{P}$ is shown in Fig. 5.4. Line (2) basically decides between two possible positions for the “a workshop” fragment. This choice determines which lambda binding rule will be applied: Either the rule (P.λ.same) applies in line (7), stating that binding within the two parallel segment terms must be parallel – the constraint that we have at this point is depicted in Fig. 5.3 (b). Or rule (P.λ.out) comes to bear in line (12), binding U_3 and its correspondent U_3' at the same variable outside the two parallel segment terms – at this point, we have the constraint shown in Fig. 5.3 (c).

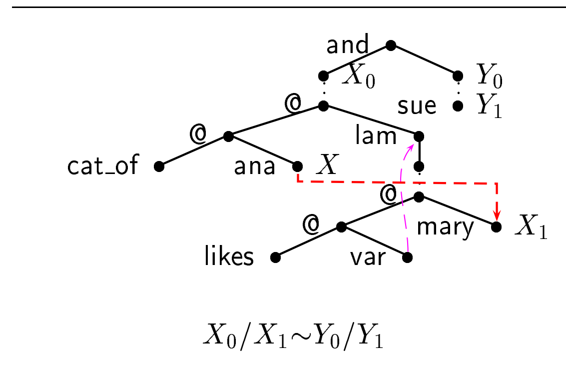


Figure 5.5: The constraint for sentence (2.11): “Mary$_1$ likes her$_1$ cat, and Sue does, too.”

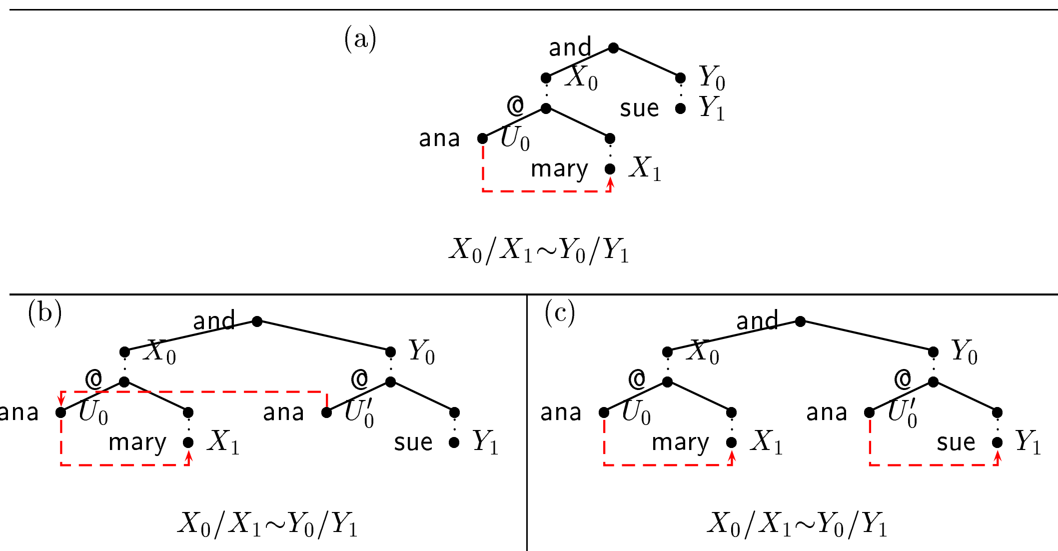


Figure 5.6: Applying anaphoric binding rules to a simpler version of Fig. 2.15

Example 5.2 (Anaphoric binding). Figure 5.5 shows the constraint representing the semantics of the sentence “Mary$_1$ likes her$_1$ cat, and Sue does, too” (where the indices signify that the “her” relates to “Mary”, i.e. that Mary likes her own cat). What is interesting for our current purpose is the anaphoric binding edge ending at the variable

(1)	$X_0 \triangleleft^* X_1 \wedge Y_0 \triangleleft^* Y_1 \wedge$	(P.init)
	$\mathrm{co}(A,B)(X_0){=}Y_0 \wedge \mathrm{co}(A,B)(X_1){=}Y_1$	
(2)	$\exists U_0'.\mathrm{co}(A,B)(U_0){=}U_0'$	(P.new)
(3)	$X_0 \triangleleft^* U_0$	(D.dom.trans)
(4)	$U_0 \perp X_1$	(D.disj)
(5)	$\mathsf{ante}(U_0'){=}U_0 \;\vee\; \mathsf{ante}(U_0'){=}U_0$	(P.ante.same)

Figure 5.7: Computation of $\mathcal{P}_\mathrm{p}$ on Fig. 5.6 (a)

X_1. So again, we use a simplified version of the constraint in Fig. 5.5: the constraint in Fig. 5.6 (a). On this constraint, the procedure $\mathcal{P}$ works as shown in Fig. 5.7. The most interesting line is (5), where (P.ante.same) is applied. It guesses an anaphoric binder for the variable U_0' in the "target segment term". This binder can be either the correspondent of U_0', which is U_0. This is the strict reading; if we saturate the constraint further by copying all labeling literals from the "source segment term" to the "target segment term", we get the constraint depicted in Fig. 5.6 (b). Or the binder of U_0' can be Y_1, which corresponds to the binder X_1 of U_0. This is the sloppy reading. If we now copy all labeling literals from the "source segment term" to the "target segment term", we get the constraint drawn in Fig. 5.6 (c).

5.2 Some Properties of the Procedure: Soundness, Saturations

In this and the following sections we examine properties of the semi-decision procedure $\mathcal{P}$ for CLLS. All results are collected in a theorem in Sec. 5.5.

5.2.1 Soundness

In Sec. 3.5, Def. 3.3, we have stated the notion of soundness that we use: We call a saturation procedure sound if all its rules are equivalence transformations. As we are working in a saturation framework, it suffices to show that in all rules the premise entails the conclusion.

The rules in Fig. 5.1 are direct translations of the conditions laid down in Def. 2.6 and 2.7. So the following lemma obviously holds:

Lemma 5.3 (Soundness). *The semi-decision procedure $\mathcal{P}$ for CLLS is sound for lambda structures.*

5.2.2 Nontermination, Fairness

There are constraints for which the procedure $\mathcal{P}$ does not terminate: This is obvious from the fact that it incorporates the procedure $\mathcal{P}_\mathrm{p}$, which has the same property.

In Sec. 4.2.3, Def. 4.11, we have laid down what we mean by fairness: Whenever a rule is applicable, one of the disjuncts in its conclusion will ultimately be added. The fairness condition we use for the procedure $\mathcal{P}$ is a straightforward adaptation of the condition for the procedure $\mathcal{P}_{\mathrm{p}}$:

> **Fairness condition.** (P.new) is applied only to constraints saturated under $\mathcal{P}-$ {(P.new)}. (P.new) is applied to variables in the order of their introduction into the constraint.

5.2.3 Saturated Constraints

In the previous two chapters we have given a sketch of what constraint graphs of $\mathcal{P}_{\mathrm{d}}$-saturations and $\mathcal{P}_{\mathrm{p}}$-saturations look like. Basically, a constraint graph for a $\mathcal{P}_{\mathrm{d}}$-saturation is a forest, and we have on the one hand labeled nodes – all their outgoing edges are labeling edges, and their children are ordered – and on the other hand unlabeled nodes – all their outgoing edges are dominance edges, and their children are unordered (p. 63). In the constraint graph of a $\mathcal{P}_{\mathrm{p}}$-saturation, there are additional conditions on the "parallel regions of the forest" (p. 90). So what additional features does a constraint graph for a $\mathcal{P}$-saturation have? This is simple: It contains two additional sorts of edges, lambda and anaphoric binding edges, and they basically obey the conditions that we have laid down in the Definitions 2.6 and 2.7 (p. 29) and 29).

5.3 Satisfiability of Saturated Constraints

In this section we show that from each saturated constraint that $\mathcal{P}$ computes, a model can be read off. We proceed as in the two previous chapters: We regard first *simple* constraints, then we extend non-simple saturated constraints to simple ones.

As before, we restrict ourselves to *generated* constraints: They only contain path parallelism literals that could have been added to process parallelism literals, but not path parallelism literals in arbitrary places. We lift the definition of generatedness (Def. 4.14) canonically from $\mathcal{C}_{\mathrm{pp}}$ to CLLS$_{\mathrm{p}}$. Lemma 4.15, which states that all $\mathcal{C}_{\mathrm{pp}}$ constraints computed by $\mathcal{P}_{\mathrm{p}}$ are indeed generated, trivially also holds for CLLS$_{\mathrm{p}}$ constraints and $\mathcal{P}$.

5.3.1 Simple Constraints

In Def. 3.7 (p. 64) we have introduced *simple* constraints: they possess a root variable dominating all others, and every variable is labeled. This definition can be lifted canonically from dominance constraints to CLLS$_{\mathrm{p}}$ constraints: A CLLS$_{\mathrm{p}}$ constraint φ is called simple iff the maximal subset of φ that is a $\mathcal{C}_{\mathrm{d}}$ constraint is simple. In this paragraph we show that every *simple generated $\mathcal{P}$-saturated CLLS$_{\mathrm{p}}$-constraint* is satisfiable.

First, we state that in a simple $\mathcal{P}$-saturated constraint, we have complete information about which variables are inside which segment terms.

Lemma 5.4. *Let φ be a simple $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint. Then for each variable $U \in \mathcal{V}ar(\varphi)$ and each segment term A of φ with $\mathsf{seg}(A)$ in φ, φ contains either $U \in \mathsf{b}(A)$ or $U \notin \mathsf{b}(A)$, and either $U \in \mathsf{b}^-(A)$ or $U \notin \mathsf{b}^-(A)$.*

Proof. Suppose we have a variable $U \in \mathcal{V}ar(\varphi)$ and a segment term A of φ such that $\mathsf{seg}(\mathrm{A})$ is in φ. We want to show that φ contains either $U \in \mathsf{b}(A)$ or $U \notin \mathsf{b}(A)$. Let $A = X/\ldots$. The easy case is $X \triangleleft^* U \in \varphi$ – here we get the desired result by the closure of φ under (P.distr.seg). So suppose otherwise.

As φ has a root variable, there exists a variable Z that is the "lowest" dominating both X and U: $Z \triangleleft^* U$, $Z \triangleleft^* X$ are in φ, and all Z' dominating both U and X in φ also dominate Z. Z is labeled, w.l.o.g. let us assume φ contains a literal $Z{:}f(Z_1, \ldots, Z_n)$. By closure of φ under (D.distr.child) there are four possibilities: Either $Z{=}X$ is in φ, but that is impossible, since we have assumed that X does not dominate U in φ; or $Z{=}U$ is in φ, then $U \triangleleft^+ X$ is in φ by closure under (P.distr.eq); or the same child of Z dominates both U and X, but that is impossible, since we have assumed that Z is the lowest variable dominating both U and X; or there are two different children Z_i, Z_j of Z, $1 \leq i \neq j \leq n$, with $Z_i \triangleleft^* X$ and $Z_j \triangleleft^* U$, then $U \perp X \in \varphi$ by closure under (D.lab.disj) and (D.disj). In any case, φ contains either $U \in \mathsf{b}(A)$ or $U \notin \mathsf{b}(A)$.

Furthermore, we know that φ also contains either $U \in \mathsf{b}^-(A)$ or $U \notin \mathsf{b}^-(A)$ because it is closed under (P.distr.eq). □

Lemma 5.5 (Satisfiability of simple generated saturations). *A simple generated $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint is satisfiable.*

Proof. Let φ be a simple generated $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint. In Chapter 4 we have shown that any simple generated $\mathcal{P}_\mathrm{p}$-saturated $\mathcal{C}_\mathrm{pp}$-constraint is satisfiable (Lemma 4.16, p. 93). Now we proceed as follows: We construct a model for the maximal subset of φ that is a parallelism constraint, in the same way as in Lemma 4.16, and then we extend this model to a lambda structure satisfying φ. So let φ_p be the maximal subset of φ that is a parallelism constraint, and let (θ, σ) be a model for φ_p constructed as in the proof of Lemmas 3.8 (p. 65) and 4.16. We now extend θ to a lambda structure $\mathcal{L}^{\theta'}$ that is a model of φ.

We have to make sure that every var-labeled node possesses a binder. Suppose $S \subseteq \mathcal{V}ar(\varphi)$ is the set of var-labeled variables without a lambda binder in φ. We construct a new tree θ' by adding one lam-labeled node "above" θ: let $\theta' = \mathsf{lam}(\theta)$. Now we define the binding functions we are going to use in the model:

$$\lambda(\sigma(X)) = \begin{cases} \sigma(Y) & \text{if } \lambda(X){=}Y \text{ in } \varphi \\ \varepsilon & \text{if } X \in S \end{cases}$$

and

$$\mathsf{ante}(\sigma(X)) = \sigma(Y) \text{ if } \mathsf{ante}(X){=}Y \text{ in } \varphi$$

for all $X \in \mathcal{V}ar(\varphi)$. It remains to show that for $\mathcal{L}^{\theta'} = (\theta', \lambda, \mathsf{ante})$, $(\mathcal{L}^{\theta'}, \sigma)$ is indeed a model of φ.

The function λ is well-defined: Concerning the variables in $\mathcal{V}ar(\varphi) - S$ this follows by the closure of φ under (D.λ.func), and for the variables in S this is due to the fact that we map them all to the same binder ε. The function ante is well-defined by the closure of φ under (D.ante.func). Each node in the domain of λ is labeled var: For the nodes interpreting variables in $\mathcal{V}ar(\varphi) - S$ this is because φ is closed under (D.λ.var), and for the nodes denoting variables in S, this is true by the definition of S. Each node in the domain of ante is labeled ana by (D.ante.ana). Each node in the range of λ is labeled $\mathsf{lam}, \exists$ or $\forall$ by (D.λ.lam) and the way we have constructed θ' from θ. The function λ is total on the var-labeled nodes of θ by the way we have defined it. For each π in the domain of λ, $\lambda(\pi)$ is a prefix of π by (D.λ.dom) and the construction of λ.

It remains to show that the conditions of Def. 2.7 on the interactions of parallelism and binding are met. For a var- or ana-labeled variable that possesses a binder in φ, the rules (P.λ...) and (P.ante...) take care of this because by Lemma 5.4 we know, for each variable in φ and each segment term involved in a parallelism literal, whether the variable is inside the segment term or not. For a var-labeled node that is not bound in the constraint φ, the construction of the function λ makes sure that the conditions of Def. 2.7 are fulfilled: For all $A \sim B$ in φ, the lam-node we have newly introduced in θ' is outside of the segments that A, B denote. So we have not introduced any hanging binders, and if some var-labeled node of θ' is inside some parallelism segment and is bound at ε, then its correspondent is bound at ε too – if a var-labeled variable in φ is unbound, then all its correspondents are unbound too by closure under (P.λ.same) and (P.λ.out). ☐

5.3.2 Non-simple Constrains

Now we consider the case of non-simple $\mathcal{P}$-saturated constraints. We first show that given a non-simple $\mathcal{P}$-saturation containing an unlabeled variable, we can extend it by labeling that variable in such a way that the extension is still $\mathcal{P}$-saturated.

Lemma 5.6 (Extension by labeling). *Every $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint with an unlabeled variable U_0 can be extended to a $\mathcal{P}$-saturated constraint in which U_0 is labeled.*

Proof. Let $\{U_1, \ldots, U_m\}$ be a maximal φ-disjointness set in $\mathrm{con}_\varphi(U_0)$. Assume that Σ contains a function symbol f of arity m. (If it does not, then we can encode it using a nullary function symbol and a symbol of arity 2, as in Lemma 3.12 (p. 67).) We use the same definition of an extension $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ of $\varphi \wedge U_0{:}f(U_1, \ldots, U_m)$ as in Lemma 4.19

(p. 98). We repeat it here:

$$\mathsf{ext}_{U_0,\ldots,U_m}(\varphi) =_{\text{def}} \varphi \;\wedge \bigwedge_{\substack{V_0:f(V_1,\ldots,V_m)\in\\ \text{copy}_\varphi(U_0,U_1,\ldots,U_m)}} \Big(V_0{:}f(V_1,\ldots,V_m) \;\wedge\; \bigwedge_{i=1}^m V_0{\neq}V_i \;\wedge \bigwedge_{\substack{V_i\triangleleft^* Z, V_j\triangleleft^* W\in\varphi,\\ 1\leq i<j\leq n}} Z\bot W \;\wedge \bigwedge_{\substack{Z:g(\ldots)\in\varphi,\\ g\neq f \;\vee\; \mathsf{ar}(g)\neq\mathsf{ar}(f)}} Z{\neq}V_0\Big)$$

Without loss of generality we can assume that f is neither var, lam or ana: In Chapter 2 we have defined the signature Σ in such a way that it contains these symbols *in addition* to at least one nullary and one at least binary function symbol.

We show for each rule of $\mathcal{P}$, unless it is already in $\mathcal{P}_{\text{p}}$, that it is not applicable to $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$.

(D.λ...), (D.ante...) We have not added any var- or lam-labels. Also, we have not added any binding literals. Thus, none of these rules are applicable.

(P.λ.same), (P.λ.out): (P.λ.same) is $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^2 \text{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \to \lambda(V_1){=}V_2$, and (P.$\lambda$.out) is $\lambda(U_1){=}Y \wedge \text{co}(A,B)(U_1){=}V_1 \wedge U_1 \in \mathsf{b}^-(A) \wedge Y\triangleleft^+X_0 \to \lambda(V_1){=}Y$. We have not added any lambda binding or parallelism literals. Also, we cannot recently have acquired new information on whether a binder or a bound variable is situated inside a segment term: if we have $\lambda(W_1){=}W_2$ and $W_1 \in \mathsf{b}^-(A)$ in φ for some segment term $A = X_0/X_1,\ldots,X_n$ involved in a parallelism literal, then φ contains either $W_2 \in \mathsf{b}^-(A)$ or $W_2 \notin \mathsf{b}^-(A)$ by closure under (D.distr.notDisj), (P.distr.seg) and (P.distr.eq). And if $W_2 \in \mathsf{b}^-(A)$ is in φ then φ contains either $W_1 \in \mathsf{b}^-(A)$ or $W_1 \notin \mathsf{b}^-(A)$ by closure under (P.distr.seg) and (P.distr.eq).

(P.λ.hang): (P.λ.hang) is $\lambda(U_1){=}U_2 \wedge A \sim B \wedge U_2 \in \mathsf{b}^-(A) \wedge U_1{\notin}\mathsf{b}^-(A) \to \texttt{false}$. We have not added any new parallelism literals or lambda binding literals. Also, we have not introduced new dominance literals. So if $\lambda(W_1){=}W_2 \in \varphi$ and there is a segment term $A = X_0/X_1,\ldots,X_n$ with $X_0\triangleleft^*W_2 \in \varphi$, then φ contains either $W_1 \in \mathsf{b}^-(A)$ or $W_1 \notin \mathsf{b}^-(A)$ by closure under (P.distr.seg) and (P.distr.eq).

(P.ante.distr): (P.ante.distr) is $\mathsf{ante}(U_1){=}U_2 \wedge A \sim B \wedge U_1 \in \mathsf{b}^-(A) \to X_0\triangleleft^*U_2 \vee U_2\triangleleft^+X_0 \vee U_2\bot X_0$. We have not introduced any new parallelism or anaphoric binding literals. Also, we have not introduced new dominance literals. So if $\mathsf{ante}(W_1){=}W_2 \in \varphi$, and we have a segment term $A = X_0/X_1,\ldots,X_n$ such that $X_0\triangleleft^*W_1 \in \varphi$ already, then by the closure of φ under (P.distr.seg) and (P.distr.eq) φ contains either $W_1 \in \mathsf{b}^-(A)$ or $W_1 \notin \mathsf{b}^-(A)$.

(P.ante.same), (P.ante.out): (P.ante.same) is $\mathsf{ante}(U_1){=}U_2 \wedge \bigwedge_{i=1}^2 \text{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \to \mathsf{ante}(V_1){=}U_1 \vee \mathsf{ante}(V_1){=}V_2$, and (P.ante.out) is $\mathsf{ante}(U_1){=}U_2 \wedge \text{co}(A,B)(U_1){=}V_1 \wedge U_2{\notin}\mathsf{b}(A) \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \to \mathsf{ante}(V_1){=}U_2$.

We have not added any anaphoric binding literals. It remains to show that we have not recently acquired new information on whether an anaphoric binder or a ana-labeled variable is situated in a segment term involved in some parallelism. Suppose $A \sim B$ and $\mathsf{ante}(W_1){=}W_2$ are in φ, and $W_1 \in \mathsf{b}^-(A)$ is in $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$. Then $W_1 \in \mathsf{b}^-(A)$ is in φ already, as pointed out above, and by closure of φ under (P.ante.distr), either $W_2 \in \mathsf{b}^-(A)$ is in φ or $W_2 \notin \mathsf{b}^-(A)$ is. So either (P.ante.same) or (P.ante.out) has been applied to W_1 and W_2 in φ already.

□

By extending a non-simple $\mathcal{P}$-saturated constraint a finite number of times until all variables are labeled, we obtain a simple saturated constraint.

Proposition 5.7. *Every generated $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint can be extended to a simple generated $\mathcal{P}$-saturated constraint.*

Proof. As for Prop. 4.21 (p. 101). □

Hence, any generated $\mathcal{P}$-saturated constraint is satisfiable.

Lemma 5.8 (Satisfiability of generated saturations). *A generated $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint is satisfiable.*

Proof. By Lemma 5.5 and Prop. 5.7. □

5.4 Completeness of Procedure $\mathcal{P}$

Given a partial order $\preceq$ on constraints, a solver is complete with respect to $\preceq$ iff it computes all $\preceq$-minimal saturated constraints for a given constraint (Def. 3.16, p. 69). The family of partial orders that we use is $\leq_\mathcal{G}$, parametrized by a set $\mathcal{G} \subseteq \mathcal{V}ar$. We have introduced it in Sec. 4.4. It can be described as subset inclusion modulo α-renaming of the variables not in $\mathcal{G}$.

For the proof that $\mathcal{P}$ is complete, we lift the equivalence relations $=^{ex}_\mathcal{G}$ and the partial orders $\leq_\mathcal{G}$ from $\mathcal{C}_\mathrm{pp}$ to CLLS$_\mathrm{p}$. All lemmas of Sec. 4.4 still hold. The proof of completeness proceeds in two steps: First, given a constraint φ and a minimal saturated constraint ς for φ, any saturation rule that is applicable to φ can be applied in such a way that the result is "closer to" ς. This step is the same as in the previous chapter: Lemma 4.34 (p. 110) holds for $\mathcal{P}$ as well as for $\mathcal{P}_\mathrm{p}$.

Now, in the second step, we show that $\mathcal{P}$ can not only move closer to any given minimal saturation ς of a constraint φ, but that it can actually reach it. It turns out that the

arguments laid out in the proof of Lemma 4.38 for $\mathcal{P}_\mathrm{p}$ cover the case of $\mathcal{P}$ as well. We again make use of Def. 4.33 in formulating the following lemma:

Lemma 5.9 (Completeness). *Let φ be a CLLS$_\mathrm{p}$ constraint and $\mathcal{G} \subseteq \mathcal{V}ar(\varphi)$. Then $\mathcal{P}$ can compute from φ, in a finite number of steps, any minimal $\mathcal{P}$-saturation for φ with respect to $\mathcal{G}$.*

Proof. In Lemma 4.38 (p. 112) we have shown that the parallelism constraint $\mathcal{P}_\mathrm{p}$ can compute each minimal saturation of a given parallelism constraint in finite time. The only saturation rules that are explicitly used in the proof are (P.new) and (P.distr.eq).

In $\mathcal{P}$, there is one additional saturation rule that introduces new variables, namely (D.λ.lam). But this rule is applicable at most once per lambda binding literal. Hence, the proof of Lemma 4.38 covers the current lemma as well. □

It is interesting to note that for the proof of completeness the actual saturation rules play almost no role. The only rules that are mentioned are those that introduce additional variables – they are the ones that make the more complex proof via a distance measure necessary – and the projection rule (P.distr.eq), which for each additionally introduced variable enforces a choice between the introduction of a new node in the constraint graph, and identification with an already existent variable.

As before, each model of a constraint is also a model of one of its minimal saturated constraints. This holds by Prop. 4.39 (p. 113).

5.5 Recapitulation: Properties of the Procedure $\mathcal{P}$

In the previous sections, we have shown a number of properties of the procedure $\mathcal{P}$, which we now sum up.

Theorem 5.10. *The semi-decision procedure $\mathcal{P}$ for CLLS has the following properties:*

1. *It is sound for lambda structures.*
2. *There are unsatisfiable CLLS constraints for which it does not terminate.*
3. *A generated $\mathcal{P}$-saturated CLLS$_\mathrm{p}$-constraint is satisfiable.*
4. *$\mathcal{P}$ is complete: Given a CLLS constraint φ, $\mathcal{P}$ computes all minimal $\mathcal{P}$-saturations for φ.*
5. *This set of minimal $\mathcal{P}$-saturations for a CLLS constraint may be infinite.*

Proof. 1. by Lemma 5.3, 2. by Ex. 4.7, 3. by Lemma 5.8, 4. by Lemma 5.9, 5. by Ex. 4.6 and 4.8. □

5.6 All Rules of $\mathcal{P}$ Collected

In this section we list all saturation rules of the solver $\mathcal{P}$ for CLLS, collected from Fig. 3.2, 4.3 and 5.1.

Let $A = X_0/X_1, \ldots, X_n$ and $B = Y_0/Y_1, \ldots, Y_n$.

Rules of $\mathcal{P}_\mathrm{d}$

(D.clash.ineq)	$X{=}Y \wedge X{\neq}Y \to \mathtt{false}$
(D.clash.disj)	$X{\perp}X \to \mathtt{false}$
(D.dom.refl)	$\varphi \to X{\triangleleft^*}X$ where $X \in \mathcal{V}ar(\varphi)$
(D.dom.trans)	$X{\triangleleft^*}Y \wedge Y{\triangleleft^*}Z \to X{\triangleleft^*}Z$
(D.lab.decom)	$X{:}f(X_1,\ldots,X_n) \wedge Y{:}f(Y_1,\ldots,Y_n) \wedge X{=}Y \to \wedge_{i=1}^n X_i{=}Y_i$
(D.lab.ineq)	$X{:}f(\ldots) \wedge Y{:}g(\ldots) \to X{\neq}Y$ where $f \neq g$
(D.lab.disj)	$X{:}f(\ldots X_i,\ldots,X_j,\ldots) \to X_i{\perp}X_j$ where $1 \leq i < j \leq n$
(D.lab.dom)	$X{:}f(\ldots,Y,\ldots) \to X{\triangleleft^+}Y$
(D.disj)	$X{\perp}Y \wedge X{\triangleleft^*}X' \wedge Y{\triangleleft^*}Y' \to Y'{\perp}X'$
(D.distr.notDisj)	$X{\triangleleft^*}Z \wedge Y{\triangleleft^*}Z \to X{\triangleleft^*}Y \ \vee \ Y{\triangleleft^*}X$
(D.distr.child)	$X{\triangleleft^*}Y \wedge X{:}f(X_1,\ldots,X_n) \to Y{=}X \ \vee \ \bigvee_{i=1}^n X_i{\triangleleft^*}Y$

Additional Rules of $\mathcal{P}_\mathrm{p}$

(P.init)	$A{\sim}B \to \mathsf{seg}(A) \wedge \mathsf{seg}(B) \wedge \mathrm{co}(A,B)(X_i){=}Y_i$ where $0 \leq i \leq n$
(P.copy.dom)	$U_1RU_2 \wedge \bigwedge_{i=1}^2 \mathrm{co}(A,B)(U_i){=}V_i \to V_1RV_2$ where $R \in \{\triangleleft^*, \perp, \neq\}$
(P.copy.lab)	$U_0{:}f(U_1,\ldots,U_m) \wedge \bigwedge_{i=0}^m \mathrm{co}(A,B)(U_i){=}V_i \wedge U_0 \in \mathsf{b}^-(A) \to$ $V_0{:}f(V_1,\ldots,V_m)$
(P.new)	$A{\sim}^{\mathrm{sym}}B \wedge U \in \mathsf{b}(A) \to \exists U'.\mathrm{co}(A,B)(U){=}U'$ where U' is a fresh variable
(P.distr.seg)	$A{\sim}^{\mathrm{sym}}B \wedge X_0{\triangleleft^*}X \to X \in \mathsf{b}(A) \ \vee \ \bigvee_{j=1}^n X_j{\triangleleft^+}X$
(P.distr.eq)	$\varphi \to X{=}Y \ \vee \ X{\neq}Y$ where $X,Y \in \mathcal{V}ar(\varphi)$
(P.path.dom)	$\mathrm{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \to X{\triangleleft^*}U \wedge Y{\triangleleft^*}V$
(P.path.eq.1)	$\mathrm{p}\left(\begin{smallmatrix} X_1 & X_3 \\ X_2 & X_4 \end{smallmatrix}\right) \wedge \bigwedge_{i=1}^4 X_i{=}Y_i \to \mathrm{p}\left(\begin{smallmatrix} Y_1 & Y_3 \\ Y_2 & Y_4 \end{smallmatrix}\right)$

(P.path.eq.2)	$\mathrm{p}(\begin{smallmatrix} X & X \\ U & V \end{smallmatrix}) \to U{=}V$
(P.trans.h)	$\mathrm{p}(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} Y & Z \\ V & W \end{smallmatrix}) \to \mathrm{p}(\begin{smallmatrix} X & Z \\ U & W \end{smallmatrix})$
(P.trans.v)	$\mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}) \to \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix})$
(P.diff.1)	$\mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}) \wedge X_2 \triangleleft^* X_3 \wedge Y_2 \triangleleft^* Y_3 \to \mathrm{p}(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix})$
(P.diff.2)	$\mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}) \wedge \mathrm{p}(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}) \wedge X_1 \triangleleft^* X_2 \wedge Y_1 \triangleleft^* Y_2 \to \mathrm{p}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})$

Additional Rules of $\mathcal{P}$

(D.λ.func)	$\lambda(X){=}Y \wedge \lambda(U){=}V \wedge X{=}U \to Y{=}V$
(D.λ.dom)	$\lambda(X){=}Y \to Y \triangleleft^* X$
(D.λ.var)	$\lambda(X){=}Y \to X{:}var$
(D.λ.lam)	$\lambda(X){=}Y \to \exists Y'.(Y{:}\mathsf{lam}(Y') \vee Y{:}\forall(Y') \vee Y{:}\exists(Y'))$
(D.ante.func)	$\mathsf{ante}(X){=}Y \wedge \mathsf{ante}(U){=}V \wedge X{=}U \to Y{=}V$
(D.ante.ana)	$\mathsf{ante}(X){=}Y \to X{:}\mathsf{ana}$

(P.λ.same)	$\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \to \lambda(V_1){=}V_2$
(P.λ.out)	$\lambda(U_1){=}Y \wedge \mathrm{co}(A,B)(U_1){=}V_1 \wedge U_1 \in \mathsf{b}^-(A) \wedge Y \triangleleft^+ X_0 \to \lambda(V_1){=}Y$
(P.λ.hang)	$\lambda(U_1){=}U_2 \wedge A \sim B \wedge U_2 \in \mathsf{b}^-(A) \wedge U_1 \notin \mathsf{b}^-(A) \to \mathtt{false}$
(P.ante.same)	$\mathsf{ante}(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}(A,B)(U_i){=}V_i \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \to \mathsf{ante}(V_1){=}U_1 \vee \mathsf{ante}(V_1){=}V_2$
(P.ante.out)	$\mathsf{ante}(U_1){=}U_2 \wedge \mathrm{co}(A,B)(U_1){=}V_1 \wedge U_2 \notin \mathsf{b}(A) \wedge U_1 \in \mathsf{b}^-(A) \wedge A \sim B \to \mathsf{ante}(V_1){=}U_2$
(P.ante.distr)	$\mathsf{ante}(U_1){=}U_2 \wedge A \sim B \wedge U_1 \in \mathsf{b}^-(A) \to X_0 \triangleleft^* U_2 \vee U_2 \triangleleft^+ X_0 \vee U_2 \perp X_0$

5.7 Summary

In this chapter we have completed the presentation of the semi-decision procedure $\mathcal{P}$ for CLLS. We have extended the semi-decision procedure $\mathcal{C}_\mathrm{p}$ for parallelism constraints by saturation rules for lambda and anaphoric binding. These rules implement the definitions of lambda and anaphoric binding of Chapter 2, in particular the conditions that govern the interaction of binding and parallelism.

While it is not hard to formulate a semi-decision procedure for CLLS constraints (just enumerate lambda structures and check for each if it satisfies the given constraint), the procedure $\mathcal{P}$ has the following properties:

- It terminates for the linguistically relevant constraints. For these constraints it computes saturations that correspond to the correct readings.

- It introduces *correspondence formulas* as a data structure for handling parallelism within partial tree descriptions.
- It includes an algorithm for solving dominance constraints. Given a dominance constraint as an input, the procedure behaves exactly like the dominance constraint solver that it encompasses. This is advantageous because, as we have seen in Chapter 2, dominance constraints play an important role in the linguistic application.
- It is built in a modular fashion: a different dominance constraint solver can be substituted for the one we use here. For example, the saturation algorithm of Duchier and Niehren [34], which needs less distribution, can be employed. Actually, a recent overview paper on processing CLLS [44] combines this latter dominance constraint solver with the rules for parallelism that we present in this chapter.

A central notion in the procedure $\mathcal{P}$ is that of a *correspondence formula*: Such a formula states that the denotations of two variables correspond with respect to a certain parallelism relationship. Correspondence formulas enable us to handle parallelism in a framework of underspecified descriptions of lambda structures: Even though the position of the nodes interpreting these two variables may not be completely determined, the correspondence formula states that the two nodes must be situated at corresponding positions in the two parallel segments. And the procedure can test the satisfiability of a correspondence formula by copying all constraints that concern one of the two variables to the other variable.

The procedure $\mathcal{P}$ is a saturation procedure that extends a set of clauses until nothing new can be added anymore. It is sound, i.e. all its rules are equivalence transformations. All the saturations that it computes are satisfiable: We have shown how to construct a model from a given saturated constraint. The procedure is also complete in the sense that it computes all minimal saturated constraints for a given constraint. We have defined minimality via a family $\leq_{\mathcal{G}}$ of partial orders (parametrized by a set $\mathcal{G} \subseteq \mathcal{V}ar$ of variables). These partial orders can be described as subset inclusion modulo α-renaming of the variables not in $\mathcal{G}$; alternatively we could say that they identify all variables that constitute the same node in the constraint graph. Interestingly, the proof of completeness for the procedure $\mathcal{P}$ is the same as for the procedure $\mathcal{P}_{\mathrm{p}}$, i.e. the proof is almost independent of the set of saturation rules used.

Part II

Applying Parallelism Constraints

Chapter 6

Underspecified Beta Reduction

The parallelism relation was originally introduced to model parallelism phenomena in linguistics. But it also allows a declarative description of rewriting steps on lambda terms, and more generally rewriting of trees. This surprising fact was first discovered in the context of *underspecified beta reduction*.

Underspecified beta reduction is a question that arises naturally when we study CLLS constraints, partial descriptions of lambda terms: Can we lift beta reduction from lambda terms to partial descriptions? By beta reducing a description, we could reduce all lambda terms that it describes at the same time. But the problem turns out not to be so easy: The simplest approach, graph rewriting, fails. The reason is that, given a CLLS constraint with a redex that we want to reduce, we may not yet know where in the lambda term some material will end up being – for example, it may or it may not be part of the redex' argument.

The solution to this problem is to reduce it to solving parallelism constraints [12]: Basically, if we take a lambda term and perform a beta reduction step, the result consists of segments of the term we had before, arranged in a different way. So if we view the original term and the resulting term as parts of the same bigger lambda structure, we can relate segments of both terms by parallelism. That is, the idea is to give a declarative description of the result of a single beta reduction step.

In the current chapter, we are going to discuss the following issues:

- We give the definitions of two additional relations on nodes of a lambda structure: The relation between segments of the lambda terms before and after a beta reduction step is described by the *beta reduction relation*, which in turn can be expressed using the *group parallelism relation* [11, 12]. Group parallelism is a generalization of parallelism, the only difference being the conditions on lambda and anaphoric binding.
- We generalize the semi-decision procedure $\mathcal{P}$ for CLLS such that it also handles *group parallelism literals* and *inverse lambda binding literals*. Extended in this way, the procedure can compute the result of a single underspecified beta reduction step.
- However, we would like to keep the lambda term description "as underspecified as it was" while we perform a beta reduction step. The procedure $\mathcal{P}$ may disambiguate

too much for that purpose. For example, given a constraint from the linguistic application, $\mathcal{P}$ resolves all scope ambiguities – while ideally ambiguity resolution and beta reduction should be kept independent.

We present a variant of the above procedure that in many cases can perform an underspecified beta reduction step without disambiguation. The idea is to exploit the fact that we know the relative positions of the segments that stand in the beta reduction relation. In this procedure we employ *underspecified correspondence formulas.*

- Finally, we show how either of the two procedures for a single beta reduction step can be integrated into a procedure for underspecified beta reduction (which can perform more than one beta reduction step in a row).

6.1 The Problem of Underspecified Beta Reduction

The problem of underspecified beta reduction is the following:

> Given an underspecified description of some set of higher-order lambda terms (in the form of a CLLS constraint), compute an underspecified description of all first-order formulas that can be derived from that set by beta reduction.

Of course, we would like to do this without enumerating readings inbetween.

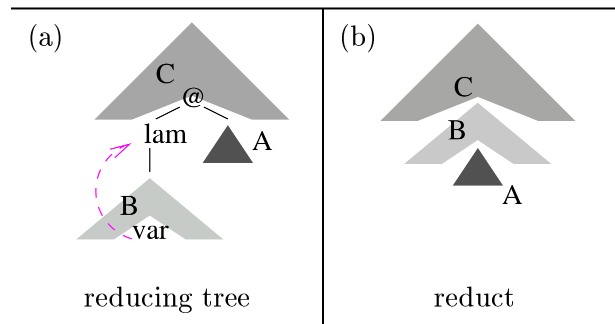


Figure 6.1: Beta reduction on lambda structures – abstract schema

We first take a look at "normal" beta reduction. Consider Fig. 6.1, a sketch of two lambda structures. The sketch in (a) is the *reducing tree*, the lambda structure to which beta reduction is applied. One beta reduction step yields the *reduct* in picture (b). The *redex* of the reducing tree starts at the @-labeled node. It contains the *body* B as well as the *argument* A. The beta reduction step replaces all occurrences of the *object-level variable* shown in the picture by occurrences of the argument. The rest of the term around the redex, the *context* C, remains unchanged.

Figure 6.2 shows an instance of this abstract schema. Again, picture (a) contains the reducing tree and (b) the reduct. Redex, context, body and argument in the reducing tree

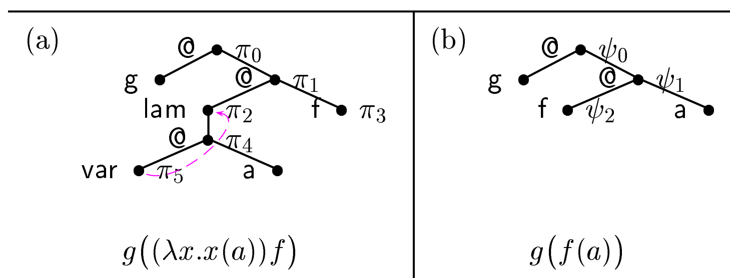


Figure 6.2: A beta reduction step performed on the lambda structure in (a) produces (b).

are segments: The redex has the form $\pi_1/$, the context is the segment π_0/π_1, the body is the segment π_4/π_5 and the argument is $\pi_3/$.

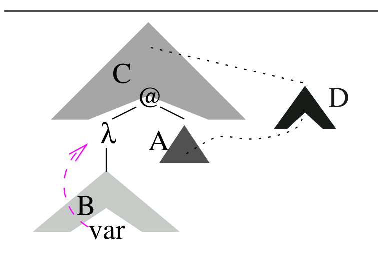


Figure 6.3: An underspecified description of a reducing tree

Up to now we have just considered lambda structures. If we move on to partial descriptions in the form of CLLS constraints, what changes? Consider Fig. 6.3, a sketch of a CLLS constraint. Again, C, B, A are context, body and argument of the reducing tree. D is some segment term that is dominated by some variable in C and dominates some variable in A, but that is all that is known about it. So we do not know whether D belongs to the context C or the argument A of the reducing tree – can we still compute the reduct without disambiguating the position of D first? The situation sketched in Fig. 6.3 is one that we will be discussing frequently in this chapter. So in reference to this figure we will informally refer to a segment term in a position that is underspecified between context, argument and body of a reducing tree as a D *segment term.*

This particular situation is actually a quite common one in the linguistic application. It occurs, for example, in the constraint in Fig. 6.4 (a), which represents the meaning of the sentence “Every student does not pay attention”. This sentence contains a scope ambiguity between “every student” and ”not”. In the reading with “every student” taking wide scope, it states that of all students it is true that their minds are wandering. In the reading with “not” taking wide scope, the sentence says that it is not the case that all students pay attention.[1] The constraint in Fig. 6.4 (a) reflects this scope ambiguity.

[1] Some speakers of English consider the use of “not” in this second fashion to be very colloquial; however, they do not judge it as ungrammatical.

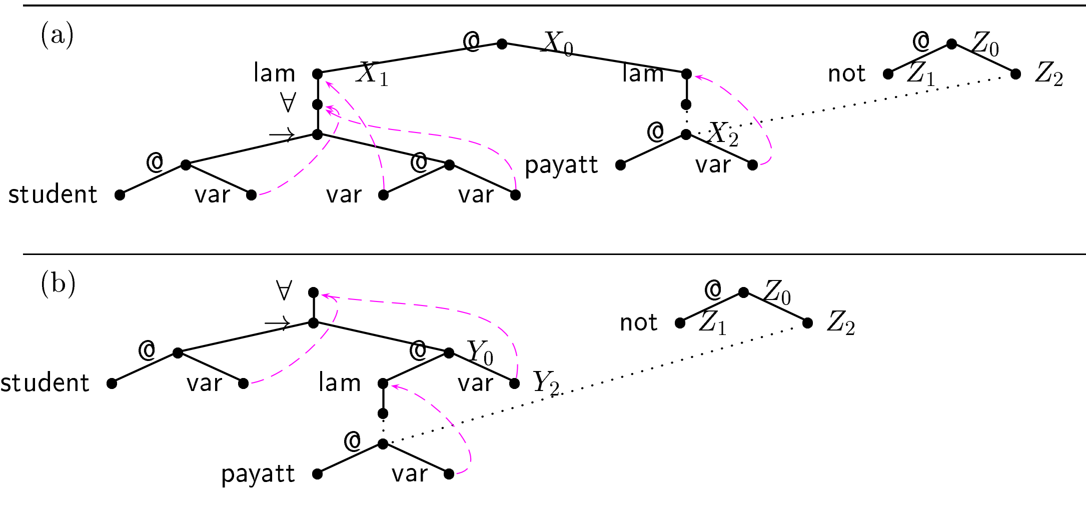


Figure 6.4: A simple beta reduction step: "Every student does not pay attention"

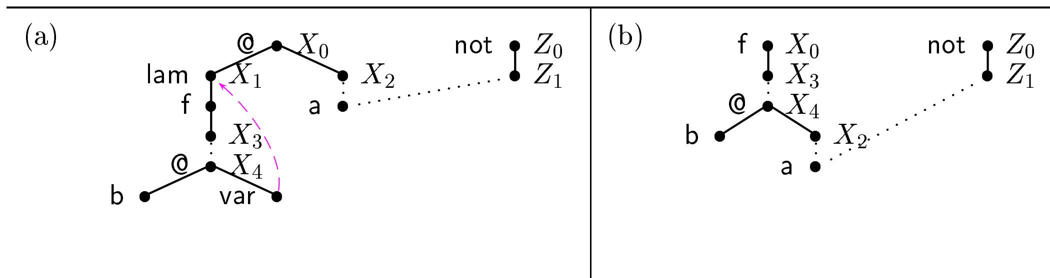


Figure 6.5: The graph rewriting approach fails

In the constraint there is a redex starting at X_0, within the representation for "every student". With respect to this redex, the representation of "not" (consisting of Z_0, Z_1, Z_2) is a D segment term (as in Fig. 6.3). It dominates X_2 in the argument, but it could also be part of the context above X_0. But in this case, it is not hard to write down an underspecified description of the reduct: it is the one in Fig. 6.4 (b). In this constraint, the object-level variable (the one bound by the lambda binder X_1 in (a)) has just been replaced by the argument.

For the example we have just seen, we can compute the result of an underspecified beta reduction step with a simple graph rewriting approach. But that is not always the case. Consider Fig. 6.5, which again shows two constraints. Picture (a) is a reducing tree, and picture (b) shows what the graph rewriting approach yields as the reduct. But this reduct is wrong, it has too many solutions: In the reducing tree in (a), there is again a D segment term. In this case, it consists of Z_0 and Z_1, and it may be either above X_0 or below X_2. However, in the reduct in (b) there are three possible positions for the D segment term: It may additionally be placed inbetween X_3 and X_4. So one of the solutions of the constraint in (b) is $f(\text{not}(b(a)))$, which cannot be obtained by reducing

any of the terms described by (a). This means that the naive graph rewriting approach is unsound; the ensuing descriptions may have too many solutions. In performing the destructive graph rewriting step, we have lost important information that was present in the reducing tree: the fact that the D segment term cannot go between X_3 and X_4.

There is an alternative approach that is sound: modeling the result of a beta reduction step by parallelism. Take another look at Fig. 6.1: If we compare the reducing tree (a) and the reduct (b), we see that the context, body and argument segment of the reducing tree all reappear in the reduct; the context and body segments reappear exactly once, and the argument appears in the reduct as often as there are occurrences of the bound object-level variable in the reducing tree.

So we regard reducing tree and reduct as parts of the same big lambda structure, and we relate the context in reducing tree and reduct by parallelism, likewise the two bodies, and the argument in the reducing tree with each argument copy in the reduct. In this way, we keep all the information of the reducing tree.

How does that help us with constraints like the one in Fig. 6.5 (a), where the simple rewriting approach resulted in too many solutions for the reduct in (b)? We now have one big constraint which includes both the reducing tree constraint in Fig. 6.5 (a) and the reduct constraint in (b) as subconstraints. The two subconstraints are related by parallelism literals. This will exclude the wrong solution that we got for the reduct constraint in Fig. 6.5 (b) on its own: In any solution of the constraint (a), the "not" fragment must go either above X_0 or below X_2, and so, by the isomorphic structure that the parallelism imposes, the "not" fragment in (b) must be either above X_0 or below X_2.

6.2 Beta Reduction and Group Parallelism

In this section we introduce the *beta reduction relation* [11, 12], which, inside a single lambda structure, relates the segments belonging to the reducing tree and the segments belonging to the reduct. Furthermore we introduce the *group parallelism relation*, a generalization of the parallelism relation (the only difference being in the conditions on binding), with which we can express the beta reduction relation. For both relations, we add matching new *literals* to the language CLLS. We also add *inverse lambda binding literals*, which specify the set of all variables bound by a binder.

6.2.1 The Beta Reduction Relation

First we make our notions of a reducing tree and a reduct precise.

Definition 6.1 (Reducing tree, reductlike). *Let $\mathcal{L}^\theta$ be a lambda structure.*

- *A* reducing tree *in $\mathcal{L}^\theta$ is a sequence (γ, β, α) of segments of $\mathcal{L}^\theta$ such that there exists nodes π_0, π_1 of $\mathcal{L}^\theta$ with the following properties.*

  $hs(\gamma) = \pi_0$, $\pi_0{:}@(\pi_1, r(\alpha))$, $\pi_1{:}\mathsf{lam}(r(\beta))$, *and* $\lambda^{-1}(\pi_1) = \{hs(\beta)\}$.

- *We call a sequence of segments* $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$ *of* $\mathcal{L}^\theta$ reductlike *iff* $hs(\gamma') = r(\beta')$, *and* $r(\alpha'_i)$ *is the ith hole of* β' *for all* $1 \leq i \leq n$.

Recall that $hs(\alpha)$ is the sequence of holes of the segment α, ordered from left to right. Also, λ^{-1} is the inverse of the lambda binding function of Def. 2.6 (p. 29).

In a reducing tree the hole of the context segment γ, the node π_0, must be labeled @. Its left child, π_1, is labeled lam, and its right child is the root of the argument segment α. Concerning the lambda binder π_1, its child must be the root of the body segment β, and the var-nodes that π_1 binds must be exactly the holes of the body segment β.

For a sequence of segments to be reductlike, it must contain one "context segment" γ' directly "on top" of a "body segment" β', and the holes of β' must be the roots of the α'_i. Note that not every reductlike segment sequence is a potential reduct: If there is a binder from the argument into the body, the sequence cannot be a reduct because that would violate the freeness condition of beta reduction.

Now, using the notions of a reducing tree and a reductlike segment sequence, we can define the beta reduction relation on sequences of segments.[2]

Definition 6.2 (Beta reduction relation). *Let* $\mathcal{L}^\theta$ *be a lambda structure. Then the* beta reduction relation $\rightarrow^\beta$ *is a relation on sequences of segments of* $\mathcal{L}^\theta$, *defined as follows:*

$$(\gamma, \beta, \alpha) \rightarrow^\beta (\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$$

holds in $\mathcal{L}^\theta$ *iff first,* (γ, β, α) *form a reducing tree and* $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$ *are reductlike. Second, there are correspondence functions* c_γ *between* γ, γ', c_β *between* β, β' *and* c^i_α *between* α, α'_i *(for* $1 \leq i \leq n$*), such that for each* δ, δ' *among these segment pairs with correspondence function* c *between them and for each* $\pi \in \mathsf{b}^-(\delta)$, *the following conditions hold:*

($\boldsymbol{\beta.\lambda}$.same) *For a* var*-labeled node bound in the same segment, the correspondent is bound by the c-corresponding binder node.*

$$\big(\lambda(\pi) \in \mathsf{b}^-(\delta) \;\vee\; \lambda(c(\pi)) \in \mathsf{b}^-(\delta')\big) \Rightarrow \lambda(c(\pi)) = c(\lambda(\pi))$$

($\boldsymbol{\beta.\lambda}$.diff) *For a* var*-labeled node bound in a different segment* ϵ, *the correspondent is bound by the* c_ϵ*-corresponding binder node.*

$$\forall (\epsilon, \epsilon', c_\epsilon) \in \{(\gamma, \gamma', c_\gamma), (\beta, \beta', c_\beta)\}$$
$$\big(\lambda(\pi) \in \mathsf{b}^-(\epsilon) \;\vee\; \lambda(c(\pi)) \in \mathsf{b}^-(\epsilon')\big) \Rightarrow \lambda(c(\pi)) = c_\epsilon(\lambda(\pi))$$

[2] In earlier texts by Bodirsky [11] and Bodirsky, Erk, Koller and Niehren [12] the conditions on lambda binding in the beta reduction relation are not symmetric. But these nonsymmetric conditions are not strong enough in cases where the reducing tree segments and the reduct segments overlap: They do not force lambda binding to behave exactly as in group parallelism.

($\beta.\lambda$.out) *For a* var*-labeled node bound above the reducing tree, the corresponding node is bound at the same place:*

$$\big(\lambda(\pi) \notin \mathsf{b}(r(\gamma)/) \ \vee \ \lambda(c(\pi)) \notin \mathsf{b}(r(\gamma')/)\big) \Rightarrow \lambda(c(\pi)) = \lambda(\pi)$$

(β.ante.same) *For an* ana*-labeled node bound in the same segment, the correspondent has two possible antecedents, matching the strict and the sloppy reading:*

$$\mathsf{ante}(\pi) \in \mathsf{b}(\delta) \Rightarrow \mathsf{ante}(c(\pi)){=}\pi \ \vee \ \mathsf{ante}(c(\pi)){=}c(\mathsf{ante}(\pi))$$

(β.ante.diff) *For an* ana*-labeled node bound inside another segment ϵ of the reducing tree, there are again two possible antecedents, only this time with respect to c_ϵ:*

$$\forall(\epsilon, c_\epsilon) \in \{(\gamma, c_\gamma), (\beta, c_\beta)\} \cup \{(\alpha, c^i_\alpha) \mid 1 \le i \le n\}$$
$$\mathsf{ante}(\pi) \in \mathsf{b}(\epsilon) \Rightarrow \mathsf{ante}(c(\pi)){=}\pi \ \vee \ \mathsf{ante}(c(\pi)){=}c_\epsilon(\mathsf{ante}(\pi))$$

(β.ante.out) *If an* ana*-labeled node is bound outside the reducing tree, then its correspondent has the same anaphoric binder:*

$$\mathsf{ante}(\pi) \notin (\mathsf{b}(\gamma) \cup \mathsf{b}(\beta) \cup \mathsf{b}(\alpha)) \Rightarrow \mathsf{ante}(c(\pi)) = \pi$$

The beta reduction relation on lambda structures models beta reduction on lambda terms faithfully. This even holds for lambda terms with global variables, although lambda structures can only model closed lambda terms. Global variables correspond to var-labeled nodes that are bound in the surrounding tree, i.e. above the reducing tree. Condition ($\beta.\lambda$.out) of Def. 6.2 thus ensures a proper treatment of global variables.

The definition of the beta reduction relation consists of two conditions:

- First, the segments concerned need to have the right relative positions: we must have a reducing tree and a reductlike sequence as defined above.
- Second, the segments in the reducing tree and the reduct must be parallel, except that the conditions on binding are less strict than in ordinary parallelism. Specifically, a lambda binder from the body (or argument) to the context of the reducing tree has to parallel a lambda binder from the body (or argument, respectively) to the context of the reduct.

This second condition can be cast in a more general form, a *group parallelism relation*.

6.2.2 The Group Parallelism Relation

Group parallelism relates a group (a sequence) of segments to another group of segments, specifying parallelism between each segment in the left group and its counterpart in the right one. The difference between one group parallelism and several "normal" parallelisms is that in group parallelism the restrictions on binding are more liberal.

Like in the case of the parallelism relation (Def. 2.7, p. 29), we define the group parallelism relation $\sim$ in two steps: First we define a symmetric relation $\sim_\lambda$ that describes conditions on lambda binding, then we define $\sim$ as a non-symmetric subrelation of $\sim_\lambda$.

Definition 6.3 (Group parallelism relation). *The relation $\sim_\lambda$ of a lambda structure $\mathcal{L}^\theta$ is the largest* symmetric *relation between equal-size groups (sequences) of segments of $\mathcal{L}^\theta$ such that $(\alpha_1, \ldots, \alpha_n) \sim_\lambda (\beta_1, \ldots, \beta_n)$ implies there are correspondence functions $c_k : \mathsf{b}(\alpha_k) \to \mathsf{b}(\beta_k)$ for all $1 \leq k \leq n$ that satisfy the following properties for all $1 \leq i, j \leq n$ and $\pi \in \mathsf{b}^-(\alpha_i)$:*

(gp.λ.same) *For a* var*-labeled node bound in the same segment, the corresponding node is bound correspondingly:*

$$\lambda(\pi) \in \mathsf{b}^-(\alpha_i) \Rightarrow \lambda(c_i(\pi)) = c_i(\lambda(\pi))$$

(gp.λ.diff) *For a* var*-labeled node bound outside α_i but inside α_j, the correspondent is bound at the corresponding place with respect to c_j:*

$$\lambda(\pi) \in \mathsf{b}^-(\alpha_j) \wedge \lambda(\pi) \notin \mathsf{b}^-(\alpha_i) \Rightarrow \lambda(c_i(\pi)) = c_j(\lambda(\pi))$$

(gp.λ.out) *Corresponding* var*-labeled nodes with binders outside the group segments are bound by the same binder:*

$$\lambda(\pi) \notin \bigcup_{k=1}^{n} \mathsf{b}^-(\alpha_k) \Rightarrow \lambda(c_i(\pi)) = \lambda(\pi)$$

(gp.λ.hang) *There are no hanging binders:*

$$\lambda^{-1}(\pi) \subseteq \bigcup_{k=1} \mathsf{b}^-(\alpha_k)$$

The group parallelism relation $\sim$ *of a lambda structure $\mathcal{L}^\theta$ is the largest relation between equal-size groups (sequences) of segments of $\mathcal{L}^\theta$ such that*

$$(\alpha_1, \ldots, \alpha_n) \sim (\beta_1, \ldots, \beta_n)$$

implies $(\alpha_1, \ldots, \alpha_n) \sim_\lambda (\beta_1, \ldots, \beta_n)$, and the correspondence functions $c_k : \mathsf{b}(\alpha_k) \to \mathsf{b}(\beta_k)$, $1 \leq k \leq n$, satisfy the following properties for all $1 \leq i, j \leq n$ and $\pi \in \mathsf{b}^-(\alpha_i)$:

(gp.ante.same) *For an* ana*-node bound within the segment, the correspondent has two possible antecedents, matching the strict and the sloppy reading:*

$$\mathsf{ante}(\pi) \in \mathsf{b}(\alpha_i) \Rightarrow \mathsf{ante}(c_i(\pi)) = \pi \;\vee\; \mathsf{ante}(c_i(\pi)) = c_i(\mathsf{ante}(\pi))$$

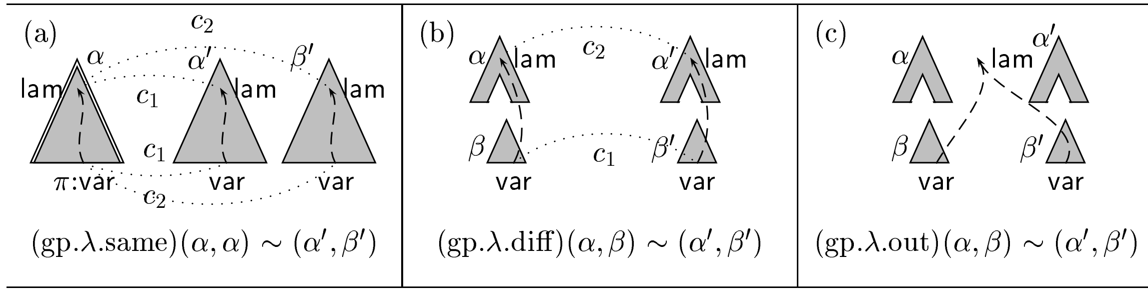


Figure 6.6: Possible bindings in a group parallelism.

(gp.ante.diff) *For an* ana*-node bound outside α_i but inside α_j, there are again two possible antecedents, only this time with respect to c_j:*

$$\mathsf{ante}(\pi) \in \mathsf{b}(\alpha_j) \wedge \mathsf{ante}(\pi) \notin \mathsf{b}(\alpha_i) \Rightarrow \mathsf{ante}(c_i(\pi)){=}\pi \;\vee\; \mathsf{ante}(c_i(\pi)){=}c_j(\mathsf{ante}(\pi))$$

(gp.ante.out) *If an* ana*-node is bound outside the group, then its correspondent has the same anaphoric binder:*

$$\mathsf{ante}(\pi) \notin \bigcup_{k=1}^{n} \mathsf{b}(\alpha_k) \Rightarrow \mathsf{ante}(c_i(\pi)) = \pi$$

Again, as in Def. 2.7 (p. 29), the conditions on lambda binding are symmetric, but the conditions on anaphoric binding are not.

The first three conditions are illustrated in Fig. 6.6. Condition (gp.λ.diff) is the main difference between parallelism and group parallelism. It allows lambda binding from one segment to another of the same group, provided that there is a parallel binder in the other group. If the segment pairs α, α' and β, β' of picture (b) were related by ordinary parallelism, the bound node in β would be bound outside the segment β, thus the (λ.out) condition would apply, and the corresponding node would have to be bound by the *same* binder. Ordinary parallelism is now simply a special case of group parallelism, with groups of size one.

Another interesting observation in Fig. 6.6 is that the conditions (gp.λ.same) and (gp.λ.diff) must be mutually exclusive. If (gp.λ.diff) was applicable in picture (a), it would enforce $\lambda(c_1(\pi)) = c_2(\lambda(\pi))$, which is clearly wrong.

6.2.3 Beta Reduction Literals and Group Parallelism Literals

Beta reduction literals are interpreted by the beta reduction relation. They have the form

$$(C, B, A) \rightarrow^{\beta} (C', B', A'_1, \ldots, A'_n)$$

for segment terms $C, B, A, C', B', A'_1, \ldots, A'_n$. *Group parallelism literals* are interpreted by the group parallelism relation. They take the form

$$(A_1, \ldots, A_m) \quad \sim \quad (B_1, \ldots, B_m).$$

for segment terms $A_1, \ldots A_m, B_1, \ldots, B_m$. *Inverse lambda binding literals* of the form

$$\lambda^{-1}(Y)=\{X_1, \ldots, X_m\}$$

for $m \geq 1$ are interpreted by the inverse of the lambda binding function. Such a literal is true in $((\theta, \lambda, \mathsf{ante}), \sigma)$ iff $\lambda^{-1}(\sigma(Y)) = \sigma(\{X_1, \ldots, X_n\})$. That is, $X_1, \ldots, X_n$ are *all* occurrences of the object-level variable bound by Y. The variables $X_1, \ldots, X_n$ need not denote distinct nodes.

We are going to extend the language CLLS not by all three types of literals but only by group parallelism and inverse lambda binding literals, because they can already express beta reduction literals. The formula that expresses a beta reduction literal makes use of formulas that state that a sequence of segments forms a reducing tree or is reductlike.

Let $C = X_0/X_1$, $B = Y_0/Y_1, \ldots, Y_n$ and $A = X_3/$, then

$$\begin{aligned}\mathrm{redtree}_{X_2}(C, B, A) \;=_{\mathrm{def}}\; & \mathsf{seg}(A) \;\wedge\; \mathsf{seg}(B) \;\wedge\; \mathsf{seg}(C) \\ & \wedge\; X_1{:}@(X_2, X_3) \;\wedge\; X_2{:}\mathsf{lam}(Y_0) \\ & \wedge\; \lambda^{-1}(X_2) = \{Y_1, \ldots, Y_n\}\end{aligned}$$

This formula states that the interpretations of the segment terms C, B, A form a reducing tree. Recall that we have defined $\mathsf{seg}(A)$, for segment terms $A = X_0/X_1, \ldots, X_n$, as $\mathsf{seg}(A) = \bigwedge_{i=1}^{n} X_0 \triangleleft^* X_i \wedge \bigwedge_{1 \leq i < j \leq n} \big((X_i \bot X_j) \vee (X_i{=}X_j)\big)$.

We need an inverse lambda binding literal in this formula because the form of the reduct depends in part on the number of occurrences of the bound object-level variable – it determines the number of copies of the argument segment in the reduct. And in an underspecified description of a reducing tree, the only way of knowing how many times the object-level variable occurs is to have an inverse lambda binding literal, because it explicitly states that we have collected *all* var-variables bound by this lambda binder.

Now we give a formula stating that the interpretations of a sequence of segment terms form a reductlike segment sequence. Let $C = X_0/X_1$, $B = Y_0/Y_1, \ldots, Y_n$ and $A_i = Z_i/$ for $1 \leq i \leq n$, then

$$\begin{aligned}\mathrm{reductlike}(C, B, A_1, \ldots, A_n) \;=_{\mathrm{def}}\; & \mathsf{seg}(A_1) \wedge \cdots \wedge \mathsf{seg}(A_n) \wedge \mathsf{seg}(B) \wedge \mathsf{seg}(C) \\ & \wedge X_1{=}Y_0 \wedge \bigwedge_{i=1}^{n} Y_i{=}Z_i\end{aligned}$$

Then we can express beta reduction literals as follows:

Lemma 6.4 (Expressing beta reduction literals with group parallelism literals).

$$\begin{aligned}(C, B, A) \rightarrow^\beta (C', B', A'_1, \dots, A'_n) \quad \models\!\!\!\models \quad & \exists X_2.\text{redtree}_{X_2}(C, B, A) \\ & \wedge\ (C, B, A, \dots, A) \sim (C', B', A'_1, \dots, A'_n) \\ & \wedge\ \text{reductlike}(C', B', A'_1, \dots, A'_n)\end{aligned}$$

Proof. We will check the two-side entailment separately, first from right to left. Let σ be a variable assignment into some lambda structure that solves the right hand side. Then the conditions (gp.λ.same), (gp.λ.diff), (gp.λ.out), (gp.ante.same), (gp.ante.diff), and (gp.ante.out) of group parallelism (Def. 6.3) subsume the corresponding conditions on the beta reduction relation (Def. 6.2).

For the other direction, let $(\mathcal{L}^\theta, \sigma')$ solve the beta-reduction literal on the left hand side, with $\sigma'(C) = \gamma$, $\sigma'(B) = \beta$, $\sigma'(A) = \alpha$, $\sigma'(C') = \gamma'$, $\sigma'(B') = \beta'$, and $\sigma'(A'_i) = \alpha'_i$ for $1 \leq i \leq n$. (Remember that we have extended valuations from variables to segment terms in Sec. 4.3.1, p. 91.) Then by Def. 6.2, (γ, β, α) must form a reducing tree in $\mathcal{L}^\theta$ with nodes π_0, π_1 as in Def. 6.1. Let σ be the variable assignment $\sigma'[\pi_0/X_0, \pi_1/X_1]$. It remains to check that $(\mathcal{L}^\theta, \sigma)$ solves the group parallelism literal on the right hand side.

We consider the relation $\approx$ which relates the group $(\gamma, \beta, \alpha, \dots, \alpha)$ to $(\gamma', \beta', \alpha'_1, \dots, \alpha'_n)$. We show that $\approx$ satisfies all conditions in the definition of group parallelism (Def. 6.3), which means that $\approx$ is subsumed by the group parallelism relation $\sim$. By Def. 6.2 there exist correspondence functions c_γ between the segments γ and γ', c_β between β and β', and c^i_α between α and α'_i for $1 \leq i \leq n$. By the definition of group parallelism (Def. 6.3), we have to check the conditions (gp.λ.same), (gp.λ.diff), and (gp.λ.out) both for the correspondence functions and their inverse functions. For the rest of the proof, let $(\delta, \delta') \in \{(\gamma, \gamma'), (\beta, \beta')\} \cup \{(\alpha, \alpha'_i) \mid 1 \leq i \leq n\}$, let $S = \mathsf{b}^-(\gamma) \cup \mathsf{b}^-(\beta) \cup \mathsf{b}^-(\alpha)$, and let $S' = \mathsf{b}^-(\gamma') \cup \mathsf{b}^-(\beta') \cup \bigcup_{i=1}^n \mathsf{b}^-(\alpha'_i)$.

(gp.λ.same): If π is a var-labeled node in $\mathsf{b}^-(\delta)$ that is bound within the same segment (i.e. $\lambda(\pi) \in \mathsf{b}^-(\delta)$), then its correspondent $c_\delta(\pi)$ must have its lambda binder within $\mathsf{b}^-(\delta')$ by (β.λ.same).

Likewise, if π' is a var-labeled node in $\mathsf{b}^-(\delta')$ that is bound in the same segment, that is, $\lambda(\pi') \in \mathsf{b}^-(\delta')$, then its correspondent $c_\delta^{-1}(\pi')$ must be lambda bound in $\mathsf{b}^-(\delta)$ by (β.λ.same).

(gp.λ.diff): Suppose π is a var-labeled node in $\mathsf{b}^-(\delta)$ that is bound in a different segment. Then we must have $\lambda(\pi) \in \mathsf{b}^-(\gamma)$ because the lambda binder must dominate all the var-nodes that it binds, and γ, β, α are arranged into a reducing tree. But then we must have $\lambda(c_\delta(\pi)) = c_\gamma(\lambda(\pi))$ by (β.λ.diff), so the condition (gp.λ.diff) is fulfilled.

Now suppose π' is a var-labeled node in $\mathsf{b}^-(\alpha'_i)$ that is bound in a different segment, i.e. $\lambda(\pi') \in S' - \mathsf{b}^-(\alpha'_i)$. There are three possibilities: Either $\lambda(\pi') \in \mathsf{b}^-(\alpha'_j)$ for $j \neq i$, or $\lambda(\pi') \in \mathsf{b}^-(\beta')$ or $\lambda(\pi') \in \mathsf{b}^-(\gamma')$. The first case is impossible as the

holes of the segment β' are disjoint (Def. 2.2, p. 27). The second case is impossible by condition ($\beta.\lambda$.diff) and the fact that no node of $\mathsf{b}(\beta)$ dominates any node of $\mathsf{b}(\alpha)$. The only remaining case is the third one: $\lambda(\pi') \in \mathsf{b}^-(\gamma')$. Let π be the correspondent of π', i.e. $c^i_\alpha(\pi) = \pi'$. Then by condition ($\beta.\lambda$.diff), we must have $\lambda(\pi') = c_\gamma(\lambda(\pi))$, so (gp.$\lambda$.diff) is fulfilled. The case that π' is a var-labeled node in $\mathsf{b}^-(\beta')$ that is bound in a different segment is completely analogous.

(gp.λ.out): The case of a var-labeled node $\pi \in \mathsf{b}^-(\delta)$ with $\lambda(\pi) \notin S$ is subsumed by ($\beta.\lambda$.out), and likewise the case of a var-labeled node $\pi' \in \mathsf{b}^-(\delta')$.

(gp.λ.hang): Both above groups satisfy condition (gp.λ.hang). This is clear for the group $(\gamma', \beta', \alpha'_1, \ldots, \alpha'_n)$, which covers the complete subtree below $r(\gamma')$. A similar argument applies to $(\gamma, \beta, \alpha, \ldots, \alpha)$. This group covers the whole tree below $r(\gamma)$ exept the @-labeled node π_0, the lam-labeled node π_1 and the var-labeled nodes $hs(\beta)$. But these var-labeled nodes are bound by π_1.

(gp.ante.same), (gp.ante.diff), (gp.ante.out): These properties of group parallelism are subsumed by the corresponding conditions on the beta reduction relation (Def. 6.2).

□

So it suffices to extend CLLS by group parallelism and inverse lambda binding literals to be able to express beta reduction literals. In the rest of this chapter, we regard $(C, B, A) \rightarrow^\beta (C', B', A'_1, \ldots, A'_n)$ as a *formula* abbreviating the right-hand side of the equation in Lemma 6.4. We call CLLS extended by group parallelism literals and inverse lambda binding literals the language $CLLS_{\text{gr}}$.

6.3 Extending the Semi-Decision Procedure for CLLS to Group Parallelism and Inverse Lambda Binding Literals

In this section we extend the semi-decision procedure $\mathcal{P}$ for CLLS such that it can also handle group parallelism and inverse lambda binding literals. As we have seen in the previous section, the thus extended procedure will also be able to handle beta reduction formulas.

We use some more formulas to make the rules easier to read. First we introduce a symmetric group parallelism formula: Let $\overline{A} = A_1, \ldots, A_n$ and $\overline{B} = B_1, \ldots, B_n$, then

$$\overline{A} \sim^{\text{sym}} \overline{B} =_{\text{def}} \overline{A} \sim \overline{B} \;\vee\; \overline{B} \sim \overline{A}.$$

This is simply an extension of the symmetric parallelism formula $A \sim^{\text{sym}} B$ to group parallelism. Similarly we extend the formulas that state that some variable is (or is not)

inside some segment term. They are defined in Chapter 4, p. 79.

$$
\begin{array}{rcl}
X \in \mathsf{b}(A_1,\ldots,A_m) & =_{\text{def}} & \bigvee_{i=1}^{m} X \in \mathsf{b}(A_i) \\
X \in \mathsf{b}^-(A_1,\ldots,A_m) & =_{\text{def}} & \bigvee_{i=1}^{m} X \in \mathsf{b}^-(A_i) \\
X \notin \mathsf{b}(A_1,\ldots,A_m) & =_{\text{def}} & \bigwedge_{i=1}^{m} X \notin \mathsf{b}(A_i) \\
X \notin \mathsf{b}^-(A_1,\ldots,A_m) & =_{\text{def}} & \bigwedge_{i=1}^{m} X \notin \mathsf{b}^-(A_i)
\end{array}
$$

Next we generalize correspondence formulas to the group parallelism case. Let $\overline{A} = A_1,\ldots,A_n$ and $\overline{B} = B_1,\ldots,B_n$. Then

$$
\begin{array}{rcl}
\mathrm{co}_k(\overline{A},\overline{B})(U){=}V & =_{\text{def}} & \overline{A} \sim^{\text{sym}} \overline{B} \wedge\ \mathrm{p}\big(\begin{smallmatrix} X_k^0 & Y_k^0 \\ U & V \end{smallmatrix}\big)\ \wedge U \in \mathsf{b}(A_k) \\
\mathrm{co}_k^-(\overline{A},\overline{B})(U){=}V & =_{\text{def}} & \mathrm{co}_k(\overline{A},\overline{B})(U){=}V \wedge U \in \mathsf{b}^-(A_k)
\end{array}
$$

for $1 \le k \le n$, $A_k = X_k^0/\ldots$, $B_k = Y_k^0/\ldots$. There are some more formulas that we discuss when we get to the rules that use them.

6.3.1 Solving CLLS$_{\text{gr}}$ Constraints: Procedure $\mathcal{P}_{\text{gr}}$.

Let $\overline{A} = A_1,\ldots,A_n$ and $\overline{B} = B_1,\ldots,B_n$.

Lifting the Core Parallelism Rules

(GP.init)	$\overline{A} \sim \overline{B} \to \mathsf{seg}(A_k) \wedge \mathsf{seg}(B_k) \wedge \mathrm{co}_k(\overline{A},\overline{B})(X_k^j){=}Y_k^j$ where $1 \le k \le n$, $A_k = X_k^0/X_k^1,\ldots,X_k^{m_k}$, $B_k = Y_k^0/Y_k^1,\ldots,Y_k^{m_k}$, and $0 \le j \le m_k$
(GP.new)	$\overline{A} \sim^{\text{sym}} \overline{B} \wedge U \in \mathsf{b}(A_k) \to \exists U'.\mathrm{co}_k(\overline{A},\overline{B})(U){=}U'$ where U' is a fresh variable, $1 \le k \le n$
(GP.copy.dom)	$U_1 R U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_k(\overline{A},\overline{B})(U_i){=}V_i \to V_1 R V_2$ where $1 \le k \le n$, $R \in \{\triangleleft^*, \perp, \neq\}$
(GP.copy.lab)	$U_0{:}f(U_1,\ldots,U_n) \wedge \bigwedge_{i=0}^{n} \mathrm{co}_k(\overline{A},\overline{B})(U_i){=}V_i \wedge U_0 \in \mathsf{b}^-(A_k) \to V_0{:}f(V_1,\ldots,V_n)$ where $1 \le k \le n$
(GP.distr.seg)	$\overline{A} \sim^{\text{sym}} \overline{B} \wedge X_k^0 \triangleleft^* X \to X \in \mathsf{b}(A_k) \vee \bigvee_{j=1}^{m_k} X_k^j \triangleleft^+ X$ where $1 \le k \le n$, $A_k = X_k^0/X_k^1,\ldots,X_k^{m_k}$

Binding

(D.λ.equal)	$\lambda(X_1){=}X_2 \wedge \bigwedge_{i=1}^{2} X_i{=}Y_i \to \lambda(Y_1){=}Y_2$
(D.λ.inverse)	$\lambda^{-1}(X){=}\{Y_1,\ldots,Y_m\} \to \bigwedge_{i=1}^{m} \lambda(Y_i){=}X$

(D.λ.distr.inv) $\lambda^{-1}(X)=\{Y_1,\ldots,Y_m\} \wedge \lambda(Z)=X \to \bigvee_{i=1}^{m} Z=Y_i$

(GP.λ.same) $\lambda(U_1)=U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_k^-(\overline{A},\overline{B})(U_i)=V_i \to \lambda(V_1)=V_2$
where $1 \leq k \leq n$

(GP.λ.diff) $\lambda(U_1)=U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_{k_i}^-(\overline{A},\overline{B})(U_i)=V_i \wedge U_2 \notin \mathsf{b}^-(A_{k_1}) \to \lambda(V_1)=V_2$
where $1 \leq k_1, k_2 \leq n$

(GP.λ.out) $\lambda(U)=Y \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(U)=V \wedge Y \notin \mathsf{b}^-(\overline{A}) \to \lambda(V)=Y$
where $1 \leq k \leq n$

(GP.λ.hang) $\lambda(U_1)=U_2 \wedge \overline{A} \sim^{\mathrm{sym}} \overline{B} \wedge U_2 \in \mathsf{b}^-(\overline{A}) \to U_1 \in \mathsf{b}^-(\overline{A})$

(GP.ante.same) $\mathsf{ante}(U_1)=U_2 \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(U_1)=V_1 \wedge \mathrm{co}_k(\overline{A},\overline{B})(U_2)=V_2 \wedge \overline{A} \sim \overline{B}$
$\to \mathsf{ante}(V_1)=U_1 \ \vee \ \mathsf{ante}(V_1)=V_2$ where $1 \leq k \leq n$

(GP.ante.diff) $\mathsf{ante}(U_1)=U_2 \wedge \mathrm{co}_{k_1}^-(\overline{A},\overline{B})(U_1)=V_1 \wedge \mathrm{co}_{k_2}(\overline{A},\overline{B})(U_2)=V_2 \wedge$
$U_2 \notin \mathsf{b}(A_{k_1}) \wedge \overline{A} \sim \overline{B} \to \mathsf{ante}(V_1)=U_1 \ \vee \ \mathsf{ante}(V_1)=V_2$
where $1 \leq k_1, k_2 \leq n$

(GP.ante.out) $\mathsf{ante}(U)=Y \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(U)=V \wedge Y \notin \mathsf{b}(\overline{A}) \wedge \overline{A} \sim \overline{B} \to \mathsf{ante}(V)=U$

(GP.λ.distr.1) $\lambda(U_1)=U_2 \wedge \overline{A} \sim^{\mathrm{sym}} \overline{B} \wedge U_1 \in \mathsf{b}^-(\overline{A}) \to \mathsf{distr}^-_{\overline{A}}(U_2)$

(GP.λ.distr.2) $\lambda(U_1)=U_2 \wedge \overline{A} \sim^{\mathrm{sym}} \overline{B} \wedge U_2 \in \mathsf{b}^-(\overline{A}) \to \mathsf{distr}^-_{\overline{A}}(U_1)$

(GP.ante.distr.1) $\mathsf{ante}(U_1)=U_2 \wedge \overline{A} \sim \overline{B} \wedge U_1 \in \mathsf{b}^-(\overline{A}) \to \mathsf{distr}_{\overline{A}}(U_2)$

(GP.ante.distr.2) $\mathsf{ante}(U_1)=U_2 \wedge \overline{A} \sim \overline{B} \wedge U_2 \in \mathsf{b}(\overline{A}) \to \mathsf{distr}^-_{\overline{A}}(U_1)$

(GP.λ.inverse) $\lambda^{-1}(X)=S_1 \wedge \mathrm{co}_k^-(\overline{A},\overline{B})(X)=Y \wedge \mathrm{co}^-(\overline{A},\overline{B})(S_1)=S_2 \cup S_3 \wedge$
$\bigwedge_{V \in S_2} \lambda(V)=Y \wedge \bigwedge_{V \in S_3} \lambda(V) \neq Y \to \lambda^{-1}(Y)=S_2$
where $1 \leq k \leq n$

plus the rules of the semi-decision procedure $\mathcal{P}$ for CLLS in Sec. 5.6.

6.3.2 The Rules in Detail

Section 6.3.1 shows the semi-decision procedure $\mathcal{P}_{\mathrm{gr}}$ for $\mathrm{CLLS}_{\mathrm{gr}}$.[3] The first block of rules lifts the core rules of the parallelism constraint procedure to the group parallelism case. (GP.init), (GP.copy.dom), (GP.copy.lab), (GP.new), and (GP.distr.seg) are straightfor-

[3] The procedure as we present it here is more extensive than in the paper by Bodirsky, Erk, Koller and Niehren [12]. This is due mostly to the fact that they used correspondence *literals* rather than formulas and thus could reuse more rules of the CLLS procedure.

ward generalization of their counterparts in Section 5.6. The tasks of the new rules are the same as those of the old ones: (GP.init) introduces correspondence formulas for roots and holes of a pair of parallel segment terms, (GP.copy.dom) copies dominance, inequality, and disjointness literals, (GP.copy.lab) copies labeling literals, (GP.new) introduces correspondence formulas for variables inside a segment term involved in a group parallelism literal, and (GP.distr.seg) guesses whether a variable is inside a segment term involved in a group parallelism literal. Compare (GP.new), which is $\overline{A} \sim^{\mathrm{sym}} \overline{B} \wedge U \in \mathsf{b}(A_k) \rightarrow \exists U'\ \mathrm{co}_k(\overline{A}, \overline{B})(U){=}U'$, where $\overline{A} = A_1, \ldots, A_n$, $\overline{B} = B_1, \ldots, B_n$, U' is a fresh variable and $1 \leq k \leq n$, to (P.new), which is $A{\sim}^{\mathrm{sym}} B \wedge U \in \mathsf{b}(A) \rightarrow \exists U'.\mathrm{co}(A, B)(X){=}U'$, where again U' is a fresh variable. The only difference is that (GP.new) picks out the k-th of n parallel segment pairs.

The rule (D.λ.equal), of the form $\lambda(X_1){=}X_2 \wedge \bigwedge_{i=1}^{2} X_i{=}Y_i \rightarrow \lambda(Y_1){=}Y_2$, will help us make the other rules easier to write down. The rule (D.λ.inverse), which is $\lambda^{-1}(X){=}\{Y_1, \ldots, Y_m\} \rightarrow \bigwedge_{i=1}^{m} \lambda(Y_i){=}X$, infers lambda binding literals from the matching inverse lambda binding literal. The rule (D.λ.distr.inv), which is $\lambda^{-1}(X){=}\{Y_1, \ldots, Y_m\} \wedge \lambda(Z){=}X \rightarrow \bigvee_{i=1}^{m} Z{=}Y_i$, states that when $\{Y_1, \ldots, Y_m\}$ are all the variables bound at X, then any variable Z bound at X must be equal to one of the Y_i. The rules (GP.λ.same) through (GP.λ.hang) and (GP.ante.same) through (GP.ante.out) express the conditions on lambda binding and anaphoric binding laid down in Def. 6.3. (GP.λ.same) has the form $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_k^-(\overline{A}, \overline{B})(U_i){=}V_i \rightarrow \lambda(V_1){=}V_2$; it handles the case of both U_1 and U_2 being inside the same segment term of a group. The rule (GP.λ.diff) is $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}_{k_i}^-(\overline{A}, \overline{B})(U_i){=}V_i \wedge U_2 {\notin} \mathsf{b}^-(A_{k_1}) \rightarrow \lambda(V_1){=}V_2$. It handles the case that U_2 is inside a segment term of the group, but not the same segment term that U_1 is in. (GP.λ.out) handles the case of U being bound outside the whole group, and (GP.λ.hang) states that hanging binders (lam-labeled nodes inside the group binding variables outside the group) are not permitted. (GP.ante.same), (GP.ante.diff), and (GP.ante.out) handle the cases of a variable being anaphorically bound inside the same segment term, in a different segment term of the group, and outside the group, respectively.

In the procedure $\mathcal{P}$, we did not need any distribution rule for determining which of (P.λ.same) or (P.λ.out) applied to a lambda binder. (This is shown in the discussion of (P.λ.same) and (P.λ.out) in the proof of Lemma 4.19, p. 98.) But for the new CLLS$_{\mathrm{gr}}$ procedure this is different because now we have to consider all segment terms of the same group. The distribution rules (GP.λ.distr...) and (GP.ante.distr...) use the formulas

$$\begin{aligned} \mathsf{distr}^-_{\overline{A}}(U) &=_{\mathrm{def}} \textstyle\bigwedge_{i=1}^{n} \big(U \in \mathsf{b}^-(A_i) \vee U {\notin} \mathsf{b}^-(A_i)\big) \\ \mathsf{distr}_{\overline{A}}(U) &=_{\mathrm{def}} \textstyle\bigwedge_{i=1}^{n} \big(U \in \mathsf{b}(A_i) \vee U {\notin} \mathsf{b}(A_i)\big) \end{aligned}$$

again for $\overline{A} = A_1, \ldots, A_n$. Given a lambda binding literal $\lambda(U_1){=}U_2$, if $U_1 \in \mathsf{b}^-(\overline{A})$ is in the constraint for a group $\overline{A}$ of a group parallelism literal, then (GP.λ.distr.1) guesses whether U_2 is inside a segment term of the same group, and if U_2 is inside the group, then (GP.λ.distr.2) guesses whether U_1 is, too. For an anaphoric binding literal $\mathsf{ante}(U_1){=}U_2$, the rules (GP.ante.distr.1) and (GP.ante.distr.2) perform the same guesses.

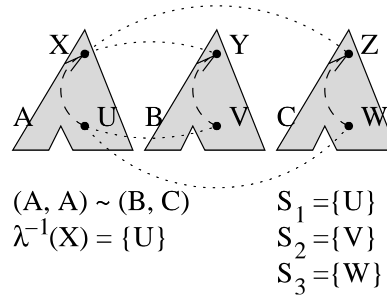


Figure 6.7: Illustrating the rule (GP.λ.inverse)

The most complex rule is (GP.λ.inverse), which handles the copying of inverse lambda binding literals. This rule is important in the application of group parallelism to beta reduction: Inverse lambda binding is needed to identify reducing trees, so we need to copy these literals to the reduct in case we want to perform a second beta reduction step. The rule uses two more formulas, the first just being

$$\lambda(X){\neq}Y =_{\text{def}} \exists Z(\lambda(X){=}Z \;\wedge\; Z{\neq}Y).$$

The second formula collects, for a finite set S_1 of variables, all correspondents with respect to $\overline{A} \sim \overline{B}$. Let S_1, S_2 stand for finite sets of variables, and let $\overline{A} = A_1 \ldots, A_n$. Then

$$\begin{array}{rcl} \text{co}^-(\overline{A},\overline{B})(S_1){=}S_2 & =_{\text{def}} & \bigwedge_{i=1}^n \bigwedge_{X\in S_1} (X{\notin}\mathsf{b}^-(A_i) \vee \bigvee_{Y\in S_2} \text{co}_i^-(\overline{A},\overline{B})(X){=}Y) \\ & \wedge & \bigwedge_{Y\in S_2} \bigvee_{X\in S_1} \bigvee_{i=1}^n \text{co}_i^-(\overline{A},\overline{B})(X){=}Y \end{array}$$

Using these two formulas, the rule (GP.λ.inverse), of the form $\lambda^{-1}(X){=}S_1 \wedge \text{co}_k^-(\overline{A},\overline{B})(X){=}Y \wedge \text{co}^-(\overline{A},\overline{B})(S_1){=}S_2 \cup S_3 \wedge \bigwedge_{V\in S_2} \lambda(V){=}Y \wedge \bigwedge_{V\in S_3} \lambda(V){\neq}Y \;\rightarrow\; \lambda^{-1}(Y){=}S_2$, collects all correspondents of a variable bound by X. We have to know, for each of these correspondents, whether it is bound by Y or definitely bound by something else. Only then can we determine $\lambda^{-1}(Y)$. The rule is illustrated in Fig. 6.7: The leftmost segment term A is parallel to both the middle segment term B and the rightmost segment term C. The variable X inside A corresponds to both Y in B and Z in C , and U, which is bound at X, corresponds to both V in B and W inside C. As usual, we draw correspondence formulas as dotted arcs. We have $\lambda^{-1}(X) = \{U\}$, so $S_1 = \{U\}$. V is bound at Y, but W is bound at Z following condition (gp.λ.same). So for the binder X and its first correspondent Y, we have $S_2 = \{V\}$ and $S_3 = \{W\}$. For the constraint sketched in Fig. 6.7, we get $\lambda^{-1}(Y) = \{V\}$ by rule (GP.λ.inverse).

6.3.3 An Example

We illustrate the procedure $\mathcal{P}_{\text{gr}}$ on the constraint in Fig. 6.8. It contains the reducing tree (C, B, A), along with a group parallelism literal $(C, B, A, A){\sim}(C', B', A', A'')$ that describes the result of a beta reduction step for this reducing tree. Furthermore, there is the lam-labeled variable Y_1, which may belong either to the context C or to the argument A. Thus, the variables Y_1 and Y_2 form a "D segment term" like the one in Fig. 6.3.

$(C, B, A, A) \sim (C', B', A', A'')$
with $C = X/X_0$,
$C' = X'/X'_0$,
$B = X_t/X_1, X_2$,
$B' = X'_0/X'_1, X'_2$,
$A = X_a/$,
$A' = X'_1/$, and
$A'' = X'_2/$.
$\lambda^{-1}(X_\ell) = \{X_1, X_2\}$,
$\lambda^{-1}(Y_1) = \{Z\}$.

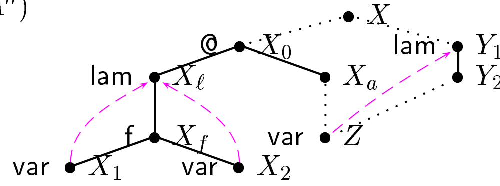


Figure 6.8: A group parallelism constraint encoding a beta reduction step

Figure 6.9 and 6.10 list computation steps that the procedure $\mathcal{P}_{\text{gr}}$ can perform on this constraint. In step (4), the procedure guesses whether Y_1, Y_2 belong to the context C or to the argument A. We have to disambiguate the position of the fragment consisting of Y_1, Y_2 before we can fully solve the group parallelism literal. The left column, starting with (4a), explores the case that Y_1, Y_2 belong to A, and the right column, starting with

<table>
<tr><td colspan="2">(1) $Y_1 \neq X_0$ (D.lab.ineq)
(2) $Y_1 \lhd^+ Y_2, X_0 \lhd^+ X_a$ (D.lab.dom)
(3) $Y_1 \lhd^* Z, X_0 \lhd^* Z$ (D.dom.trans)
(4) $X_0 \lhd^* Y_1 \ \vee\ Y_1 \lhd^* X_0$ (D.distr.notDisj)</td></tr>
<tr><td>(4a) $X_0 \lhd^* Y_1$:
(5) $X_0{=}Y_1 \vee X_\ell \lhd^* Y_1 \vee X_a \lhd^* Y_1$ (D.distr.child)</td><td>(4b) $Y_1 \lhd^* X_0$:
(7) $Y_1{=}X_0 \ \vee\ Y_2 \lhd^* X_0$ (D.distr.child)</td></tr>
<tr><td>(5a) $X_0{=}Y_1$: (5b) $X_\ell \lhd^* Y_1$:
... both lead to <code>false</code></td><td>(7a) $Y_1{=}X_0$:
(8) <code>false</code> (D.clash.ineq)</td></tr>
<tr><td>(5c) $X_a \lhd^* Y_1$:
(9) $\text{co}(A, A')(X_a){=}X'_1$ (P.init)
(10) $\text{co}(A, A')(Y_1){=}Y'_1$, (P.new)
$\text{co}(A, A')(Y_2){=}Y'_2$,
$\text{co}(A, A')(Z){=}Z'$
(11) $X'_1 \lhd^* Y'_1, Y'_2 \lhd^* Z'$ (P.copy.dom)
(12) Y'_1:lam(Y'_2) (P.copy.lab)
(13) $\text{co}(A, A'')(X_a){=}X'_2$ (P.init)
(14) $\text{co}(A, A'')(Y_1){=}Y''_1$, (P.new)
$\text{co}(A, A'')(Y_2){=}Y''_2$,
$\text{co}(A, A'')(Z){=}Z''$
(15) $X'_2 \lhd^* Y''_1, Y''_2 \lhd^* Z''$ (P.copy.dom)
(16) Y''_1:lam(Y''_2) (P.copy.lab)</td><td>(7b) $Y_2 \lhd^* X_0$:
(17) $\text{co}(C, C')(X){=}X'$, (P.init)
$\text{co}(C, C')(X_0){=}X'_0$
(18) $\text{co}(C, C')(Y_1){=}Y'_1$, (P.new)
$\text{co}(C, C')(Y_2){=}Y'_2$
(19) $X' \lhd^* Y'_1, Y'_2 \lhd^* X'_0$ (P.copy.dom)
(20) Y'_1:lam(Y'_2) (P.copy.lab)</td></tr>
</table>

Figure 6.9: Solving the group parallelism constraint in Fig. 6.8

	Continuing (5c)			
(21)	$\lambda(Z') = Y_1'$, $\lambda(Z'') = Y_1''$	(GP.λ.same)		
(22)	$Z \notin \mathtt{b}^-(B) \;\vee\; Z \in \mathtt{b}^-(B)$	(GP.λ.distr.2)		
(22a)	$Z \notin \mathtt{b}^-(B)$		(22b)	$Z \notin \mathtt{b}^-(B)$
(23)	$Z \notin \mathtt{b}^-(C) \;\vee\; Z \in \mathtt{b}^-(C)$	(GP.λ.distr.2)	...	`false`
(23a)	$Z \notin \mathtt{b}^-(C)$		(23b)	$Z \in \mathtt{b}^-(C)$
(24)	$Y_1' \neq Y_1'' \;\vee\; Y_1' = Y_1''$	(P.distr.eq)	...	`false`
(24a)	$Y_1' \neq Y_1''$		(24b)	$Y_1' = Y_1''$
(25)	$\lambda^{-1}(Z'') \neq Y_1'$		...	`false`
(26)	$\lambda^{-1}(Y_1'') = \{Z'\}$	(GP.λ.inverse)		
(27)	$\lambda^{-1}(Y_1'') = \{Z''\}$	(GP.λ.inverse)		

Figure 6.10: Inverse Binding in case (5c)

(4b), explores the case that they belong to the context C. In the left column we make two copies of Y_1 and Y_2 each. This is because A receives two copies in the reduct, A' and A'' (which in turn is caused by X_ℓ binding two occurrences of the object-level variable, X_1 and X_2). In the right column, on the other hand, Y_1 and Y_2 are only copied once: They belong to the context C, which is parallel only to C'.

Figure 6.10 continues the left column of Fig. 6.9, i.e. Y_1, Y_2 belong to the argument A. The purpose of this figure is to demonstrate rule (GP.λ.inverse): All steps from (22) on prepare the determination of $\lambda^{-1}(Y_1')$ and $\lambda^{-1}(Y_1'')$ in (25) and (26).

So, to sum up, the procedure $\mathcal{P}_{\mathrm{gr}}$ can solve this group parallelism literal, and the beta reduction formula that it is part of, but it has to disambiguate the position of the fragment consisting of Y_1, Y_2. In the following section we introduce a procedure for a single beta reduction step that can avoid disambiguation in many cases. (However, it will not be able to handle this particular example without disambiguating.)

6.3.4 Properties of the Procedure $\mathcal{P}_{\mathrm{gr}}$

We now prove properties of the procedure $\mathcal{P}_{\mathrm{gr}}$, extending the proofs from Chapter 4. At the end of this section, we sum them up in one theorem.

Soundness. As we have defined in Def. 3.3 (p. 62), a saturation rule is sound iff it is an equivalence transformation. And as we have remarked there, it suffices to show that the left-hand side of the rule entails the right-hand disjunction because we are working in a saturation framework.

The rules (GP.init), (GP.new), (GP.copy.dom), and (GP.copy.lab) are sound for the same reason as their (P...) counterparts – see Sec. 4.2.1 (p. 89). For rules (GP.distr.seg), (D.λ.equal), (D.λ.inverse) and (D.λ.distr.inv) soundness is obvious, likewise for (GP.λ.distr.1), (GP.λ.distr.2), (GP.ante.distr.1), and (GP.ante.distr.2). The

rules (GP.λ.same), (GP.λ.diff), (GP.λ.out), (GP.λ.hang), (GP.ante.same), (GP.ante.diff), and (GP.ante.out) are direct translations of the conditions laid down in Def. 6.3. It remains to show the soundness of (GP.λ.inverse), which is not obvious: Is it really sufficient to look among the correspondents of $\lambda^{-1}(X)$ to compute $\lambda^{-1}(Y)$? The following lemma shows that it is.

Lemma 6.5 (Inverse lambda binding). *Given a lambda structure in which the group parallelism* $(\alpha_1, \ldots, \alpha_n) \sim (\alpha'_1, \ldots, \alpha'_n)$ *holds with correspondence functions* $c_1, \ldots, c_n$. *Then for all* $1 \leq k \leq n$ *and all* $\pi \in \mathsf{b}^-(\alpha_k)$,

$$\lambda^{-1}(c_k(\pi)) \subseteq \bigcup_{i=1}^{n} \{c_i(\pi') \mid \pi' \in \lambda^{-1}(\pi) \cap \mathsf{b}^-(\alpha_i)\}$$

Proof. Let $\psi' \in \lambda^{-1}(c_k(\pi))$. The "no hanging binders" condition (gp.λ.hang) of Def. 6.3 is critical here. It enforces $\psi' \in \bigcup_{i=1}^{n} \mathsf{b}^-(\alpha'_i)$. There are two possibilities. Either we have $\psi' \in \mathsf{b}^-(\alpha'_k)$. Then there is a node $\psi \in \mathsf{b}^-(\alpha_k)$ with $c_k(\psi) = \psi'$. ψ is var-labeled by Def. 2.3 (p. 27) and has a binder since λ is total. We must have $\lambda(c_k(\psi)) = c_k(\lambda(\psi))$ by condition (gp.λ.same). Now $\lambda(c_k(\psi)) = c_k(\pi)$ and c_k is a bijection, so $\psi \in \lambda^{-1}(\pi)$. The other possibility is that $\psi' \notin \mathsf{b}^-(\alpha'_k)$ but $\psi' \in \mathsf{b}^-(\alpha'_j)$ for some $j \neq k, 1 \leq j \leq n$. Then there is again a node ψ with $c_j(\psi) = \psi'$, and $\lambda(c_j(\psi)) = c_k(\lambda(\psi'))$ by condition (gp.λ.diff), so again $\psi \in \lambda^{-1}(\pi)$. □

To sum up, we have shown the following:

Lemma 6.6 (Soundness). *The semi-decision procedure* $\mathcal{P}_{\mathrm{gr}}$ *for CLLS*$_{\mathrm{gr}}$ *is sound for lambda structures.*

Nontermination. The procedure $\mathcal{P}_{\mathrm{gr}}$ subsumes the procedure $\mathcal{P}$, which for some input constraints does not terminate. An example is shown in Ex. 4.7.

Fairness. In Sec. 4.2.3 (p. 89) we have stated what we mean by fairness: Whenever a rule is applicable, one of the disjuncts in its conclusion will ultimately be added. For the procedure $\mathcal{P}_{\mathrm{gr}}$ we adapt the fairness conditions of Chapter 4 and 5 in a straightforward way:

> **Fairness condition.** (P.new) and (GP.new) are applied only to constraints saturated under $\mathcal{P}_{\mathrm{gr}}-$ {(P.new), (GP.new)}. (GP.new) and (P.new) are applied to variables in the order of their introduction into the constraint.

That is, the fairness condition simply treats (P.new) and (GP.new) the same way.

Saturated constraints. As in the previous chapters, we attempt to give an independent description of saturated constraints by describing what their constraint graphs look like (i.e. we attempt no more than an informal description of saturations). In the current case, this is particularly easy: A $\mathcal{P}_{\text{gr}}$-saturation looks just like a $\mathcal{P}$-saturation, except that the properties of lambda binding and anaphoric binding for some parallel regions (more concretely, those parallel regions that arise from group parallelism literals) are different.

Satisfiability of saturated constraints. We show that each saturated constraint that $\mathcal{P}_{\text{gr}}$ computes is satisfiable. To this purpose, we extend the definitions and lemmas of Section 5.3, which proves the same result for $\mathcal{P}$-saturated constraints.

As before, we restrict ourselves to *generated* constraints, which only contain path literals that have been added to process parallelism literals, but not path literals in arbitrary places. We have to adapt the definition of generatedness: Path literals can be licensed not only by segment terms of parallelism literals, but also by those of group parallelism literals. We write CLLS_{grp} for the language CLLS_{gr} extended by path literals.

Definition 6.7 (Correspondence-generated). *Let φ be a $CLLS_{\text{grp}}$-constraint. A path literal $p(\begin{smallmatrix} U_1 & V_1 \\ U_2 & V_2 \end{smallmatrix}) \in \varphi$ is* correspondence-generated in φ *iff*

- *either there exists some literal $A \sim B \in \varphi$ with $A = U_1/\ldots$ and $B = V_1/\ldots$ such that either $U_2 \in \mathsf{b}(A)$ or $V_2 \in \mathsf{b}(B)$ is in φ.*
- *or there exists some literal $(A_1, \ldots, A_n) \sim (B_1, \ldots, B_n) \in \varphi$ and some $k \in \{1, \ldots, n\}$ with $A_k = U_1/\ldots$ and $B_k = V_1/\ldots$ such that either $U_2 \in \mathsf{b}(A_k)$ or $V_2 \in \mathsf{b}(B_k)$ is in φ.*

We lift the definition of a *generated* constraint (Def. 4.14, p. 92) canonically to this new definition of correspondence-generatedness. Lemma 4.15 (p. 92) still holds for the new definition of generatedness.

In Chapters 3, 4 and 5 we showed satisfiability of saturated constraints in two steps: We showed first that every *simple* generated saturation is satisfiable, then we showed how to reduce a non-simple saturation to a simple one. We proceed in the same way here. According to Def. 3.7 (p. 64), a simple $\mathcal{C}_{\text{d}}$-constraint possesses a root variable dominating all others, and every variable of the constraint is labeled. We lift this definition canonically: A CLLS_{grp} constraint is called simple iff its maximal subset that is a $\mathcal{C}_{\text{d}}$ constraint is simple.

Lemma 6.8 (Satisfiability of simple generated saturations). *A simple generated $\mathcal{P}_{\text{gr}}$-saturated $CLLS_{\text{grp}}$-constraint is satisfiable.*

Proof. Let φ be a CLLS_{grp} constraint that is a simple generated $\mathcal{P}_{\text{gr}}$-saturation. Since group parallelism is a canonical extension of parallelism that only differs in its conditions on binding, we can basically reuse the proofs of Lemmas 4.16 and 5.5 (p. 93 and 126).

First, let φ_{dom} be the maximal subset of φ that is a $\mathcal{C}_\text{d}$ constraint, and let (θ,σ) be a model for φ_{dom} constructed as in the proof of Lemma 3.8 (p. 65). Then each path literal and each group parallelism literal of φ is satisfied in θ: The proof is completely analogous to the one for parallelism literals in Lemma 4.16, except that in the argument the rules (GP.init), (GP.new), (GP.copy.dom), (GP.copy.lab), (GP.distr.seg) replace their (P...) counterparts.

It remains to consider the binding literals. As in the proof of Lemma 5.5, we extend θ to a lambda structure $\mathcal{L}^{\theta'}$ that is a model of φ: Let $S \subseteq \mathcal{V}ar(\varphi)$ be the set of var-labeled variables without a lambda binder in φ, then we construct our new tree θ' as $\theta' = \mathsf{lam}(\theta)$. As before, we set

$$\lambda(\sigma(X)) = \begin{cases} \sigma(Y) & \text{if } \lambda(X)=Y \text{ in } \varphi \\ \varepsilon & \text{if } X \in S \end{cases}$$

and

$$\mathsf{ante}(\sigma(X))=\sigma(Y) \text{ if } \mathsf{ante}(X)=Y \text{ in } \varphi$$

for all $X \in \mathcal{V}ar(\varphi)$. We have shown in Lemma 5.5 that both functions are well-defined, the binders and the bound nodes are labeled as they should be, and that λ is a total function in which a binder dominates each node that it binds. The binding functions of $\mathcal{L}^{\theta'}$ interact correctly with its group parallelism relation: Either a rule (GP.λ...) or a rule (GP.ante...) is applicable to each bound variable of φ because of Lemma 5.4 (p. 126), so the rules (GP.λ.same), (GP.λ.diff), (GP.λ.out), (GP.λ.hang), (GP.ante.same), (GP.ante.diff), and (GP.ante.out) enforce the conditions of Def. 6.3. If a var-labeled variable is not bound in φ, our definition of θ' ensures that the variable's binder interacts correctly with the group parallelism relation (again in the same way as in Lemma 5.5). Furthermore, all inverse lambda binding literals of φ are satisfied in $\mathcal{L}^{\theta'}$: If φ contains a literal $\lambda^{-1}(X)=\{Y_1,\dots,Y_m\}$ then by closure under (D.λ.inverse) and the fact that $m \geq 1$, $\sigma(X)$ is not bound at ε in $\mathcal{L}^{\theta'}$, and by closure under (D.λ.distr.inv) and our construction of the function λ, if $\lambda(\pi)=\sigma(X)$ in $\mathcal{L}^{\theta'}$, then $\sigma(Y_i)=\pi$ for some $i \in \{1,\dots,m\}$. □

Next we show that we can again extend any non-simple saturated constraint to a simple one. To that end, we have to adapt the definition of $\hookrightarrow_\varphi$ slightly: It has to build on group parallelism (which subsumes normal parallelism) now.

Definition 6.9 (Copy set). *Let φ be a CLLS$_\text{grp}$ constraint. Then $\hookrightarrow_\varphi$ is the largest relation on equal-sized sequences of variables in $\mathcal{V}ar(\varphi)$ such that*

$$(U_0, U_1, \dots, U_m) \hookrightarrow_\varphi (V_0, V_1, \dots, V_m)$$

implies that there exists a group parallelism literal $(A_1,\dots,A_n) \sim (B_1,\dots,B_n)$ in φ and some $k \leq n$ such that $U_i \in \mathsf{b}(A_k)$ and $U_0 \in \mathsf{b}^-(A_k)$ are in φ for $1 \leq i \leq m$ and $co_k(\overline{A},\overline{B})(U_i)=V_i$ is in φ for $0 \leq i \leq m$.

The definition of $\text{copy}_\varphi(U_0,U_1,\dots,U_m)$ is lifted canonically to the new definition of $\hookrightarrow_\varphi$. Lemma 4.18 (p. 96) still holds: Sequences $(U_0,U_1,\dots,U_n)$, $(V_0,V_1,\dots,V_n)$ in the same

copy set share the properties that first, if U_0 is unlabeled, then so is V_0; second, if $(U_1, \ldots, U_n)$ is in the connectedness set (Def. 3.9, p. 66) of U_0, then $(V_1, \ldots, V_n)$ is in the connectedness set of V_0; third, if $(U_1, \ldots, U_n)$ are a maximal disjointness set (Def. 3.10, p. 67) within the connectedness set of U_0, then the same holds of $(V_1, \ldots, V_n)$ and V_0.

Now we show, like in the previous chapters, that given a generated saturation that is not simple, we can extend it by labeling at least one previously unlabeled variable.

Lemma 6.10 (Extension by labeling). *Every $\mathcal{P}_{\mathrm{gr}}$-saturated $CLLS_{\mathrm{grp}}$-constraint with an unlabeled variable U_0 can be extended to a $\mathcal{P}_{\mathrm{gr}}$-saturated constraint in which U_0 is labeled.*

Proof. Let $\{U_1, \ldots, U_m\}$ be a maximal φ-disjointness set in $\mathrm{con}_\varphi(U_0)$. As in the proof of Lemma 5.6 (p. 127), we assume that Σ contains a function symbol f of arity m that is neither var, lam or ana. (If it does not, then we can encode it using a nullary function symbol and a symbol of arity 2, as in Lemma 3.12.) We use the same definition of an extension $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ of $\varphi \wedge U_0{:}f(U_1, \ldots, U_m)$ as in Lemma 4.19 (p. 98). We repeat it here:

$$\mathsf{ext}_{U_0,\ldots,U_m}(\varphi) =_{\mathrm{def}} \varphi \ \wedge \bigwedge_{\substack{V_0:f(V_1,\ldots,V_m)\in \\ \mathrm{copy}_\varphi\left(U_0,U_1,\ldots,U_m\right)}} \begin{array}{l}\Big(V_0{:}f(V_1, \ldots, V_m) \ \wedge \ \bigwedge_{i=1}^{m} V_0{\neq}V_i \ \wedge \\ \bigwedge_{\substack{V_i \lhd^* Z, V_j \lhd^* W \in \varphi, \\ 1\leq i<j\leq n}} Z{\perp}W \ \wedge \\ \bigwedge_{\substack{Z:g(\ldots)\in\varphi, \\ g\neq f \ \vee \ \mathsf{ar}(g)\neq\mathsf{ar}(f)}} Z{\neq}V_0\Big)\end{array}$$

For simplicity, we write $\mathsf{ext}(\varphi)$ for $\mathsf{ext}_{U_0,\ldots,U_m}(\varphi)$ in the rest of the proof.

To show that $\mathsf{ext}(\varphi)$ is a $\mathcal{P}_{\mathrm{gr}}$-saturation, we examine each of the rules in Sec. 6.3.1 and show that none of them is applicable to the constraint. (For the other rules of $\mathcal{P}_{\mathrm{gr}}$, listed in Sec. 5.6, we have already shown in Lemmas 3.12, 4.19, and 5.6 (p. 67, 98, and 127) that they are not applicable.)

(GP.init), (GP.new), (GP.copy.dom), (GP.copy.lab), (GP.distr.seg): These rules are just group parallelism versions of the matching (P...) rules. The only difference is that the group parallelism rules pick out the k-th of n segment terms of a group while in the parallelism rules there is only one segment term in each "group" (i.e. on each side of the $\sim$). The constraint $\mathsf{ext}(\varphi)$ is closed under the group parallelism rules for the same reasons that it is closed under their (P...) counterparts – see Lemma 4.19.

(D.λ.equal), (D.λ.inverse), (D.λ.distr.inv): We have not added any lambda binding, inverse lambda binding, or dominance literals.

(GP.λ.distr.i), (GP.ante.distr.i), $i = 1, 2$: (GP.λ.distr.1) has the form $\lambda(U_1){=}U_2 \wedge \overline{A} \sim^{\text{sym}} \overline{B} \wedge U_1{\in}\mathsf{b}^-(\overline{A}) \to \mathsf{distr}^-_{\overline{A}}(U_2)$, and the other three rules are very similar. We have not introduced any new group parallelism, lambda binding or anaphoric binding literals. Also, we have not introduced new dominance literals. So the only thing that could be new in $\mathsf{ext}(\varphi)$ could be the information $U_1 \in \mathsf{b}^-(\overline{A})$. Let $\overline{A} = (A_1, \ldots, A_n)$. Now if there exists some $k \leq n$ with $A_k = X^0_k/X^1_k, \ldots, X^{m_k}_k$ and $X^0_k \triangleleft^* U_1 \in \varphi$, then by the closure of φ under (GP.distr.seg) and (P.distr.eq) the constraint φ contains either $U_1 \in \mathsf{b}^-(A_k)$ or $U_1 \notin \mathsf{b}^-(A_k)$ already.

(GP.λ.same), (GP.λ.diff), (GP.λ.out), (GP.λ.hang): We focus on (GP.λ.out), which is $\lambda(U){=}Y \wedge \mathrm{co}^-_k(\overline{A}, \overline{B})(U){=}V \wedge Y{\notin}\mathsf{b}^-(\overline{A}) \to \lambda(V){=}Y$. We have not added any lambda binding literals. The correspondence formula stands for $\overline{A} \sim^{\text{sym}} \overline{B} \wedge\ \mathrm{p}\begin{pmatrix} X^0_k & Y^0_k \\ U & V \end{pmatrix} \wedge U \in \mathsf{b}(A_k) \wedge U \in \mathsf{b}^-(A_k)$, where $\overline{A} = A_1, \ldots, A_n$, $\overline{B} = B_1, \ldots, B_n$, $1 \leq k \leq n$, $A_k = X^0_k/\ldots$ and $B_k = Y^0_k/\ldots$. We have not added any group parallelism or path literals. The only thing that could still be new in $\mathsf{ext}(\varphi)$ is a formula $U \in \mathsf{b}(A_k)$ or $U \in \mathsf{b}^-(A_k)$ or $Y \notin \mathsf{b}^-(\overline{A})$. So suppose $U \in \mathsf{b}(A_k)$ is new in $\mathsf{ext}(\varphi)$. Since we have not added any dominance literals, $X^0_k \triangleleft^* U$ must be in φ already. But since φ is closed under (GP.distr.seg), φ already contains either $U \in \mathsf{b}(A_k)$ or $U \notin \mathsf{b}(A_k)$. And likewise $U \in \mathsf{b}^-(A_k)$ cannot be new in $\mathsf{ext}(\varphi)$ because φ already contains either $U \in \mathsf{b}(A_k)$ or $U \notin \mathsf{b}(A_k)$ and is closed under (P.distr.eq), which guesses equality or inequality between U and the holes of A_k. Furthermore, if $U \in \mathsf{b}^-(A_k)$ is in φ already, then by closure of φ under (GP.λ.distr.1) we know for each $j \leq n$ whether $Y \in \mathsf{b}^-(A_j)$ or $Y{\notin}\mathsf{b}^-(A_j)$. So one of (GP.λ.same), (GP.λ.diff), and (GP.λ.out) has been applied to U and Y in φ already.

For the rules (GP.λ.same), (GP.λ.diff) and (GP.λ.hang), the argument is the same.

(GP.ante.same), (GP.ante.diff), (GP.ante.out): We focus on (GP.ante.out), which is $\mathsf{ante}(U){=}Y \wedge \mathrm{co}^-_k(\overline{A}, \overline{B})(U){=}V \wedge Y{\notin}\mathsf{b}(\overline{A}) \wedge \overline{A} \sim \overline{B} \to \mathsf{ante}(V){=}U$. We have not added any anaphoric binding literals or group parallelism literals. Concerning the correspondence formula, the argument is the same as for the rules (GP.λ.same) through (GP.λ.hang): We have not added any group parallelism or path literals, and the formula $U \in \mathsf{b}^-(A_k)$ cannot be new because we have added no dominance literals and φ is closed under (GP.distr.seg). It remains to be shown that the formula $Y \notin \mathsf{b}(\overline{A})$ cannot be new in $\mathsf{ext}(\varphi)$. So suppose $(A_1, \ldots, A_n) \sim (B_1, \ldots, B_n)$ and $\mathsf{ante}(U){=}Y$ are in φ, and $U \in \mathsf{b}^-(A)_k$ is in $\mathsf{ext}(\varphi)$ for some $k \leq n$. Then in φ (GP.ante.distr.1) has already decided, for each $j \leq n$, whether Y belongs to $\mathsf{b}(A_j)$ or not.

For the rules (GP.ante.same) and (GP.ante.diff) the argument is analogous.

(GP.λ.inverse): This rule is $\lambda^{-1}(X){=}S_1 \wedge \mathrm{co}^-_k(\overline{A}, \overline{B})(X){=}Y \wedge \mathrm{co}^-(\overline{A}, \overline{B})(S_1){=}S_2 \cup S_3 \wedge \bigwedge_{V \in S_2} \lambda(V){=}Y \wedge \bigwedge_{V \in S_3} \lambda(V){\neq}Y \ \to\ \lambda^{-1}(Y){=}S_2$. We have not added any inverse lambda binding literals or lambda binding literals. Concerning correspondence formulas, we have argued above (for the rules (GP.λ.same) through (GP.λ.hang)) that they cannot be new in $\mathsf{ext}(\varphi)$.

Next, the formula $\text{co}^-(\overline{A},\overline{B})(S_1)=S_2 \cup S_3$ stands for $\bigwedge_{i=1}^n \bigwedge_{Z\in S_1}(Z\notin \mathsf{b}^-(A_i) \vee \bigvee_{Y\in S_2\cup S_3} \text{co}_i^-(\overline{A},\overline{B})(Z)=Y) \wedge \bigwedge_{Y\in S_2\cup S_3} \bigvee_{Z\in S_1} \bigvee_{i=1}^n \text{co}_i^-(\overline{A},\overline{B})(Z)=Y$. We show for each part of the formula that it cannot be new in $\mathsf{ext}(\varphi)$. Since φ contains $\lambda^{-1}(X)=S_1$ and is closed under (D.λ.inverse), it contains $\lambda(Z)=X$ for each $Z \in S_1$. So by (GP.λ.distr.2) and the fact that $X \in \mathsf{b}^-(\overline{A})$, the constraint φ contains either $Z \in \mathsf{b}^-(A_i)$ or $Z \notin \mathsf{b}^-(A_i)$ for each $1 \leq i \leq n$. Again, a correspondence formula $\text{co}_i^-(\overline{A},\overline{B})(Z)=Y$ cannot be new in $\mathsf{ext}(\varphi)$, for the same reasons as above. Finally, $\lambda(V)\neq Y$ cannot be new in $\mathsf{ext}(\varphi)$ because we have not added any lambda binding literals, and φ is closed under (P.distr.eq).

□

Completeness. For completeness, no new proofs are needed. Lemma 4.34 (p. 110) is general enough to cover the new saturation rules as well, and the proofs of Lemmas 4.38 and 5.9 (p. 112 and 130), which handle fresh variables introduced by (P.new) and by (D.λ.lam), apply to (GP.new) and (GP.λ.inverse) as well. So we directly get the following result (for the notation see Def. 4.33, p. 110):

Lemma 6.11 (Completeness). *Let φ be a CLLS$_{\text{grp}}$ constraint and $\mathcal{G} \subseteq \mathcal{V}ar(\varphi)$. Then $\mathcal{P}_{\text{gr}}$ can compute from φ, in a finite number of steps, any minimal $\mathcal{P}_{\text{gr}}$-saturation for φ with respect to $\mathcal{G}$.*

Recapitulation

In this section we have shown a number of properties of the procedure $\mathcal{P}_{\text{gr}}$, which we now sum up.

Theorem 6.12. *The semi-decision procedure $\mathcal{P}_{\text{gr}}$ for CLLS$_{\text{gr}}$ has the following properties:*

1. *It is sound for lambda structures.*
2. *There are unsatisfiable CLLS$_{\text{gr}}$ constraints for which it does not terminate.*
3. *A generated $\mathcal{P}_{\text{gr}}$-saturated CLLS$_{\text{grp}}$-constraint is satisfiable.*
4. *$\mathcal{P}_{\text{gr}}$ is complete: Given a CLLS$_{\text{gr}}$ constraint φ and a set $\mathcal{G} \subseteq \mathcal{V}ar(\varphi)$, $\mathcal{P}_{\text{gr}}$ can compute from φ any minimal $\mathcal{P}_{\text{gr}}$-saturation for φ with respect to $\mathcal{G}$ in a finite number of steps.*
5. *This set of minimal $\mathcal{P}_{\text{gr}}$-saturations for a CLLS$_{\text{gr}}$ constraint may be infinite.*

Proof. 1. by Lemma 6.6, 2. by Ex. 4.7, 3. by Lemma 6.8, 4. by Lemma 6.11, 5. by Ex. 4.6 and 4.8. □

6.4 Disambiguating Less: A Second Procedure for a Single Beta Reduction Step

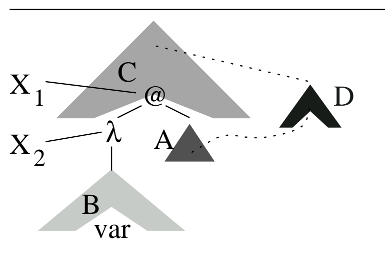


Figure 6.11: An underspecified description of a reducing tree

In the previous section, we have discussed the procedure $\mathcal{P}_{\text{gr}}$ for CLLS_{gr}. It is complete, so it computes all minimal saturations for a constraint containing a beta reduction formula. However, it may not be exactly the procedure we want to use to perform an underspecified beta reduction step. In Sec. 6.3.3 we have illustrated $\mathcal{P}_{\text{gr}}$ on an example constraint, which contained a D segment term, i.e. a segment term in a position that is underspecified between two segment terms of the reducing tree. This situation is depicted in Fig. 6.3, repeated here as Fig. 6.11. To solve the beta reduction literal, $\mathcal{P}_{\text{gr}}$ first had to disambiguate the position of the D segment term. However, we would like to separate the two tasks, on the one hand underspecified beta reduction, on the other hand the enumeration of readings.

In this section we present a second procedure $\mathcal{P}_{\beta}$ for a single beta reduction step that differs from $\mathcal{P}_{\text{gr}}$ in the following ways:

- It includes all *deterministic* saturation rules of $\mathcal{P}_{\text{gr}}$, but none of the distribution rules.
- There are reducing trees for which the procedure cannot compute the reduct. But for a large class of reducing trees, one that is important in the linguistic application, it can compute the reduct without any disambiguation.
- To that end, the procedure $\mathcal{P}_{\beta}$ contains additional deterministic saturation rules that make use of *underspecified correspondence literals* to handle D segment terms.

The main problem in performing a beta reduction step on a partial description of a lambda structure is that we do not always know which part of the reducing tree a variable belongs to. We have referred to this as the problem of "D segment terms". Figure 6.11 shows a schematic view of such a situation: there is a segment D which may belong either to the context C or to the argument A of the reducing tree.

The main idea about $\mathcal{P}_{\beta}$ is that as soon as we are certain that a variable must be in one of $\mathsf{b}(C)$, $\mathsf{b}(B)$, or $\mathsf{b}(A)$ (in the notation of Fig. 6.11), we can copy that variable to the

reduct, even if we do not know yet which of the three segment terms the variable belongs to. That is, we just have to make sure that the variable cannot be above the root of C, and it cannot be X_2 (again in the notation of Fig. 6.11).

The data structure we use for this are *underspecified correspondence literals* of the form u-co$(\{(C_1, D_1), \dots, (C_n, D_n)\})(X){=}Y$, which state that co$(C_i, D_i)(X){=}Y$ holds for some $i, 1 \leq i \leq n$. We define them as literals, not as formulas, because we want to avoid distribution whenever possible. For the same reason, we now redefine some formulas of the previous chapters as literals.

Definition 6.13 (Underspecified correspondence literal). *Let A, B, $A_1, \dots, A_n$, $B_1, \dots, B_n$ be segment terms, and let $(\mathcal{L}^\theta, \sigma)$ be a pair of a lambda structure and a valuation. Then the* correspondence literal

$$co(A, B)(X){=}Y$$

is satisfied by $(\mathcal{L}^\theta, \sigma)$ iff there exists a correspondence function c in $\mathcal{L}^\theta$ between $\sigma(A)$ and $\sigma(B)$, and $c(\sigma(X)) = \sigma(Y)$. The symmetric group parallelism literal

$$(A_1, \dots, A_n) \sim^{\text{sym}} (B_1, \dots, B_n)$$

is satisfied by $(\mathcal{L}^\theta, \sigma)$ iff either $(\sigma(A_1), \dots, \sigma(A_n)) \sim (\sigma(B_1), \dots, \sigma(B_n))$ or $(\sigma(B_1), \dots, \sigma(B_n)) \sim (\sigma(A_1), \dots, \sigma(A_n))$ holds in $\mathcal{L}^\theta$. The underspecified correspondence literal

$$u\text{-}co(\{(A_1, B_1), \dots, (A_n, B_n)\})(X){=}Y$$

is satisfied by $(\mathcal{L}^\theta, \sigma)$ iff three conditions are met: First, for $1 \leq i < j \leq n$, $\mathsf{b}(\sigma(A_i))$ and $\mathsf{b}(\sigma(A_j))$ are disjoint. Second, for all $1 \leq i \leq n$ there exists a correspondence function c_i between $\sigma(A_i)$ and $\sigma(B_i)$. Third, there exists an i, $1 \leq i \leq n$, such that $c_i(\sigma(X)) = \sigma(Y)$.

Like path literals, these literals are used during the computation, but they do not belong to the language that may be used for the input constraint to the procedure.

We redefine the formulas $\text{co}_k(\overline{A}, \overline{B})$ and $\text{co}_{\underline{k}}^{-}(\overline{A}, \overline{B})$ from earlier in this chapter according to the new literals that we now have: Let $\overline{A} = A_1, \dots, A_n$ and $\overline{B} = B_1, \dots, B_n$, then

$$\begin{array}{rcl} \text{co}_k(\overline{A}, \overline{B})(U){=}V & =_{\text{def}} & \text{co}(A_k, B_k)(U){=}V \\ \text{co}_{\underline{k}}^{-}(\overline{A}, \overline{B})(U){=}V & =_{\text{def}} & \text{co}(A_k, B_k)(U){=}V \wedge U \in \mathsf{b}^{-}(A_k) \end{array}$$

for $1 \leq k \leq n$. Furthermore, we use the following formulas for beta reduction steps and underspecified correspondence (with $1 \leq i \leq n$):

$$\begin{array}{rcl} \text{beta}_{X_2,n} & =_{\text{def}} & \text{redtree}_{X_2}(C, B, A) \wedge (C, B, A, \dots, A) \sim (C', B', A', \dots, A') \wedge \\ & & \text{reductlike}(C', B', A'_1, \dots, A'_n) \\ \text{beta}_n & =_{\text{def}} & \text{beta}_{X_2,n} \\ \text{u-co}_i(Z){=}Z' & =_{\text{def}} & \text{u-co}(\{(C, C'), (B, B'), (A, A'_i)\})(Z){=}Z'. \end{array}$$

That is, we use $\text{beta}_{X_2,n}$ when we need to talk about the variable X_2 and beta_n else. Furthermore, with $C = X_0/X_1$ and $B = Y_0/Y_1, \ldots, Y_n$, we use

$$\text{u-co}_i^-(Z){=}Z' =_{\text{def}} \text{u-co}(\{(C,C'),(B,B'),(A,A'_i)\})(Z){=}Z' \wedge Z{\neq}X_1 \wedge \bigwedge_{i=1}^{n} Z{\neq}Y_i.$$

Note that the formulas for *reducing tree* and *reductlike* use subformulas of the form $\mathsf{seg}(A)$, which contain disjunctions. Hence we have to be careful in using these formulas in a framework where we want to restrict indeterministic rules as far as possible. But we will only use these formulas on left rule sides, where they do not present a problem.

6.4.1 A procedure that partially solves CLLS_{gr} constraints: $\mathcal{P}_\beta$

Let $C = X_0/X_1$, $B = Y_0/Y_1, \ldots, Y_n$, and $A = X_3/$.

All rules are parametrized by $1 \le i \le n$.

(UB.init) $(A_1, \ldots, A_m) \sim (B_1, \ldots, B_m) \wedge \mathsf{seg}(A_k) \wedge \mathsf{seg}(B_k)$
$\rightarrow \text{co}(A_k, B_k)(Z_j){=}W_j$
where $1 \le k \le m$, $A_k = Z_0/Z_1, \ldots, Z_{m_k}$, $B_k = W_0/W_1, \ldots, W_{m_k}$, and $0 \le j \le m_k$

(UB.new) $\text{beta}_{X_2,n} \wedge X_0 \lhd^* Z \wedge Z{\neq}X_2 \rightarrow \exists Z'.\text{u-co}_i(Z){=}Z'$

(UB.copy.lab) $\text{beta}_n \wedge Z_0{:}f(Z_1, \ldots, Z_\ell) \wedge \bigwedge_{k=0}^{\ell} \text{u-co}_i(Z_k){=}Z'_k \wedge Z_0{\neq}X_1 \wedge \bigwedge_{j=1}^{n} Z_0{\neq}Y_j$
$\rightarrow Z'_0{:}f(Z'_1, \ldots, Z'_\ell)$

(UB.copy.dom) $\text{beta}_n \wedge \bigwedge_{k=1}^{2} \text{u-co}_i(Z_k){=}Z'_k \wedge Z_1 \lhd^* Z_2 \rightarrow Z'_1 \lhd^* Z'_2$

(UB.copy.ineq) $\text{beta}_n \wedge \bigwedge_{k=1}^{2} \text{u-co}_i(Z_k){=}Z'_k \wedge Z_1{\neq}Z_2 \wedge \bigwedge_{k=1}^{2}(Z_k{\neq}X_1 \wedge \bigwedge_{j=1}^{n} Z_k{\neq}Y_j)$
$\rightarrow Z'_1{\neq}Z'_2$

(UB.λ.in) $\text{beta}_n \wedge \bigwedge_{k=1}^{2} \text{u-co}_i^-(Z_k){=}Z'_k \wedge \lambda(Z_1){=}Z_2 \rightarrow \lambda(Z'_1){=}Z'_2$

(UB.λ.out) $\text{beta}_n \wedge \lambda(Z_1){=}Z_2 \wedge \text{u-co}_i^-(Z_1){=}Z'_1 \wedge (Z_2 \lhd^* X_0 \vee Z_2 \perp X_0) \rightarrow \lambda(Z'_1){=}Z_2$

(UB.λ.1) $\text{beta}_{X_2,n} \wedge \lambda(Z_1){=}Z_2 \wedge \text{u-co}_i^-(Z_2){=}Z'_2$
$\rightarrow X_0 \lhd^* Z_1 \wedge Z_1 \neq X_1 \wedge Z_1 \neq X_2 \wedge \bigwedge_{j=1}^{n} Z_1 \neq Y_j$

(UB.λ.2) $\text{beta}_n \wedge \lambda(Z_1){=}Z_2 \wedge \text{u-co}_i^-(Z_1){=}Z'_1 \rightarrow X_0 \lhd^* Z_2 \vee Z_2 \lhd^+ X_0 \vee Z_2 \perp X_0$

(UB.λ.inverse) $\text{beta}_1 \wedge \lambda^{-1}(Z_0){=}\{Z_1, \ldots, Z_m\} \wedge$
$\bigwedge_{k=0}^{m} \text{u-co}_1(Z_k){=}Z'_k \rightarrow \lambda^{-1}(Z'_0){=}\{Z'_1, \ldots, Z'_m\}$ redex linear

(UB.co.path) $\text{co}(D, D')(U){=}V \rightarrow \text{p}\left(\begin{smallmatrix} Z_0 & W_0 \\ U & V \end{smallmatrix}\right)$ where $D = Z_0/\ldots$, $D' = W_0/\ldots$

(UB.co.corr)	$\mathrm{p}(\begin{smallmatrix} Z_0 & W_0 \\ U & V \end{smallmatrix}) \wedge (A_1, \ldots, A_m) \sim^{\mathrm{sym}} (B_1, \ldots, B_m) \wedge U \in \mathsf{b}(A_k) \rightarrow \mathrm{co}(A_k, B_k)(U){=}V$ where $1 \leq k \leq m$, $A_k = Z_0/\ldots$, $B_k = W_0/\ldots$
(UB.co.u-corr)	$\mathrm{beta}_n \wedge \mathrm{co}(D, D')(U){=}V \rightarrow \mathrm{u\text{-}co}_i(U){=}V$ where $(D, D') \in \{(C, C'), (B, B'), (A, A'_i)\}$
(UB.sym)	$\overline{A} \sim \overline{B} \rightarrow \overline{A} \sim^{\mathrm{sym}} \overline{B} \wedge \overline{B} \sim^{\mathrm{sym}} \overline{A}$ where $\overline{A} = A_1, \ldots, A_m$, $\overline{B} = B_1, \ldots, B_m$
(D.diffParent)	$Z{:}f(Z_1, \ldots, Z_m) \wedge W{:}g(W_1, \ldots, W_\ell) \wedge Z{\neq}W \rightarrow \bigwedge_{j=1}^{m} \bigwedge_{k=1}^{\ell} Z_j{\neq}W_k$
(D.diffChild)	$Z{:}f(Z_1, \ldots, Z_m) \wedge W{:}f(W_1, \ldots, W_m) \wedge Z_j{\neq}W_j \rightarrow Z{\neq}W$ where $1 \leq j \leq m$

plus all rules of the semi-decision procedure $\mathcal{P}_{\mathrm{gr}}$ for $\mathrm{CLLS}_{\mathrm{gr}}$ in Sec. 6.3.1 and 5.6 that are *deterministic* if correspondence and symmetric group parallelism are literals rather than formulas, plus (D.λ.lam) and (D.λ.distr.inv).

6.4.2 The Rules in Detail

Section 6.4.1 shows the procedure $\mathcal{P}_\beta$. The first block of rules are the core rules dealing with group parallelism literals, but this time geared to the special case of beta reduction formulas. The main idea is to copy all material from the reducing tree to the reduct, except the part between the hole of the context and the roots of body and argument, and except the parts below the holes of the body. The rule (UB.init), which is $(A_1, \ldots, A_m) \sim (B_1, \ldots, B_m) \wedge \mathsf{seg}(A_k) \wedge \mathsf{seg}(B_k) \rightarrow \mathrm{co}(A_k, B_k)(Z_j){=}W_j$, for $1 \leq k \leq m$, $A_k = Z_0/Z_1, \ldots, Z_{m_k}$, $B_k = W_0/W_1, \ldots, W_{m_k}$, and $0 \leq j \leq m_k$, states that in two parallel segment terms, the two roots correspond, and matching holes correspond as well. The difference between this rule and (GP.init) is that (UB.init) demands $\mathsf{seg}(A_k)$ and $\mathsf{seg}(B_k)$ in the premise, while (GP.init) states it in the conclusion (which makes it a distribution rule). The rule (UB.new), of the form $\mathrm{beta}_{X_2,n} \wedge X_0 \triangleleft^* Z \wedge Z{\neq}X_2 \rightarrow \exists Z'.\mathrm{u\text{-}co}_i(Z){=}Z'$, states that all variables of the reducing tree have a correspondent in the reduct except X_2 (which is the lam-labeled variable between C and B, see Fig. 6.11). The rule (UB.copy.lab) has the form $\mathrm{beta}_n \wedge Z_0{:}f(Z_1, \ldots, Z_\ell) \wedge \bigwedge_{k=0}^{\ell} \mathrm{u\text{-}co}_i(Z_k){=}Z'_k \wedge Z_0{\neq}X_1 \wedge \bigwedge_{j=1}^{n} Z_0{\neq}Y_j \rightarrow Z'_0{:}f(Z'_1, \ldots, Z'_\ell)$. It states that all labeling literals of the reducing tree reappear in the reduct, except the labels of X_1, X_2 and Y_1 through Y_n. (UB.copy.dom) says that all dominance literals of the reducing tree must also hold between the corresponding variables in the reduct: $\mathrm{beta}_n \wedge \bigwedge_{k=1}^{2} \mathrm{u\text{-}co}_i(Z_k){=}Z'_k \wedge Z_1 \triangleleft^* Z_2 \rightarrow Z'_1 \triangleleft^* Z'_2$. The rule (UB.copy.ineq) copies inequalities. All inequalities from the reducing tree are copied to the reduct, except the following: The correspondent of X_1 and the correspondent of B's root are equal in the reduct. And the correspondent of Y_i and the root of the i-th argument copy are equal in the reduct. (UB.copy.ineq) expresses this by stating

$\text{beta}_n \wedge \bigwedge_{k=1}^{2} \text{u-co}_i(Z_k){=}Z'_k \wedge Z_1{\neq}Z_2 \wedge \bigwedge_{k=1}^{2}(Z_k{\neq}X_1 \wedge \bigwedge_{j=1}^{n} Z_k{\neq}Y_j) \to Z'_1{\neq}Z'_2$.

The rule (UB.λ.in), which is $\text{beta}_n \wedge \bigwedge_{k=1}^{2} \text{u-co}_i^-(Z_k){=}Z'_k \wedge \lambda(Z_1){=}Z_2 \to \lambda(Z'_1){=}Z'_2$, copies lambda binding literals from the reducing tree to the reduct. The rule (UB.λ.out), of the form $\text{beta}_n \wedge \lambda(Z_1){=}Z_2 \wedge \text{u-co}_i^-(Z_1){=}Z'_1 \wedge (Z_2{\triangleleft^*}X_0 \vee Z_2{\perp}X_0) \to \lambda(Z'_1){=}Z_2$, handles the case of a lambda binder outside the reducing tree. The rule (UB.λ.1), of the form $\text{beta}_{X_2,n} \wedge \lambda(Z_1){=}Z_2 \wedge \text{u-co}_i^-(Z_2){=}Z'_2 \to X_0{\triangleleft^*}Z_1 \wedge Z_1 \neq X_1 \wedge Z_1 \neq X_2 \wedge \bigwedge_{j=1}^{n} Z_1 \neq Y_j$, states that if the lambda binder is inside the reducing tree, then so is any bound variable. The rule (UB.λ.2) is a distribution rule, of the form $\text{beta}_n \wedge \lambda(Z_1){=}Z_2 \wedge \text{u-co}_i^-(Z_1){=}Z'_1 \to X_0{\triangleleft^*}Z_2 \vee Z_2{\triangleleft^+}X_0 \vee Z_2{\perp}X_0$: Given a bound variable in the reducing tree, it guesses whether the binder is in the reducing tree too. The rule (UB.λ.inverse) is $\text{beta}_1 \wedge \lambda^{-1}(Z_0){=}\{Z_1,\ldots,Z_m\} \wedge \bigwedge_{k=0}^{m} \text{u-co}_1(Z_k){=}Z'_k \to \lambda^{-1}(Z'_0){=}\{Z'_1,\ldots,Z'_m\}$. It copies inverse lambda binding literals, i.e. the information that all variables bound by a lambda binder are known. It is a special case of (GP.λ.inverse). The segments of a beta reduction literal, those in the reducing tree group as well as those in the reduct group, do not overlap properly, so we need not consider the possibilities that any correspondents of $Z_1,\ldots,Z_m$ might be bound somewhere else than at the correspondent of Z_0. Note that the rule (UB.λ.inverse) is restricted to *linear* redexes: In a linear redex there is exactly one occurrence of the bound object-level variable. For the nonlinear case, we have to take recourse to disambiguation. In section 6.6 we will discuss an example that shows why nonlinear redexes are a problem.

Now that we have correspondence literals rather than correspondence formulas, we have to make the connection between them and path literals explicit by saturation rules: The rules (UB.co.path) and (UB.co.corr), which are $\text{co}(D,D')(U){=}V \to \text{p}(\begin{smallmatrix} Z_0 & W_0 \\ U & V \end{smallmatrix})$ for $D = Z_0/\ldots$, $D' = W_0/\ldots$ and $\text{p}(\begin{smallmatrix} Z_0 & W_0 \\ U & V \end{smallmatrix}) \wedge (A_1,\ldots,A_m) \sim^{\text{sym}} (B_1,\ldots,B_m) \wedge U \in \mathsf{b}(A_k) \to \text{co}(A_k,B_k)(U){=}V$ for $1 \leq k \leq m$, $A_k = Z_0/\ldots$, $B_k = W_0/\ldots$, describe this connection. The rule (UB.co.u-corr), of the form $\text{co}(D,D')(U){=}V \to \text{u-co}_i(U){=}V$ for $(D,D') \in \{(C,C'),(B,B'),(A,A'_i)\}$, infers underspecified correspondence literals from normal ones. Note that if $D' = A'_i$, then we can only infer $\text{u-co}_i(U){=}V$ for that same i. Because of this rule, we can use the (UB.copy...) rules to copy information for variables for which we know which segment term they are in, as well as for variables for which we only have underspecified correspondence literals. The rule (UB.sym) is $\overline{A} \sim \overline{B} \to \overline{A} \sim^{\text{sym}} \overline{B} \wedge \overline{B} \sim^{\text{sym}} \overline{A}$. Since we now have $\sim^{\text{sym}}$ literals rather than formulas, we make the connection between them and group parallelism literals explicit in this rule.

The rules (D.diffParent), which is $Z{:}f(Z_1,\ldots,Z_m) \wedge W{:}g(W_1,\ldots,W_\ell) \wedge Z{\neq}W \to \bigwedge_{j=1}^{m} \bigwedge_{k=1}^{\ell} Z_j{\neq}W_k$, and (D.diffChild), which is $Z{:}f(Z_1,\ldots,Z_m) \wedge W{:}f(W_1,\ldots,W_m) \wedge Z_j{\neq}W_j \to Z{\neq}W$ for $1 \leq j \leq m$, infer additional inequalities: Two inequal parent variables must have inequal child variables, and vice versa. We will need these rules to trigger (UB.new).

Design decisions. The procedure $\mathcal{P}_\beta$ contains three distribution rules: the rule (UB.λ.2), which lets us choose between (UB.λ.in) and (UB.λ.out) for copying lambda

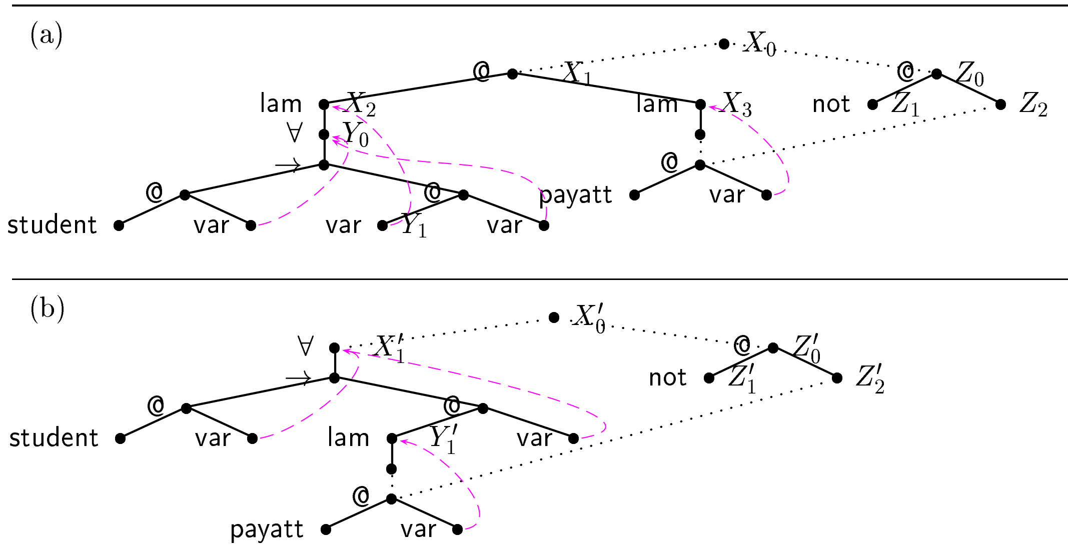


Figure 6.12: A simple beta reduction step: "Every student does not pay attention"

binding literals; the rule (D.λ.lam), which determines the label of a binder; and the rule (D.λ.distr.inv), which enforces compliance with an inverse lambda binding literal. Why do we allow these distribution rules, and omit all others?

The idea is that the procedure $\mathcal{P}_\beta$, which focuses on beta reduction steps, has to make decisions concerning lambda binding, because we may need the information on lambda binding for subsequent beta reduction steps; all other disambiguation rules, which may lead to an enumeration of readings in a constraint from the linguistic application, are avoided. In particular, the procedure $\mathcal{P}_\beta$ does not make decisions about anaphoric binding literals, nor does it disambiguate scope ambiguities like the one in Fig. 2.8, p. 34.

6.4.3 Examples

Figure 6.4 (a), repeated (and extended by the root variable of the context) in Fig. 6.12 (a), shows the constraint that represents the semantics of the sentence "Every student does not pay attention." This constraint in picture (a) is the reducing tree. To perform a single underspecified beta reduction step, we extend it by

$$\begin{array}{l}(C, B, A) \to^\beta (C', B', A'_1) \wedge X_0 \bot X'_0 \text{ with} \\ C = X_0/X_1, \quad B = Y_0/Y_1, \quad A = X_3/, \\ C' = X'_0/X'_1, \quad B' = X'_1/Y'_1, \quad A'_1 = Y'_1/\end{array}$$

for new variables X'_0, X'_1, Y'_1. Actually, this is inexact: Instead of the disjunction $\mathsf{seg}(C)$ that is part of $\text{redtree}_{X_2}(C, B, A)$ (which is itself part of the beta reduction formula) we add $X_0 \triangleleft^* X_1 \wedge X_1{=}X_1$, and likewise for all other $\mathsf{seg}(.)$ formulas.

Now the procedure $\mathcal{P}_\beta$ can work as shown in Fig. 6.13. To be able to apply (UB.new) and the (UB.copy...) rules, we have to derive inequalities: For (UB.new) we have to

(1)	$Z_0 \neq X_2$	(D.lab.ineq)	(7)	$Z_0 \neq Y_1$	(D.lab.ineq)
(2)	$Z_1 \neq X_2$	(D.lab.ineq)	(8)	Z'_0:@(Z'_1, Z'_2)	(UB.lab)
(3)	$Z_2 \neq X_2$	(D.diffChild)	(9)	$Z_1 \neq X_1$	(D.lab.ineq)
(4)	$\exists Z'_i$.u-co$_1(Z_i)=Z'_i$ $i = 0, 1, 2$	(UB.new)	(10)	$Z_1 \neq Y_1$	(D.lab.ineq)
			(11)	Z'_1:not	(UB.copy.lab)
(5)	$Z_1 \neq X_2$	(D.lab.ineq)	(12)	$X'_0 \triangleleft^* Z'_0$	(UB.copy.dom)
(6)	$Z_0 \neq X_1$	(D.diffParent)			

Figure 6.13: Computation of $\mathcal{P}_\beta$ on Fig. 6.12.

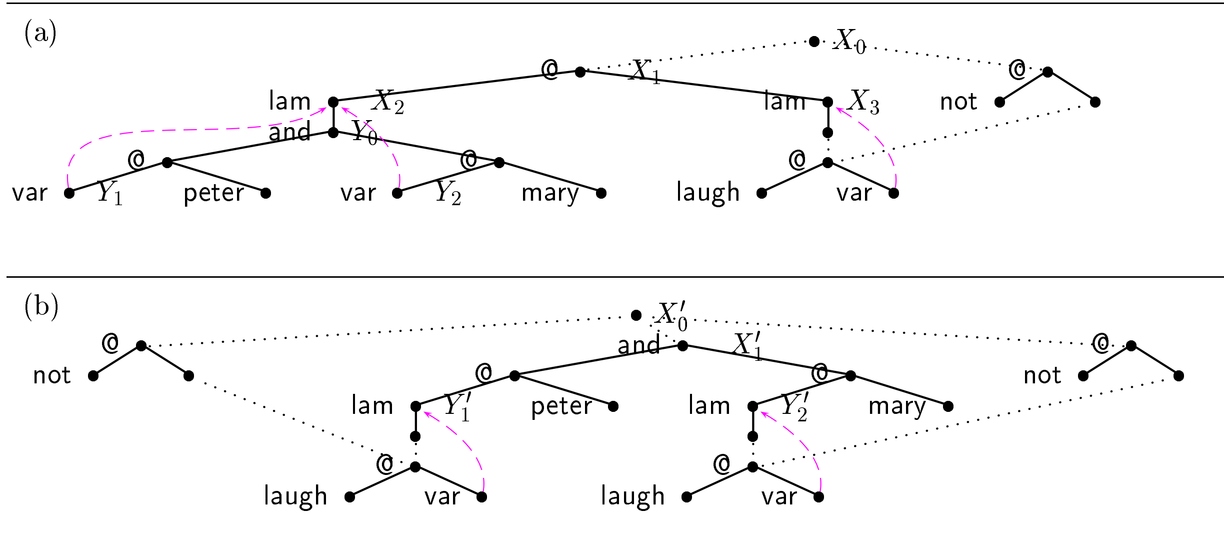


Figure 6.14: A nonlinear redex: Constraint for "Peter and Mary do not laugh."

show that Z_0, Z_1, Z_2 in Fig. 6.12 (a) are not the variable X_2 between the context and the body. And for (UB.copy.lab) we have to show that they are not equal to a hole of the context or the body segment term. Figure 6.13 only shows the part of the computation that pertains to Z_0, Z_1, Z_2. For all other variables in the reducing tree, it is known inside which segment term they are. So to them the "normal" group parallelism literal rules (of Sec. 6.3.1) apply. The result of the computation is the constraint in Fig. 6.12 (a) plus the constraint in Fig. 6.12 (b).

Figure 6.14 (a) shows the constraint that represents the semantics of the sentence "Peter and Mary do not laugh." This constraint contains a *nonlinear* redex: The lambda binder X_2 binds two occurrences of the object-level variable, at Y_2 and Y_3. We extend the constraint in (a) by

$$
\begin{aligned}
&(C, B, A) \rightarrow^\beta (C', B', A'_1, A'_2) \wedge X_0 \perp X'_0 \text{ with} \\
&C = X_0/X_1, \quad B = Y_0/(Y_1, Y_2), \quad A = X_3/, \\
&C' = X'_0/X'_1, \quad B' = X'_1/(Y'_1 Y'_2), \quad A'_1 = Y'_1/, A'_2 = Y'_2/
\end{aligned}
$$

for new variables X_0', X_1', Y_1', Y_2'. Again, this is inexact, since we do not add disjunctions for the $\mathsf{seg}(.)$ formulas: In the case of B we add $Y_0 \lhd^* Y_i \wedge Y_i{=}Y_i \wedge Y_1 \perp Y_2$ for $1 = 1, 2$, and likewise for B'.

From this constraint, the procedure $\mathcal{P}_\beta$ computes the constraint in Fig. 6.14 (a) plus (b): As the redex is 2-ary, the rules produce two copies of the negation fragment, one via u-co$_1$ and one via u-co$_2$. We return to this example in a minute.

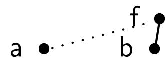
6.4.4 Properties of the Procedure $\mathcal{P}_\beta$

Soundness. As we have defined in Def. 3.3 (p. 62), we call a saturation rule sound iff it is an equivalence transformation. And as we have remarked there, it suffices to show that the left-hand side of the rule entails the right-hand disjunction because we are working in a saturation framework.

(UB.init) is sound for the same reason as (P.init) is (see Sec. 4.2.1, p. 89). The u-co$_i$ literals in (UB.new) through (UB.copy.ineq) are fulfilled because $\text{redtree}_{X_2}(C, B, A)$ is part of the formula beta_n: The segments interpreting C, B, A do not share any nodes. Concerning (UB.new), if X_0 is the root variable of C and X_2 the lam-labeled variable that is the child of C's hole, then all variables that are inside $\mathsf{b}(X_0/)$ and inequal to X_2 must denote a node in either the context, the body or the argument segment of the reducing tree. According to the definition of correspondence functions (Def. 2.3, p. 27), (UB.copy.lab) must additionally exclude the holes of B and C before copying labeling literals. Similar arguments yield the soundness of the rules (UB.copy.dom) and (UB.copy.ineq). The rule (UB.λ.in) is sound by the conditions (β.λ.same) and (β.λ.diff) of the beta reduction relation (Def. 6.2, p. 142). (UB.λ.1) is sound because a lambda binder must dominate the variables it binds, so if we have $\lambda(Z_1){=}Z_2$, and $Z_2 \in \mathsf{b}(X_0/)$ and $Z_2{\neq}X_2$ are in φ, then Z_1 must be dominated by X_0 and must not be the hole of either C, B or A by condition (gp.λ.hang) of the group parallelism relation (Def. 6.3, p. 144). (UB.λ.2) is obviously sound. (UB.λ.inverse) is a simplified version of (GP.λ.inverse): (UB.λ.inverse) is $\text{beta}_1 \wedge \lambda^{-1}(Z_0){=}\{Z_1, \ldots, Z_m\} \wedge \bigwedge_{k=0}^{m} \text{u-co}_1(Z_k){=}Z_k' \;\rightarrow\; \lambda^{-1}(Z_0'){=}\{Z_1', \ldots, Z_m'\}$. (GP.$\lambda$.inverse) is $\lambda^{-1}(X){=}S_1 \wedge \text{co}_k^-(\overline{A}, \overline{B})(X){=}Y \wedge \text{co}^-(\overline{A}, \overline{B})(S_1){=}S_2 \cup S_3 \wedge \bigwedge_{V \in S_2} \lambda(V){=}Y \wedge \bigwedge_{V \in S_3} \lambda(V){\neq}Y \rightarrow \lambda^{-1}(Y){=}S_2$. Since the redex in (UB.λ.inverse) is linear, Z_0 will have exactly one correspondent in the reduct, as will $Z_1, \ldots, Z_m$. Furthermore all of the correspondents of $Z_1, \ldots, Z_m$ have to be bound at the correspondent of Z_0 by (UB.λ.in). So the distinction of S_2 and S_3 which is necessary in (GP.λ.inverse) can be omitted in (UB.λ.inverse) because S_3 will always be empty. Returning to the question of soundness: We have shown the soundness of (GP.λ.inverse) in Lemma 6.5, so (UB.λ.inverse) is sound as well. (UB.co.path) and (UB.co.corr) spell out the connection between correspondence literals and their previous definition as a formula, and (UB.sym) does the same for symmetric group parallelism literals. (UB.co.u-corr) is sound because the context, body, and argument segments of the reducing tree must be disjoint. (D.diffParent) and (D.diffChild) are obviously sound.

Saturated constraints. It is not easy to describe what a $\mathcal{P}_\beta$-saturated constraint looks like. It need not be saturated under $\mathcal{P}_{\text{gr}}$ because in $\mathcal{P}_\beta$ we have left out a number of rules. The procedure $\mathcal{P}_\beta$ may have partially solved the beta reduction formulas occurring in the input constraint. Below we describe conditions under which the procedure can solve beta reduction formulas.

Fairness. Above (p. 155) we have stated a fairness condition for the procedure $\mathcal{P}_{\text{gr}}$. For the procedure $\mathcal{P}_\beta$, we extend this condition canonically by treating (UB.new) the same way as (P.new) and (GP.new).

Satisfiability of saturated constraints? While the procedure $\mathcal{P}_{\text{gr}}$ checks a constraint for satisfiability and enumerates readings for a linguistically relevant constraint, the procedure $\mathcal{P}_\beta$ is concerned just with beta reduction formulas. Accordingly, a $\mathcal{P}_\beta$-saturated constraint need not necessarily be satisfiable. In the procedure $\mathcal{P}_\beta$ we have left out, among other rules, (D.distr.child), which means for example that $\mathcal{P}_\beta$ cannot detect the unsatisfiability of this constraint: 

Which constraints can the procedure handle? The procedure $\mathcal{P}_\beta$ allows us to perform an underspecified beta reduction step for many examples from underspecified natural language semantics without any disambiguation. Why is this so? We now take a closer look at the constraints that occur in the linguistic application.

The semantic representation of a sentence is constructed according to the syntactic structure of the sentence, in the syntax/semantics interface. A recent overview paper on CLLS by Egg, Koller and Niehren [41] describes the syntax/semantics interface currently in use with CLLS. With this semantic construction, a CLLS constraint representing a sentence's semantics has the following properties: The constraint possesses a root variable as defined in Def. 3.7, p. 64: a variable that dominates all others. The constraint does not contain any disjointness literals $X \perp Y$. For all lam-labeled variables X, there exists an inverse lambda binding literal $\lambda^{-1}(X)=\{\ldots\}$. If the constraint contains labeling literals $X{:}f(\ldots)$ and $Y{:}g(\ldots)$ for two distinct variables X, Y, then it also contains $X \neq Y$, independent of whether f and g are the same symbol. And finally, the constraint graph has no "empty fragments", fragments that consist of a single node.

To perform a single underspecified beta reduction step, we extend such a constraint as follows[4]:

- We add a beta reduction formula $(C, B, A) \to^\beta (C', B', A'_1, \ldots, A'_n)$ for appropriate segment terms C for the context, B for the body, and A for the argument. This

[4]A discussion of a procedure for underspecified beta reduction that automates this step follows in the next section.

beta reduction formula also states that any two distinct holes of the body segment term B are disjoint: If $B = Y_0/Y_1, \ldots, Y_n$ then we also add $Y_i \perp Y_j$ for $1 \leq i < j \leq n$.

- The variables that form the reduct segment terms $C', B', A'_1, \ldots, A'_n$ are all fresh.
- If Z is the root variable of the constraint and $C' = X'_0/\ldots$, then we also add $X'_0 \perp Z$.

What can $\mathcal{P}_\beta$ compute for such a constraint φ? Suppose $C = X_0/X_1$, $B = Y_0/Y_1, \ldots, Y_n$, and X_1 is the parent of X_2 like in Fig. 6.11. Now first, if we have a variable U with $X_0 \triangleleft^* U \in \varphi$, then $\mathcal{P}_\beta$ can derive either $U{=}X_2$ or $U{\neq}X_2$ using the fact that all variables U_1, U_2 with $U_1{:}f(\ldots)$, $U_2{:}g(\ldots)$ are inequal in φ, the fact that we have no empty fragments, and the rule (D.diffParent). For such a variable U we can further derive either $U{=}X_1$ or $U{\neq}X_1$, and $U{=}Y_i$ or $U{\neq}Y_i$ for $1 \leq i \leq n$, in the same way. This is all the information that is needed for the (UB.copy...) rules of $\mathcal{P}_\beta$.

For all variables in the reducing tree that have a lambda binder, we additionally need to know whether (UB.λ.in) applies or (UB.λ.out). This is determined by (UB.λ.2). So we can copy lambda binding literals from the reducing tree to the reduct. But how about inverse lambda binding literals? The rule (UB.λ.inverse) is restricted to *linear* redexes, in which there is exactly one occurrence of the bound object-level variable. So we can copy inverse lambda binding literals with $\mathcal{P}_\beta$ only if the redex that we are working with is linear. Otherwise we need to fall back on the procedure $\mathcal{P}_{\mathrm{gr}}$ to disambiguate the position of the lambda binder and the bound variables. But note that this does not mean that we cannot handle nonlinear redexes at all: The second example for $\mathcal{P}_\beta$ that we have discussed in Sec. 6.4.3 contained a nonlinear redex, and $\mathcal{P}_\beta$ could handle it without any problem. This was due to the fact that in this case the D segment term did not contain any lambda binder or bound variable.

Since the reducing tree and the reduct in φ are disjoint, and the variables of the reduct are not involved in any (group) parallelism literals except the one introduced by the beta reduction formula, we can say in advance how many new variables $\mathcal{P}_{\mathrm{gr}}$ will add at most to compute the reduct: as many variables as X_0 dominates in φ. So if the constraint φ contains no further (group) parallelism literals that could cause trouble (or if we have temporally removed any such literals), $\mathcal{P}_\beta$ terminates on φ.

6.5 Underspecified Beta Reduction

Underspecified beta reduction means performing a series of beta reduction steps on an underspecified description of a lambda term, with the aim of achieving a description of a first-order term. This task can be broken up into two parts, each handled by a different procedure: on the one hand a procedure that performs a single beta reduction step (i.e. solves a beta reduction formula), and on the other hand a procedure that performs multiple beta reduction steps one after the other, in each step identifying a redex, constructing a suitable beta reduction formula and calling the first procedure to have the constraint solved.

$\mathcal{UB}_{\mathcal{P}roc}(\varphi, Z, X) =$
  if all reducing trees in φ below X are reduced then return (φ, X)
  else
    pick a reducing tree $\text{redtree}_{X_2}(C, B, A)$ in φ that is unreduced,
      a reducing tree with an n-ary redex such that X is the root variable of C
    add $\varsigma = (C, B, A) \to^{\beta} (C', B', A'_1, \ldots, A'_n)$ to φ
      where $C', B', A'_1, \ldots, A'_n$ are segment terms consisting of fresh variables
    let $C' = X'_0/\ldots$, then
      add $\varsigma' = Z'{:}f(Z, X'_0)$ to φ for a fresh variable Z'
    for all $\mathcal{P}$roc-solved forms φ' of $\varphi \wedge \varsigma \wedge \varsigma'$ do $\mathcal{UB}_{\mathcal{P}roc}(\varphi', Z', X'_0)$
  end

Figure 6.15: Underspecified beta reduction

In the current section we present a procedure for the second part of the task. It knows a "current term" within the constraint, within which it identifies a reducing tree. It adds a suitable beta reduction formula to the constraint and solves it using either $\mathcal{P}_{\text{gr}}$ or $\mathcal{P}_{\beta}$. This yields a new "current term", the reduct. So the whole constraint represents a chain of lambda terms that arise during the beta reduction process, and the "current term" is the latest lambda term in this chain.

This procedure $\mathcal{UB}$ for underspecified beta reduction is shown in Figure 6.15. It is parametrized by a procedure $\mathcal{P}$roc for computing the result of a single beta reduction step. As arguments, $\mathcal{UB}$ takes a constraint φ, a root variable $Z \in \mathcal{V}ar(\varphi)$ of the whole constraint φ, and a variable X which is the root variable of the "current term". For example for the constraint in Fig. 6.12 (a) both the root variable Z and the current term root variable X would be X_0.

The procedure selects an unreduced redex within the "current term". By this we mean a reducing tree (C, B, A) such that $(C, B, A) \to^{\beta} (C', B', A'_1, \ldots, A'_n)$ is *not* in φ for any segment terms $C', B', A'_1, \ldots, A'_n$ and a suitable n. Once such an unreduced reducing tree $\text{redtree}_{X_2}(C, B, A)$ is found, the procedure $\mathcal{UB}$ adds a description of the reduct, constructed out of fresh variables, and forces the reduct to be a disjoint position from all of φ by adding the labeling literal $Z'{:}f(Z, X'_0)$ for a new root variable Z'.

To this constraint the procedure $\mathcal{P}$roc is applied to solve the beta reduction formula. This procedure can be either $\mathcal{P}_{\text{gr}}$ or $\mathcal{P}_{\beta}$. Finally, $\mathcal{UB}$ calls itself recursively with each new constraint that $\mathcal{P}$roc has computed. The new root variable is Z', and the new current term starts at X'_0, the root variable of the reduct.

6.6 Discussion: Nonlinear Redexes

The rule (UB.λ.inverse) requires the redex of the reducing tree under consideration to be *linear*, with exactly one occurrence of the bound object-level variable. So what does a constraint look like that this rule cannot handle? Figure 6.16 (a) shows such a constraint.

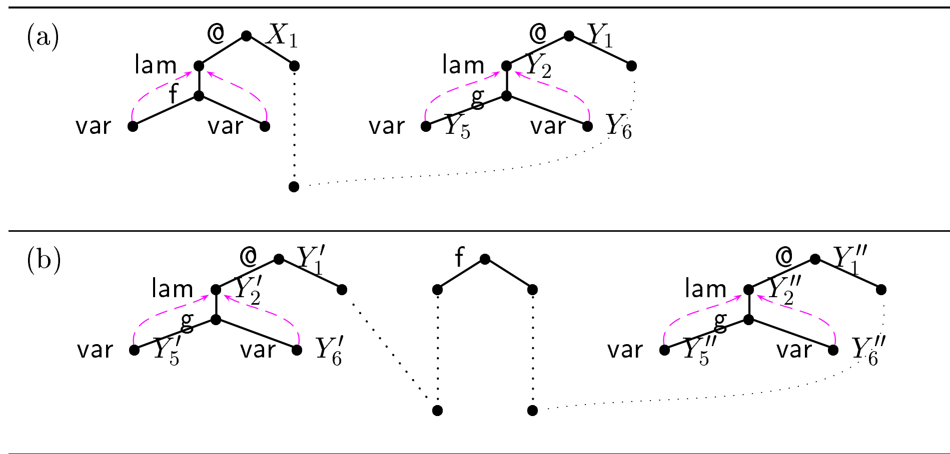


Figure 6.16: A problematic nonlinear redex

There is a reducing tree starting at X_1, with two occurrences of the bound object-level variable, and there is a D segment term starting at Y_1. This segment term again contains a lambda binder, Y_2, which binds two occurrences of its object-level variable.

Figure 6.16 (b) shows the result of a beta reduction step at the reducing tree that starts at X_1. This reduct contains two copies of the D segment term starting at Y_1. Now suppose the constraint shown in (a) contains $\lambda^{-1}(Y_2)=\{Y_5, Y_6\}$ for the lambda binder Y_2 within the D segment term. Then what is $\lambda^{-1}(Y_2')$? This actually depends on whether Y_2 is inside the context or the argument segment term of the reducing tree in (a). If Y_2 is inside the context, then Y_2' and Y_2'' denote the same node, and $\lambda^{-1}(Y_2')=\{Y_5', Y_6', Y_5'', Y_6''\}$, but if Y_2 is inside the argument segment term, then Y_2' and Y_2'' lie at disjoint positions, and $\lambda^{-1}(Y_2')=\{Y_5', Y_6'\}$.

There is another interesting problem with this constraint. Suppose that we do have inverse lambda binding literals for Y_2' and Y_2'' in picture (b), and that we want to do a second beta reduction step, this time for the reducing tree starting at Y_1'. Then the reduct of this beta reduction step will contain *two* copies of the fragment starting at Y_1''. Each of these copies contains another reducing tree (again supposing that we get the inverse lambda binding literals from somewhere). And if we then perform another beta reduction step at one of these reducing trees, this gives us two copies of the other copy, and so on at infinitum.

Both problems disappear if we disambiguate the position of the D segment term. But how can we solve these problems without enumerating readings? I think a solution to this problem would be to put up a constraint stating that the redexes at Y_1' and Y_1'' have to be reduced *simultaneously*. It is easy to list all the variables bound *by* Y_2' *and* Y_2'' *together*: we must have $\lambda^{-1}(Y_2') \cup \lambda^{-1}(Y_2'')=\{Y_5', Y_6', Y_5'', Y_6''\}$. Furthermore, if we reduce at both redexes at the same time, we can make sure that neither the fragment starting at Y_1' nor the one starting at Y_1'' gets copied to the reduct more than once.

6.7 Summary

Underspecified beta reduction is beta reduction on partial descriptions of lambda terms. While a naive rewriting approach may generate spurious solutions, a sound approach is to give a declarative description of the result of the rewriting step using parallelism constraints.

We have defined the *beta reduction relation*, which holds between the reducing tree and the reduct. Furthermore we have introduced the *group parallelism relation*, a generalization of the parallelism relation that relates two *groups* of segments instead of two single segments, the only difference being the conditions on binding. We have shown that the beta reduction relation can be expressed by the group parallelism relation plus conditions detailing the relative position of the context, body, and argument segments in the reducing tree, and likewise of the context, body and argument segments in the reduct. Accordingly, we have defined the language CLLS_{gr}, which extends the language CLLS by group parallelism literals and inverse lambda binding literals. In this language, a beta reduction formula expresses that a beta reduction relationship holds.

We have presented a sound and complete procedure $\mathcal{P}_{\text{gr}}$ for CLLS_{gr}. This procedure can also compute the result of a single beta reduction step.

However, performing underspecified beta reduction and enumerating readings are two separate tasks, and we would like to keep them separate. So we have introduced a second procedure $\mathcal{P}_{\beta}$ for solving beta reduction formulas, which can perform an underspecified beta reduction step without any disambiguation for many examples from underspecified semantics. This is made possible by the specific layout of the segment terms in a reducing tree, along with underspecified correspondence literals.

Finally we have discussed a procedure for underspecified beta reduction. It identifies a reducing tree in the current term to be considered, generates a suitable beta reduction formula, and calls either $\mathcal{P}_{\text{gr}}$ or $\mathcal{P}_{\beta}$ to solve it. This yields a new current term under consideration, which may again be reduced.

Chapter 7

Modeling Ellipsis with Group Parallelism and Jigsaw Parallelism

The current chapter focuses on *modeling ellipsis*: We look at problems for modeling ellipsis with parallelism constraints, and we show how these problems can be solved.

- Up to now we have assumed that the semantics of a contrasting element is a subtree of the lambda structure. But there are cases where the semantics of a contrasting element forms a *segment* rather than a subtree. To put it differently: In these cases, a *group* of tree segments for the source sentence semantics are structurally isomorphic to a group of tree segments for the target sentence semantics.

  This problem can be solved using *group parallelism* rather than normal parallelism for modeling ellipsis. Group parallelism is a canonical extension to parallelism that we have introduced in the previous chapter, in the context of underspecified beta reduction. The difference between parallelism and group parallelism lies in the conditions on binding, where the group parallelism conditions on binding are more permissive.

- The semantic contribution of a contrasting element may partake in scope ambiguities. In some cases this means that a *disjunction* of group parallelism literals is needed to model the meaning of an elliptical sentence, because the position of the contrasting element semantics is not sufficiently specified. This is unsatisfactory: We want our modeling language to be flexible enough to model ellipsis without explicit disjunction.

  We solve this problem by introducing *jigsaw parallelism.* While the jigsaw parallelism *relation* does not add any expressive power with respect to the group parallelism relation, a jigsaw parallelism *literal* may express a disjunction of group parallelism literals.

7.1 The Phenomenon

In this section we discuss ellipsis cases that can only be modeled by a language that goes beyond normal parallelism constraints.

7.1.1 Modeling Ellipsis with Parallelism Constraints

We briefly recall the way we model ellipsis with parallelism constraints. Consider again the simple elliptical sentence (7.1), repeated from (2.5).

(7.1) Every man sleeps, and so does Mary.

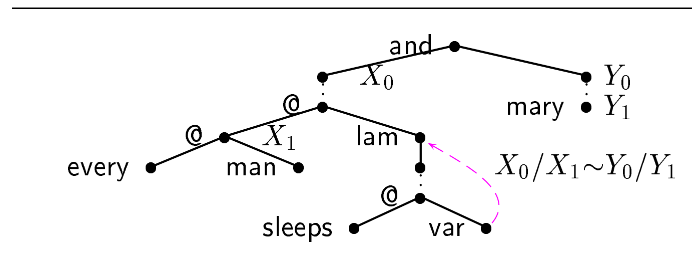


Figure 7.1: Constraint for "Every man sleeps, and so does Mary."

We represent the meaning of the sentence by the constraint in Fig. 7.1, repeated from Fig. 2.10: The meaning of the source sentence "every man sleeps" (represented by the part of the constraint graph dominated by X_0) is the same as the meaning of the target sentence (represented by the part of the constraint graph dominated by Y_0), except for the contributions of the two subjects "every man" and "Mary". In each model of the constraint, the segment denoted by X_0/X_1 must be structurally isomorphic to the one denoted by Y_0/Y_1.

7.1.2 Problematic Cases

Consider the elliptical sentences (7.2) through (7.6). Sentence (7.2) has two pairs of contrasting elements: On the one hand "George" in the target sentence contrasts "Dan" in the source sentence; on the other hand the "not" in the target sentence contrasts to an empty contrasting element in the source sentence. So, the source sentence semantics except for the contribution of "Dan" has the same structure as the target sentence semantics except for the contributions of "George" and "not". Seen as a lambda structure, the semantics of this sentence follows the general schema shown in Fig. 7.2 (b): The semantics of "not" is sketched as the deeper shaded segment within the right segment. The remainder of the right segment, the segments α_1 and α_2, put together have the same structure as the segment α' in the left of picture (b).

(7.2) Dan left, but George did *not*.

(7.3) Bob has *wisely* walked to work, at least he has claimed he has. [56]

(7.4) Heute hat sich *anscheinend* Peter das letzte Stück Kuchen genommen, und gestern hat er das auch getan. [106]

(7.5) Every colleague paid attention, but every student did *not.*

(7.6) John went to the station, and every student did, too, *on a bike.*

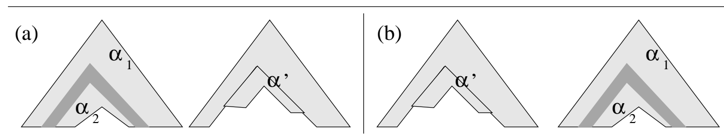


Figure 7.2: Cutting out a contrasting element in the middle: extra contrasting element (a) in the source sentence, (b) in the target sentence

Sentence (7.3) follows the schema in Fig. 7.2 (a): The target sentence means the same as "...at least he has claimed he has walked to work". The "wisely" of the source sentence does not reoccur in the target. So the deeper shaded segment within the left segment of Fig. 7.2 (a) represents the meaning of "wisely". Sentence (7.4) evinces the same phenomenon as sentence (7.3), but for German. Sentences (7.5) and (7.6) conform to the schema in Fig. 7.2 (b): In sentence (7.5) the "not" in the target sentence is a contrasting element that is parallel to nothing in the source sentence, and the same holds for "on a bike" in the target sentence of (7.6).

How can we describe these phenomena? In both pictures of Fig. 7.2, we have two segments on one side being structurally isomorphic to one bigger segment on the other side. We have decided on the generalization shown in Fig. 7.3:

- All cases of ellipsis that we have seen in previous chapters follow the schema depicted in Fig. 7.3 (a). The semantics of the source and the target sentence, the subtrees $\pi_0/$ and $\psi_0/$, have the same structure, except for the semantics of the contrasting elements, the subtrees $\pi_1/$ and $\psi_1/$, drawn as deeper shaded areas.

- The cases of ellipsis that we have seen in the current section can be generalized to the schema in Fig. 7.3 (b): The semantics of the source and target sentences, the subtrees $\pi_0/$ and $\psi_0/$, have the same structure, except for the semantics of the contrasting elements, the *segments* π_1/π_2 and ψ_1/ψ_2. (Again, the excepted segments are drawn as deeper shaded areas.)

  Note that there are mixed cases, with subtree-shaped as well as segment-shaped contrasting element semantics. (In fact, all the examples we have considered above are of this type.) Note further that in the examples that we have considered above, we have either a non-singleton excepted segment for the source sentence, and a singleton excepted segment for the target (sentences (7.3) and (7.4)), or vice versa (sentences (7.2), (7.5), (7.6)), where a singleton segment has the form π/π for some node π.

The sentences (7.5) and (7.6) are interesting for a further reason. We focus on (7.5), the simpler of the two. The target sentence means the same as "...but every student did not

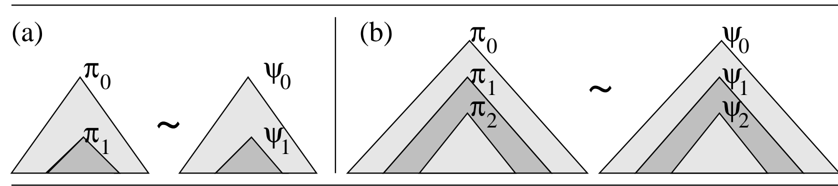


Figure 7.3: Modeling ellipsis: semantics of the contrasting element (a) a subtree, (b) a segment of the sentence semantics

pay attention"; it contains a scope ambiguity between "every student" and "not". In the reading with "not" taking wide scope, the sentence says that it is not the case that all students pay attention. In the reading with "every student" taking wide scope, it states that of all students it is true that their minds are wandering.

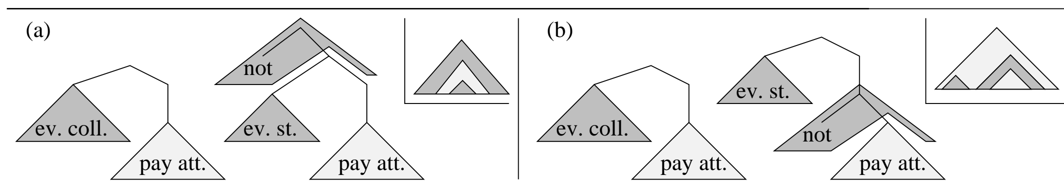


Figure 7.4: Sketch of the two readings of sentence (7.5): "Every colleague paid attention, but every student didn't."

These two readings are sketched in Fig. 7.4. The darker shaded segments are the semantics of the contrasting elements. Note the relative position of the two excepted segments, the one for "every student" and the one for "not": In Fig. 7.4 (a), the "not" segment dominates the "every student" segment, while in picture (b) the two excepted segments are in disjoint positions from each other in the lambda structure. So the target sentence semantics in (a) follows the schema sketched in the upper right-hand corner of that picture: We have a included segment in the middle with one excluded segment above and another below. For the target sentence semantics in (b), we have again sketched the schema in the upper right-hand corner: It comprises two included segments separated by one excluded segment, and another excluded segment in a disjoint position.

7.2 Modeling Ellipsis with Group Parallelism Constraints

How can we model the ellipsis phenomena that we have just sketched? Reconsider the schema in Fig. 7.3 (b). We have two parallel groups of segments, each consisting of two segments. We have seen something similar in the previous chapter, when we studied underspecified beta reduction. And in fact we can now reuse the *group parallelism relation*, which we introduced to model the result of an underspecified beta reduction step.

Recall that the group parallelism relation relates *groups*, tuples of segments, of a lambda

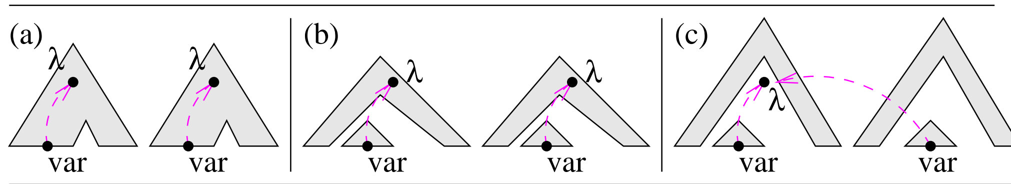


Figure 7.5: Possible bindings in a group parallelism.

structure. Group parallelism

$$(\alpha_1, \dots, \alpha_n) \sim (\beta_1, \dots, \beta_n)$$

holds if first, there exists a correspondence function for each segment pair α_i, β_i, $1 \leq i \leq n$, and second the group parallelism conditions on lambda and anaphoric binding are met. The conditions on lambda binding are sketched in Fig. 7.5: For a var-labeled node bound within the same segment, the corresponding node is bound within its own segment as shown in picture (a). For a var-labeled node of a segment α_i bound inside a different segment α_j of the same group, the correspondent is bound at the corresponding place within β_j, as shown in picture (b). This condition constitutes the main difference from normal parallelism. And for a var-labeled node bound outside its group, its correspondent is bound by the same binder, as shown in picture (c). Apart from that we have the familiar exclusion of hanging binders. The conditions on anaphoric binding are extended in the same way: An anaphoric binder between two segments of one group has to be paralleled by an anaphoric binder between the corresponding segments of the other group.

In the previous chapter we have defined the language $CLLS_{\text{gr}}$. It extends CLLS by inverse lambda binding literals and by group parallelism literals of the form

$$(A_1, \dots, A_m) \quad \sim \quad (B_1, \dots, B_m).$$

for segment terms $A_1, \dots A_m, B_1, \dots, B_m$. We can use this language to model the semantics of the ellipses in the previous chapter as follows.

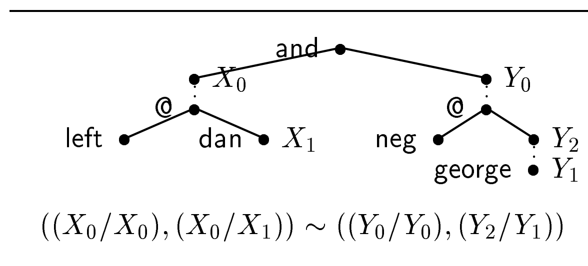
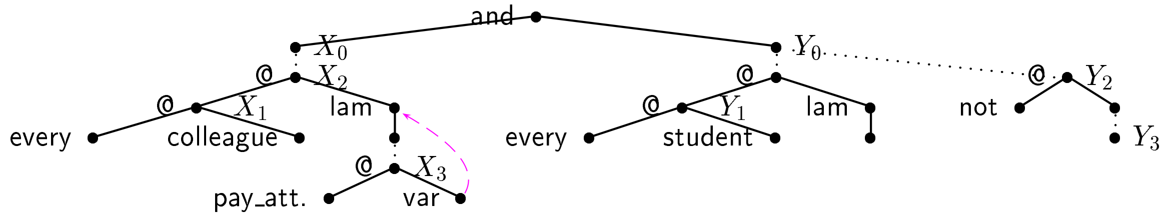


Figure 7.6: Constraint for sentence (7.2): “Dan left, but George didn’t.”

We first consider sentence (7.2), “Dan left, but George didn’t.” Figure 7.6 shows the constraint representing the semantics of this sentence. We want to state that the subtrees

$X_0/$ and $Y_0/$ are structurally isomorphic, except for the contributions of "Dan" and "George" and "not". Here we have done this by stating two pairs of not-excepted segment terms: The target segment term above the contribution of "not" and its counterpart in the source, and the target segment term below the "not" and its counterpart. The first pair of segment terms, X_0/X_0 and Y_0/Y_0, denote "singleton segments" consisting of just one node. We have chosen this representation because it generalizes to more complex cases where the segment terms above the excepted segments denote non-singleton segments.



$$((X_0/X_2),(X_2/X_1)) \sim ((Y_0/Y_2),(Y_3/Y_1)) \;\vee\; ((X_0/X_1,X_3),(X_3/)) \sim ((Y_0/Y_1,Y_2),(Y_3/))$$

Figure 7.7: Constraint for the sentence (7.5): "Every student paid attention, but every colleague didn't."

Now we turn to sentence (7.5), "Every colleague paid attention, but every student didn't." The constraint representing the semantics of this sentence is shown in Fig. 7.7. As sketched in Fig. 7.4, this sentence has two readings, and to model its meaning using the language CLLS_{gr}, we have to use a *disjunction* of group parallelism literals: The situation where "not" takes wide scope (Fig. 7.4 (a)) is described by the first disjunct in Fig. 7.7: Y_3 dominates Y_1. The situation where "every student" takes wide scope (Fig. 7.4 (b)) is described by the second disjunct: Y_1 and Y_2 lie in disjoint positions.

Note that in the reading sketched in Fig. 7.4 (b), a var-labeled node in the target "paid attention" fragment is bound by a binder in the "every student" fragment. This lambda binding obeys the condition sketched in Fig. 7.5 (b), i.e. it connects a var-labeled variable in one segment to a lam-labeled variable in another segment of the group. So to model the semantics of sentence (7.5), we truly need group parallelism literals; normal parallelism would not suffice because it does not allow for this kind of lambda binding.

However, this model is not quite satisfactory: We would like to model the meaning of sentences like this without resorting to disjunction. In the rest of this chapter, we show how this can be done.

7.3 Jigsaw Parallelism

In this section we introduce the *jigsaw parallelism relation*, which relates pairs of *jigsaw segments*, and we extend the language CLLS_{gr} by *jigsaw parallelism literals*, which are

interpreted by the jigsaw parallelism relation. In the next section we then show how jigsaw parallelism literals can be used to model the semantics of sentences like (7.5) without disjunction.

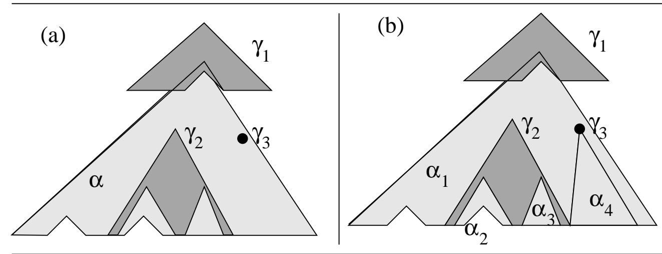


Figure 7.8: Sketch of (a) a jigsaw segment $\alpha/\gamma_1, \gamma_2, \gamma_3$ and (b) its remainder set $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

We start with a sketch of the idea of jigsaw segments and jigsaw parallelism, then we define the concepts formally.

A *jigsaw segment* has the form

$$\alpha/\gamma_1, \ldots, \gamma_n$$

for segments $\alpha, \gamma_1, \ldots, \gamma_n$ of a lambda structure. It can be read as "the segment α except for the segments $\gamma_1, \ldots, \gamma_n$". What does that mean? Consider Fig. 7.8. Picture (a) shows a sketch of a jigsaw segment $\alpha/\gamma_1, \gamma_2, \gamma_3$. The segment α has two holes, and segments $\gamma_1, \gamma_2, \gamma_3$ are being excluded from α. γ_1 overlaps only partially with α, and γ_3 is a singleton segment. Picture (b) shows what we get when we exclude $\gamma_1, \ldots, \gamma_3$ from α. It is a set of segments, the *remainder set* $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ of $\alpha/\gamma_1, \gamma_2, \gamma_3$: The segment α is cut along the segments $\gamma_1, \gamma_2, \gamma_3$ (hence the name jigsaw segment). We call the excluded segments (in our example $\gamma_1, \ldots, \gamma_3$) *gamma segments*, and the elements of the remainder set (in our example $\alpha_1, \ldots, \alpha_4$) *alpha segments* for short.

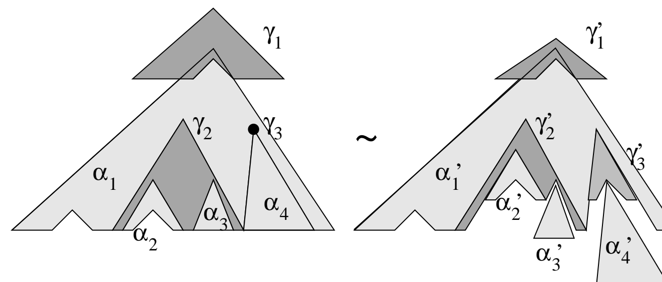


Figure 7.9: Sketch of jigsaw parallelism $\alpha/\gamma_1, \gamma_2, \gamma_3 \sim \alpha'/\gamma'_1, \gamma'_2, \gamma'_3$

Jigsaw parallelism relates pairs of jigsaw segments. We write

$$\alpha/\gamma_1, \ldots, \gamma_n \sim \alpha'/\gamma'_1, \ldots, \gamma'_n$$

to state that the jigsaw segments $\alpha/\gamma_1, \ldots, \gamma_n$ and $\alpha'/\gamma'_1, \ldots, \gamma'_n$ are parallel. The two jigsaw segments need to have an equal number of gamma segments as well as an equal number of alpha segments. Two jigsaw segments are parallel if *group parallelism* holds between their remainder sets. How do we determine which pairs of alpha segments should be parallel in the group parallelism? We use the relative positions of alpha and gamma segments. Consider Fig. 7.9, a sketch of a jigsaw parallelism $\alpha/\gamma_1, \gamma_2, \gamma_3 \sim \alpha'/\gamma'_1, \gamma'_2, \gamma'_3$. The order of gamma segments is important: It states that γ_1 is at the same position with respect to α that γ'_1 occupies with respect to α', and likewise for the pairs γ_2, γ'_2 and γ_3, γ'_3. This order then determines that segments α_1 and α'_1 have to be parallel because their roots coincide with the holes of matching gamma segments γ_1, γ'_1 and their holes coincide with the roots of matching gamma segments γ_2, γ'_2 and γ_3, γ'_3. In the same way, all segment pairs α_i, α'_i, $1 \leq i \leq 4$, are matched. So the group parallelism that needs to hold in this case is $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \sim (\alpha'_1, \alpha'_2, \alpha'_3, \alpha'_4)$. Note that matching gamma segments γ_i, γ'_i do not need to have the same structure; after all, they are *excluded* from the parallelism.

7.3.1 Jigsaw Segments

We now define *jigsaw segments* and *remainder sets*. We proceed in two steps: First we define unary jigsaw segments α/γ, then we generalize the definition to jigsaw segments with several gamma segments.

Two segments α, β of a lambda structure *overlap properly* iff either $\mathsf{b}^-(\alpha) \cap \mathsf{b}^-(\beta) \neq \emptyset$, or α is a singleton with $r(\alpha) \in \mathsf{i}(\beta)$. (Recall that by Def. 2.2, p. 27, the set of "interior nodes" $\mathsf{i}(\beta)$ is $\mathsf{b}^-(\beta) - \{r(\beta)\}$.) We call a segment α of a lambda structure a *singleton* iff $|\mathsf{b}(\alpha)| = 1$.

Definition 7.1 (Unary jigsaw segment, remainder set). *A* unary jigsaw segment *of a lambda structure $\mathcal{L}^\theta$ is a tuple α/γ of segments α, γ of $\mathcal{L}^\theta$. The* remainder set of *α/γ, $\mathsf{js}(\alpha/\gamma)$ is defined as follows.*

1. $\mathsf{js}(\alpha/\gamma) = \{\}$ *if* $\mathsf{b}(\alpha) \subseteq \mathsf{b}(\gamma)$.

2. $\mathsf{js}(\alpha/\gamma) = \{\alpha\}$ *if α and γ do not overlap properly.*

3. *For non-singleton α to which the first two cases do not apply, let*

$$\mathsf{roots}(\alpha/\gamma) = \big(\{r(\alpha)\} - \mathsf{b}^-(\gamma)\big) \cup \big(hs(\gamma) \cap \mathsf{i}(\alpha)\big)$$
$$\mathsf{holes}(\alpha/\gamma) = \big(hs(\alpha) - (\mathsf{i}(\gamma) \cup hs(\gamma))\big) \cup \big(\{r(\gamma)\} \cap \mathsf{i}(\alpha)\big)$$

and for $\pi \in \mathsf{roots}(\alpha/\gamma)$, *let*

$$\mathsf{holes{-}of}(\pi, \alpha/\gamma) = \{\psi \in \mathsf{holes}(\alpha/\gamma) \mid \pi \lhd^+ \psi \textit{ and } \nexists \pi' \in \mathsf{roots}(\alpha/\gamma) \textit{ such that } \big(\pi \lhd^+ \pi' \lhd^+ \psi\big)\}$$

Then

$$\mathsf{js}(\alpha/\gamma) = \{\pi_0/\pi_1, \ldots, \pi_n \mid \pi_0 \in \mathsf{roots}(\alpha/\gamma), \quad \pi_1, \ldots, \pi_n \textit{ are the members of } \mathsf{holes{-}of}(\pi_0, \alpha/\gamma) \textit{ ordered left to right}\}.$$

The segment α is broken into pieces by cutting out γ. What remains is a set of segments rooted by the members of $\mathsf{roots}(\alpha/\gamma)$. A remainder segment rooted by π has as its exceptions the nodes in $\mathsf{holes{-}of}(\pi, \alpha/\gamma)$. They are those members of $\mathsf{holes}(\alpha/\gamma)$ that are dominated by π such that no other member of $\mathsf{roots}(\alpha/\gamma)$ intervenes. Figure 7.10 illustrates the definition of $\mathsf{roots}(\alpha/\gamma)$ and $\mathsf{holes}(\alpha/\gamma)$: Picture (a) shows a case where γ is "in the middle of" α. Then the root of α and the hole of γ are in $\mathsf{roots}(\alpha/\gamma)$, and the root of γ and the hole of α are in $\mathsf{holes}(\alpha/\gamma)$. In picture (b) $r(\alpha)$ lies in $\mathsf{i}(\gamma)$. The root of α is in $\mathsf{roots}(\alpha/\gamma)$ only if it does not lie in $\mathsf{b}^-(\gamma)$, and the root of γ lies in $\mathsf{holes}(\alpha/\gamma)$ only if it is in $\mathsf{i}(\alpha)$. So $\mathsf{roots}(\alpha/\gamma)$ only contains the hole of γ, and $\mathsf{holes}(\alpha/\gamma)$ consists of the hole of α. In picture (c) the hole of γ is also the hole of α. This excludes the hole of γ from $\mathsf{roots}(\alpha/\gamma)$ and the hole of α from $\mathsf{holes}(\alpha/\gamma)$, which leaves us $\mathsf{roots}(\alpha/\gamma) = \{r(\alpha)\}$ and $\mathsf{holes}(\alpha/\gamma) = \{r(\gamma)\}$. In all three pictures, the remainder set, $\mathsf{js}(\alpha/\gamma)$, consists of the remaining light-shaded segments, conforming to our intuition about cutting α along γ.

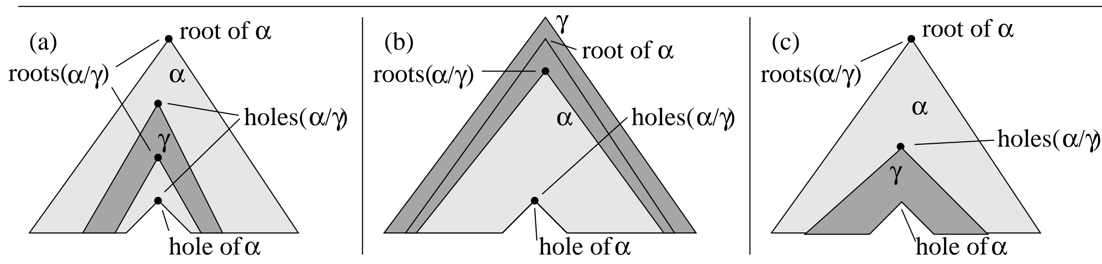


Figure 7.10: Illustrating Def. 7.1: roots and holes

This definition is rather complex. To show that it conforms to the above sketch of what a jigsaw segment is, we prove the following four properties: The remainder set only contains non-singleton segments, unless α is a singleton; no two members of the remainder set overlap properly; all are contained within α, and together with γ, they cover all of α.

Lemma 7.2 (Remainder set consists of segments). *If α/γ is a unary jigsaw segment, then all elements of $\mathsf{js}(\alpha/\gamma)$ are segments. If α is non-singleton, then so are all elements of $\mathsf{js}(\alpha/\gamma)$.*

Proof. By definition 7.1, the set $\mathsf{js}(\alpha/\gamma)$ consists of elements $\pi_0/\pi_1, \ldots, \pi_n$ such that π_0 strictly dominates all of $\pi_1, \ldots, \pi_n$. It remains to show that $\pi_1, \ldots, \pi_n$ are pairwise disjoint.

The nodes $\pi_1, \ldots, \pi_n$ are from the set $hs(\alpha) \cup \{r(\gamma)\}$. The nodes in $hs(\alpha)$ are pairwise disjoint by the definition of segments. It remains to consider $r(\gamma)$. Suppose $r(\gamma) \in \mathsf{holes}(\alpha/\gamma)$ and suppose there is a node $\pi \in hs(\alpha)$ with $r(\gamma)\lhd^*\pi$. If $r(\gamma){=}\pi$ we are done. So suppose $r(\gamma)\lhd^+\pi$. There are two possibilities: Either π is in $\mathsf{i}(\gamma) \cup hs(\gamma)$, then $\pi \notin \mathsf{holes}(\alpha/\gamma)$. Or there is a hole ψ of γ with $\psi\lhd^+\pi$, and we have $\pi \in \mathsf{holes}(\alpha/\gamma)$. In this case all of γ lies in the interior of α: On the one hand $r(\gamma) \in \mathsf{i}(\alpha)$ since $r(\gamma) \in \mathsf{holes}(\alpha/\gamma)$ and on the other hand $\psi\lhd^+\pi$ and π is a hole of α. So we also have $\psi \in \mathsf{roots}(\alpha/\gamma)$. But in that case, $r(\gamma)$ lies in $\mathsf{holes{-}of}(r(\alpha), \alpha/\gamma)$, and π does not lie in $\mathsf{holes{-}of}(r(\alpha), \alpha/\gamma)$

but rather in $\mathsf{holes{-}of}(\psi, \alpha/\gamma)$, i.e. π and $r(\gamma)$ are not holes of the same element of the remainder set $\mathsf{js}(\alpha/\gamma)$. □

Lemma 7.3 (Remainder segments are non-overlapping). *Let α/γ be a unary jigsaw segment of some lambda structure, then its remainder set $\mathsf{js}(\alpha/\gamma)$ is a set of segments that do not overlap properly.*

Proof. Only the third case of Def. 7.1 is of interest here. If the root of each segment in $\mathsf{js}(\alpha/\gamma)$ is in $hs(\gamma) \cap \mathsf{i}(\alpha)$, then the segments lie in disjoint positions. Now suppose $\mathsf{js}(\alpha/\gamma)$ contains a segment α_1 with $r(\alpha_1) = r(\alpha) \notin \mathsf{b}^-(\gamma)$. If $\mathsf{js}(\alpha/\gamma)$ contains another segment α_2 besides α_1, then we must have $r(\alpha_2) \in hs(\gamma) \cap \mathsf{i}(\alpha)$. Suppose α_1 and α_2 properly overlap, then $r(\alpha_1) \triangleleft^+ r(\alpha_2)$. As $r(\alpha) \notin \mathsf{b}^-(\gamma)$ but $r(\alpha)$ dominates a hole of γ, we must have $r(\alpha_1) \triangleleft^+ r(\gamma)$. So we get $r(\gamma) \in \mathsf{i}(\alpha)$ since $r(\alpha_2) \in \mathsf{i}(\alpha)$. Thus, $r(\gamma) \in \mathsf{holes}(\alpha/\gamma)$ and also $r(\gamma) \in \mathsf{holes{-}of}(r(\alpha_1), \alpha/\gamma)$ since it cannot be dominated by any other element of $\mathsf{roots}(\alpha/\gamma)$. Which means that α_1 and α_2 do not properly overlap, after all. □

This is even true of $\{\gamma\} \cup \mathsf{js}(\alpha/\gamma)$: The only interesting case is the one where $\mathsf{js}(\alpha/\gamma)$ contains a segment α_1 with $r(\alpha_1) = r(\alpha) \notin \mathsf{b}^-(\gamma)$. Suppose α_1 and γ overlap properly, then $r(\alpha_1) \triangleleft^* r(\gamma)$. If $r(\gamma) \notin \mathsf{holes}(\alpha/\gamma)$, then there must be some $\pi \in hs(\alpha) \cap \mathsf{holes{-}of}(r(\alpha_1), \alpha/\gamma)$ dominating it. If $r(\gamma) \in \mathsf{holes}(\alpha/\gamma)$, then it is in $\mathsf{holes{-}of}(r(\alpha_1), \alpha/\gamma)$ since $r(\alpha_1) \notin \mathsf{b}^-(\gamma)$.

Lemma 7.4 (Remainder segments are contained in α). *If α/γ is a unary jigsaw segment of some lambda structure, with $\mathsf{js}(\alpha/\gamma) = \{\alpha_1, \ldots, \alpha_n\}$, then $\bigcup_{i=1}^n \mathsf{b}(\alpha_i) \subseteq \mathsf{b}(\alpha)$.*

Proof. Again, we need only consider the third case of Def. 7.1. By the definition of $\mathsf{roots}(\alpha/\gamma)$, $\mathsf{js}(\alpha/\gamma)$ contains no segment with a root that strictly dominates $r(\alpha)$. It remains to check that no segment of $\mathsf{js}(\alpha/\gamma)$ extends below a hole of α.

Let $\pi \in hs(\alpha)$ with $\pi \notin \mathsf{holes}(\alpha/\gamma)$. Then $\pi \in \mathsf{i}(\gamma)$, so $r(\gamma) \triangleleft^+ \pi$. Let $\psi \in \mathsf{roots}(\alpha/\gamma)$ with $\psi \triangleleft^* \pi$. Then $\psi \notin hs(\gamma)$ by the definition of the "interior" function i. If $\psi = r(\alpha)$ then $\psi \notin \mathsf{b}^-(\gamma)$ so $\psi \triangleleft^+ r(\gamma)$ and $r(\gamma) \in \mathsf{i}(\alpha)$. So $r(\gamma) \in \mathsf{holes{-}of}(\psi, \alpha/\gamma)$, and the segment beginning at ψ ends above π already.

Now suppose $\pi \in hs(\alpha)$ with $\pi \in \mathsf{holes}(\alpha/\gamma)$. If there is some $\psi \in hs(\gamma) \cap \mathsf{roots}(\alpha/\gamma)$ with $r(\alpha) \triangleleft^+ \psi \triangleleft^* \pi$, then $\pi \in \mathsf{holes{-}of}(\psi, \alpha/\gamma)$ since $\psi \in \mathsf{i}(\alpha)$. Otherwise, $\pi \in \mathsf{holes{-}of}(r(\alpha), \alpha/\gamma)$: we have $r(\alpha) \triangleleft^+ \pi$ since α is non-singleton. □

Lemma 7.5 (Partitioning α with γ and the remainder set). *If α/γ is a unary jigsaw segment of some lambda structure with $\mathsf{js}(\alpha/\gamma) = \{\alpha_1, \ldots, \alpha_n\}$, then $\mathsf{b}(\alpha) \subseteq \mathsf{b}(\gamma) \cup \bigcup_{i=1}^n \mathsf{b}(\alpha_i)$.*

Proof. As above, we need only consider the third case of Def. 7.1. Suppose $\pi \in \mathsf{b}(\alpha) - \bigcup_{i=1}^n \mathsf{b}(\alpha_i)$ and $\pi \notin \mathsf{b}(\gamma)$. Then $r(\alpha) \triangleleft^* \pi$. There are two cases.

Either $r(\alpha) \in \mathsf{b}^-(\gamma)$. Then there must be some $\psi \in hs(\gamma)$ such that $\psi \triangleleft^+ \pi$. Then $\psi \in \mathsf{i}(\alpha)$, so there exists some $j \in \{1, \ldots, n\}$ with $\psi = r(\alpha_j)$. As $\pi \notin \mathsf{b}(\alpha_j)$, there must be some $w \in hs(\alpha_j)$ with $w \triangleleft^+ \pi$. But then by the definition of $\mathsf{holes}(\alpha/\gamma)$, we must have $w \in hs(\alpha)$, hence $\pi \notin \mathsf{b}(\alpha)$, a contradiction.

The other case is $r(\alpha) \notin \mathsf{b}^-(\gamma)$. Then there exists some segment $\alpha_1 \in \mathsf{js}(\alpha/\gamma)$ with $r(\alpha_1) = r(\alpha)$. We have $\pi \notin \mathsf{b}(\alpha_1)$ so there must be some $\psi \in hs(\alpha_1)$ with $\psi \triangleleft^+ \pi$. Since $\pi \in \mathsf{b}(\alpha)$, it must hold that $\psi \notin hs(\alpha)$, so $\psi = r(\gamma) \in \mathsf{i}(\alpha)$. Now $\pi \notin \mathsf{b}(\gamma)$, and we can proceed as in the previous case and get a contradiction the same way. □

We extend the definition from unary to general jigsaw segments, where we can except several segments $\gamma_1, \ldots, \gamma_n$ from one segment α. To that end, we first generalize our notion of a remainder set as follows:

Definition 7.6 (Remainder set). *Given segments $\alpha_1, \ldots, \alpha_n, \gamma$ of a lambda structure such that for all $1 \leq i < j \leq n$, α_i and α_j do not properly overlap; then the* remainder set *of $\{\alpha_1, \ldots, \alpha_n\}$ and γ is*

$$\mathsf{js}(\{\alpha_1, \ldots, \alpha_n\} \,/\, \gamma) =_{\text{def}} \bigcup_{i=1}^{n} \mathsf{js}(\alpha_i/\gamma).$$

Now we can define the concept of a *jigsaw segment.*

Definition 7.7 (Jigsaw segment, remainder set). *A* jigsaw segment *ω of a lambda structure $\mathcal{L}^\theta$ is a tuple*

$$\omega = \alpha \,/\, \gamma_1, \ldots, \gamma_n$$

of segments $\alpha, \gamma_1, \ldots, \gamma_n$ of $\mathcal{L}^\theta$.

The remainder set *of ω is*

$$\mathsf{js}(\omega) =_{\text{def}} \mathsf{js}(\ldots \mathsf{js}(\mathsf{js}(\alpha/\gamma_1) \,/\, \gamma_2) \ldots \,/\, \gamma_n).$$

$\mathsf{js}(\omega)$ is a set of segments. By Lemma 7.3, $\mathsf{js}(\ldots \mathsf{js}(\mathsf{js}(\alpha/\gamma_1) \,/\, \gamma_2) \ldots \,/\, \gamma_i)$ is a set of non-overlapping segments for $1 \leq i \leq n$, so $\mathsf{js}(\omega)$ is well-defined. We use

$$\mathsf{b}(\omega) =_{\text{def}} \bigcup_{\alpha' \in \mathsf{js}(\omega)} \mathsf{b}(\alpha').$$

In the following two lemmas we show that the definition of jigsaw segments conforms to the sketch we have given earlier: The above observations on coverage and partitioning hold for general jigsaw segments as well, and the order in which gamma segments are excluded does not matter.

Lemma 7.8 (Partitioning α with the alpha and gamma sets).

Let $\omega = \alpha \,/\, \gamma_1, \ldots, \gamma_n$. Lemmas 7.4 and 7.5 scale up:

1. $\mathsf{b}(\omega) \subseteq \mathsf{b}(\alpha)$.

2. $\mathsf{b}(\alpha) \subseteq \bigcup_{i=1}^{n} \mathsf{b}(\gamma_i) \cup \mathsf{b}(\omega)$.

Proof. We proceed by induction on n.

1. Suppose the first claim is true for $\omega_\ell = \alpha \,/\, \gamma_1, \ldots, \gamma_\ell$ for some ℓ, $1 \leq \ell < n$. Let $\mathsf{js}(\omega_\ell) = \{\alpha'_1, \ldots, \alpha'_k\}$. Then for each $1 \leq i \leq k$, $\mathsf{js}(\alpha'_i/\gamma_{\ell+1}) \subseteq \mathsf{b}(\alpha'_i)$ by Lemma 7.4. Hence, $\mathsf{js}(\alpha \,/\, \gamma_1, \ldots, \gamma_{\ell+1}) \subseteq \mathsf{js}(\alpha \,/\, \gamma_1, \ldots, \gamma_\ell) \subseteq \mathsf{b}(\alpha)$.

2. Suppose the second claim is true for $\omega_\ell = \alpha \,/\, \gamma_1, \ldots, \gamma_\ell$ for some ℓ, $1 \leq \ell < n$. Let $\mathsf{js}(\omega_\ell) = \{\alpha'_1, \ldots, \alpha'_k\}$. Then for each $1 \leq i \leq k$, $\mathsf{b}(\alpha'_i) \subseteq \mathsf{b}(\gamma_{\ell+1}) \cup \mathsf{js}(\alpha'_i/\gamma_1, \ldots, \gamma_{\ell+1})$ by Lemma 7.5. Hence, $\mathsf{b}(\alpha) \subseteq \bigcup_{i=1}^{\ell+1} \mathsf{b}(\gamma_i) \cup \mathsf{js}(\alpha \,/\, \gamma_1, \ldots, \gamma_{\ell+1})$.

□

Lemma 7.9 (Order-independence of gamma segments). *Let $\alpha_1, \ldots, \alpha_n, \gamma_1, \gamma_2$ be segments of the same lambda structure such that for all $1 \leq i < j \leq n$, α_i and α_j do not overlap properly. Then*

$$\mathsf{js}(\mathsf{js}(\{\alpha_1, \ldots, \alpha_n\} \,/\, \gamma_1) \,/\, \gamma_2) = \mathsf{js}(\mathsf{js}(\{\alpha_1, \ldots, \alpha_n\} \,/\, \gamma_2) \,/\, \gamma_1)$$

Proof. We write $\mathcal{S}_i = \mathsf{js}(\{\alpha_1, \ldots, \alpha_n\} \,/\, \gamma_i)$, $i = 1, 2$, for short. Let $\alpha' \in \mathsf{js}(\mathcal{S}_1/\gamma_2)$. We have to show that $\alpha' \in \mathsf{js}(\mathcal{S}_2/\gamma_1)$ holds as well. As α' is in $\mathsf{js}(\mathcal{S}_1/\gamma_2)$, there must be some $\alpha'' \in \mathcal{S}_1$ with $\alpha' \in \mathsf{js}(\alpha''/\gamma_2)$ and some k, $1 \leq k \leq n$, with $\alpha'' \in \mathsf{js}(\alpha_k/\gamma_1)$. We reason over the possible positions of α' and α''.

Suppose $\alpha'' = \alpha_k$. Then α_k and γ_1 do not overlap properly, and neither do α' and γ_1. So $\alpha' \in \mathsf{js}(\alpha_k/\gamma_2)$ and also $\alpha' \in \mathsf{js}(\mathsf{js}(\alpha_k/\gamma_2) \,/\, \gamma_1)$.

Now suppose otherwise. W.l.o.g. we consider the case that $r(\alpha'') = r(\alpha_k)$ but $r(\gamma_1) \in hs(\alpha'')$. (The case where $r(\alpha'') \in hs(\gamma_1)$ and $hs(\alpha'') \subseteq hs(\alpha_k)$ is analogous.)

If $r(\gamma_1) \notin \mathsf{b}(\alpha')$, then $\alpha' \in \mathcal{S}_2$ already, and α' and γ_1 do not overlap properly, so $\alpha' \in \mathsf{js}(\mathcal{S}_2/\gamma_1)$. Now suppose $r(\gamma_1) \in \mathsf{b}(\alpha')$. If additionally $\mathsf{b}(\gamma_2) \cap \mathsf{b}(\alpha') = \emptyset$, then $\alpha' = \alpha''$ and there are two possibilities: either γ_2 does not properly overlap α_k, i.e. $\alpha_k \in \mathcal{S}_2$, so $\alpha' \in \mathsf{js}(\mathcal{S}_2/\gamma_1)$; or $r(\gamma_1) \lhd^* r(\gamma_2)$ and there exists a segment $\alpha''' \in \mathcal{S}_2$ with $r(\gamma_2) \in hs(\alpha''')$ and $\alpha' \in \mathsf{js}(\alpha'''/\gamma_1)$.

Now suppose $r(\gamma_1) \in \mathsf{b}(\alpha')$ as well as $\mathsf{b}(\gamma_2) \cap \mathsf{b}(\alpha') \neq \emptyset$. Then there are two possibilities: either $r(\gamma_1) = r(\gamma_2)$ and $\alpha'' = \alpha' \in \mathcal{S}_2$ as well as $\alpha' \in \mathsf{js}(\mathcal{S}_2/\gamma_1)$; or γ_1, γ_2 do not overlap properly, that is, they except pieces of α_k that do not overlap properly either, so the order in which the two exclusions take place does not matter. □

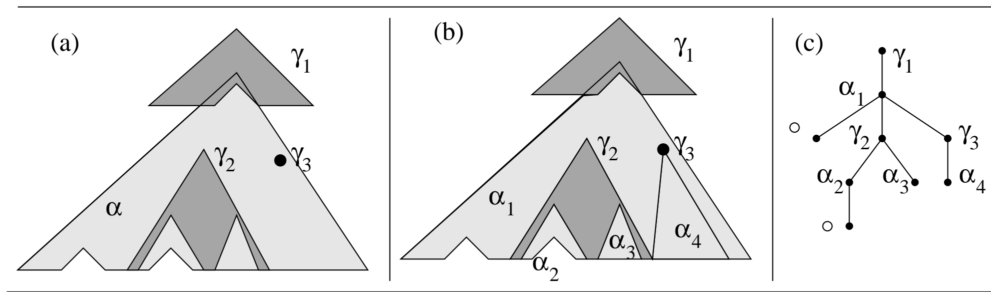


Figure 7.11: Jigsaw segments and alpha-gamma trees.

7.3.2 The Jigsaw Parallelism Relation

Now we define the *jigsaw parallelism relation*, which relates two jigsaw segments $(\alpha \,/\, \gamma_1, \ldots, \gamma_n) \sim (\alpha' \,/\, \gamma'_1, \ldots, \gamma'_n)$. Two jigsaw segments are parallel if *group parallelism* holds between their remainder sets, where we use the relative positions of alpha and gamma segments to determine which pairs of alpha segments should be parallel. For example, we have argued that in Fig. 7.9 the segments α_1 and α'_1 have to be parallel because their roots coincide with the holes of matching gamma segments γ_1, γ'_1 and their holes coincide with the roots of matching gamma segments γ_2, γ'_2 and γ_3, γ'_3. We formalize the relative positions of alpha and gamma segments in the notion of an *alpha-gamma tree*.

Consider Fig. 7.11. Picture (a) again shows the jigsaw segment $\alpha/\gamma_1, \gamma_2, \gamma_3$ that we have discussed above, and picture (b) again shows how the remainder set $\{\alpha_1, \ldots, \alpha_4\}$ of this jigsaw segment is obtained by cutting α along the gamma segments. Note that adjacent segments generally share a node, which is the root of one and a hole of the other segment.

The way that the alpha and gamma segments are plugged into each other is represented in the *alpha-gamma tree* shown in picture (c). An alpha-gamma tree is a tree which contains exactly one node with label α_i for each i, at most one node with label γ_i for each i, and nodes with label $\bullet$ for holes of alpha or gamma segments that are not roots of another segment (in Fig. 7.11 represented as $\circ$). The children of a node labelled α_i are labelled by the segments plugged into the holes of α_i in the correct left-to-right order; the same holds for a node labelled γ_i.

Definition 7.10 (Alpha-gamma tree). *An* alpha-gamma tree *for a jigsaw segment $\omega = \alpha \,/\, \gamma_1, \ldots, \gamma_n$ of a lambda structure is a tree θ such that the following conditions are fulfilled, with $\mathcal{S} = \mathsf{js}(\omega) \cup \{\gamma_i \mid i \in \{1, \ldots, n\}, \gamma_i$ and α overlap properly$\}$:*

1. *the nodes in θ all bear labels from the set $\mathcal{S} \cup \{\bullet\}$;*
2. *for all $\beta \in \mathcal{S}$, there is exactly one node labeled β in θ;*
3. *for all $\beta \in \mathcal{S}$, the node labeled β has exactly $|hs(\beta)|$ children;*

4. *if a node labeled β has as its i-th child a node labeled β', then the i-th hole of the segment β (in left-to-right order) is $r(\beta')$; if a node labeled β has as its i-child a node labeled $\bullet$, then the i-th hole of β is not in $\mathsf{b}^-(\alpha)$.*

Below, we will use alpha-gamma trees to define jigsaw parallelism: Two parallel jigsaw segments need to have matching alpha-gamma trees, and the way that they match will determine which pairs of alpha segments have to be parallel. But before we define the jigsaw parallelism relation, we need to show that alpha-gamma trees are defined in a useful way. In the following two lemmas, we show that if the gamma segments do not overlap properly, the jigsaw segment possesses alpha-gamma trees; and if they exist, alpha-gamma trees are unique up to permutations of equal singleton gamma segments.

Lemma 7.11 (Existence of alpha-gamma trees). *Let $\omega = \alpha \,/\, \gamma_1, \ldots, \gamma_n$ be a jigsaw segment of a lambda structure such that for all $1 \leq i < j \leq n$, γ_i and γ_j do not overlap properly. Then ω possesses an alpha-gamma tree.*

Proof. We proceed by induction on n.

$n = 1$: Then $\omega = \alpha/\gamma$. If $\mathsf{js}(\omega) = \{\}$ then $\theta = \gamma(\bullet, \ldots, \bullet)$ is the only alpha-gamma tree for ω. If $\mathsf{js}(\omega) = \{\alpha\}$ then $\theta = \alpha(\bullet, \ldots, \bullet)$ is the only alpha-gamma tree for ω.

Now suppose α is not a singleton, and the two first cases of Def. 7.1 do not apply. Then there exists a single alpha-gamma tree θ for ω, which is constructed as follows: let the holes of α, ordered left to right in the tree, be $\pi_1, \ldots, \pi_m$, and the holes of γ, similarly ordered, $\psi_1, \ldots, \psi_\ell$. Suppose there exists some $\alpha_1 \in \mathsf{js}(\alpha/\gamma)$ with $r(\alpha_1) = r(\alpha)$. Then there exist $1 \leq i < j \leq m$ such that $hs(\alpha_1)$, ordered left to right, is $\pi_1, \ldots, \pi_i, r(\gamma), \pi_j, \ldots, \pi_m$. Then θ has the form

$$\alpha_1(\underbrace{\bullet, \ldots, \bullet,}_{i \text{ times}} \gamma(\theta_1, \ldots, \theta_\ell), \quad \underbrace{\bullet, \ldots, \bullet}_{(m-j+1) \text{ times}} \quad)$$

for trees $\theta_1, \ldots, \theta_\ell$ that we explain below. If, on the other hand, there exists no such α_1, then θ has the form $\gamma(\theta_1, \ldots, \theta_\ell)$, again for trees $\theta_1, \ldots, \theta_\ell$ that we explain next.

For $1 \leq i \leq \ell$, if $\psi_i = r(\alpha')$ for some $\alpha' \in \mathsf{js}(\omega)$, then $\theta_i = \alpha'(\underbrace{\bullet, \ldots, \bullet}_{|hs(\alpha')| \text{ times}})$.

Otherwise, $\theta_i = \bullet$.

$(n-1) \to n$: Let θ' be an alpha-gamma tree for $\omega' = \alpha \,/\, (\gamma_1, \ldots, \gamma_{n-1})$. Such a tree exists by the inductive hypothesis. There are three possibilities: (1) γ_n and α do not overlap properly; or (2) γ_n and α overlap properly, but there exists no $\alpha' \in \mathsf{js}(\omega')$ such that γ_n and α' overlap properly; or (3) there exists exactly one segment $\alpha' \in \mathsf{js}(\omega')$ such that γ_n and α' overlap properly. No further cases exist: any two segments in $\mathsf{js}(\omega')$ must be separated by some γ_i, $1 \leq i \leq n$, otherwise they

would not be separate segments, but γ_n does not properly overlap with any other γ_i.

In case (1), θ' is also an alpha-gamma tree for ω. Case (2) implies that γ_n must be a singleton segment, and that there exists some j, $1 \leq j \leq n-1$, such that either (2a) $r(\gamma_n) = r(\gamma_j)$ or (2b) $r(\gamma_n) \in hs(\gamma_j)$. (There may be more than one such j.) In case (2a), θ' contains a subtree $\gamma_j(\theta_j)$ for some θ_j. Replacing this subtree by $\gamma_n(\gamma_j(\theta_j))$, we obtain an alpha-gamma tree for ω. In case (2b), suppose $r(\gamma_n)$ is the i-th hole of γ_j. θ' has a subtree $\gamma_j(\ldots,\theta_i,\ldots)$, where the root of θ_i is the i-th child of the node labeled γ_j. Replacing θ_i by $\gamma_n(\theta_i)$, we obtain an alpha-gamma tree for ω.

We now consider case (3). Let θ_{new} be the only alpha-gamma tree for α'/γ_n constructed as shown above. θ' contains a subtree $\alpha'(\theta_1,\ldots,\theta_m)$ for some m and some trees $\theta_1,\ldots,\theta_m$. For $1 \leq i \leq m$, let β_i be the label of the root of θ_i. Now for each $\beta \in \mathsf{js}(\alpha'/\gamma_n) \cup \{\gamma_n\}$, let π be the node in θ_{new} labeled β; if the j-th hole of β is equal to $r(\beta_i)$, then exchange the j-th child of π by θ_i. (In that case, the j-th child of π must be labeled $\bullet$ in θ_{new}.) Let θ'_{new} be the tree that results from all these substitutions. (Note that if for some $i \in \{1,\ldots,m\}$, θ_i did not get picked, then $\theta_i = \bullet$ because γ_n does not overlap any segments of $\mathsf{js}(\omega')$ except α'.) Then the tree θ obtained from θ' by replacing the subtree $\alpha'(\theta_1,\ldots,\theta_m)$ by θ'_{new} is an alpha-gamma tree for ω.

□

Lemma 7.12 (Uniqueness of alpha-gamma trees). *Let $\omega = \alpha \,/\, \gamma_1,\ldots,\gamma_n$ be a jigsaw segment of some lambda structure such that for all $1 \leq i < j \leq n$, γ_i and γ_j do not overlap properly, and ω admits two different alpha-gamma trees θ_1, θ_2. Then θ_2 can be obtained from θ_1 by permuting the singleton γ_i labels.*

Proof. This follows from the proof of the previous lemma: The only case in the construction of an alpha-gamma tree where we had any choice was case (2), the choice of γ_j for the case where γ_n was singleton. □

Now we define the jigsaw parallelism relation: Two jigsaw segments are parallel if their alpha-gamma trees can be matched by a tree isomorphism, such that gamma segments with the same index are matching nodes in the alpha-gamma trees, and alpha segments that are matching nodes in the alpha-gamma trees are structurally isomorphic.

Definition 7.13 (Jigsaw parallelism relation). *The* jigsaw parallelism relation $\sim$ *of a lambda structure $\mathcal{L}^\theta$ is the largest relation between jigsaw segments with equal numbers of gamma segments such that*

$$(\alpha \,/\, \gamma_1,\ldots,\gamma_n) \sim (\alpha' \,/\, \gamma'_1,\ldots,\gamma'_n)$$

for jigsaw segments $\omega = \alpha \,/\, \gamma_1,\ldots,\gamma_n$, $\omega' = \alpha' \,/\, \gamma'_1,\ldots,\gamma'_n$ implies

- *there exists a bijection* $f : \mathsf{js}(\omega) \to \mathsf{js}(\omega')$ *such that, for* $\mathsf{js}(\omega) = \{\alpha_1, \ldots, \alpha_m\}$,

  $$(\alpha_1, \ldots, \alpha_m) \sim (f(\alpha_1), \ldots, f(\alpha_m))$$

  holds in $\mathcal{L}^\theta$.
- *there are alpha-gamma trees* θ, θ' *for* ω, ω' *and a tree isomorphism* $h : D_\theta \to D_{\theta'}$ *that satisfies the following condition:*

  A node π *of* θ *is labeled* γ_j *iff* $h(\pi)$ *is labeled* γ'_j, $1 \leq j \leq n$; π *is labeled* α_j *iff* $h(\pi)$ *is labeled* $f(\alpha_j)$, $1 \leq j \leq m$; *and* π *is labeled* $\bullet$ *iff* $h(\pi)$ *is.*

Note that the jigsaw parallelism relation is defined only for jigsaw segments in which the gamma segments do not overlap properly.

Figure 7.12 illustrates the isomorphism h on alpha-gamma trees. The two trees in the figure have the same shape. Whenever we have a node labeled γ_j in the left tree, the matching node in the right tree is labeled γ'_j: The order of gamma segments in the jigsaw segments is obeyed, in that matching gamma segments are in the same positions in the alpha-gamma tree. And parallel alpha segments are characterized by the fact that their roots are the holes of matching gamma segments, and their holes are the roots of matching gamma segments. We could say that the condition that we are imposing is actually an extended notion of *correspondence*, this time correspondence between two alpha-gamma trees.

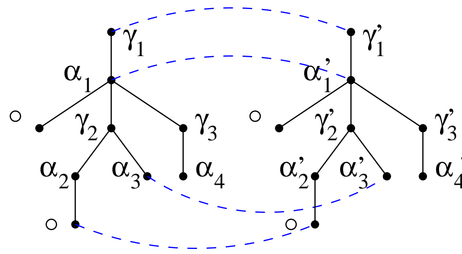


Figure 7.12: Matching two alpha-gamma trees by the tree isomorphism h of Def. 7.13: Some h-mappings are drawn in.

The jigsaw parallelism relation gives us no additional expressive power:

Lemma 7.14 (Jigsaw parallelism subsumed by group parallelism). *Given a lambda structure* $\mathcal{L}^\theta$ *in which* $\omega \sim \omega'$ *holds for jigsaw segments* ω, ω' *of* $\mathcal{L}^\theta$. *Then the group parallelism* $\varpi(\mathsf{js}(\omega)) \sim \varpi'(\mathsf{js}(\omega'))$ *holds in* $\mathcal{L}^\theta$ *for some permutations* ϖ, ϖ' *of the remainder sets* $\mathsf{js}(\omega), \mathsf{js}(\omega')$.

Proof. This follows from Def. 7.13. □

There are group parallelism relationships that cannot be expressed using jigsaw parallelism, since group parallelism allows segments of the same group to overlap, while the alpha segments of a jigsaw segment do not overlap properly (Lemma 7.8, p. 185).

7.3.3 Jigsaw Parallelism Literals

A *jigsaw segment term* has the form

$$A_0 \,/\, A_1, \ldots, A_n$$

for segment terms $A_0, \ldots, A_n$. We extend CLLS by *jigsaw parallelism literals* that are interpreted by the jigsaw parallelism relation. A *jigsaw parallelism literal* has the form

$$A_0 \,/\, A_1, \ldots, A_m \sim B_0 \,/\, B_1, \ldots, B_m$$

for segment terms $A_0, \ldots, A_m, B_0, \ldots, B_m$. We write CLLS_j for the language $\mathrm{CLLS_{gr}}$ extended by jigsaw parallelism literals.

Whenever two jigsaw segments are parallel, their remainder sets are related by group parallelism, as Lemma 7.14 shows. However a single jigsaw parallelism literal can express a disjunction of group parallelism literals. We state this in the following lemma.

The gamma segments of a jigsaw segment term may not overlap properly. We can use the following formula: Let $A = X_0/\ldots$, then

$$\mathit{nonovl}(A, \{A_1, \ldots, A_m\}) =_{\mathrm{def}} \bigwedge_{i=1}^{m} X_0 \notin \mathsf{b}^-(A_i).$$

Lemma 7.15 (Jigsaw p. literals express disjunctions of group p. literals). *Given a jigsaw parallelism literal $C_0/C_1, \ldots, C_m \sim C'_0/C'_1, \ldots, C'_m$, there exist group parallelism literals $\overline{A}_1 \sim \overline{B}_1,\ \ldots,\ \overline{A}_n \sim \overline{B}_n$ such that*

$$\begin{aligned} C_0 \,/\, C_1, \ldots, C_m \sim C'_0 \,/\, C'_1, \ldots, C'_m \;\models\!\!\!\mid\; & \textstyle\bigwedge_{i=1}^{m} \mathit{nonovl}(C_i, \{C_1, \ldots, C_m\} - \{C_i\}) \wedge \\ & \textstyle\bigwedge_{i=1}^{m} \mathit{nonovl}(C'_i, \{C'_1, \ldots, C'_m\} - \{C'_i\}) \wedge \\ & (\textstyle\bigvee_{i=1}^{n} \overline{A}_i \sim \overline{B}_i). \end{aligned}$$

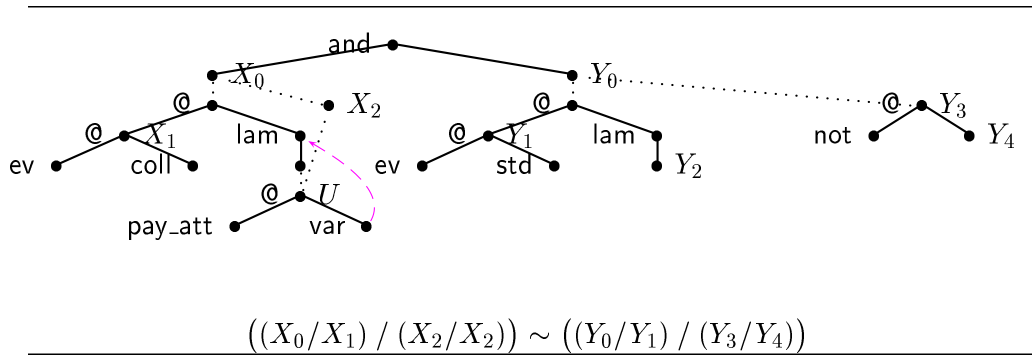
Proof. We abbreviate the jigsaw parallelism literal $C_0 \,/\, C_1, \ldots, C_m \sim C'_0 \,/\, C'_1, \ldots, C'_m$ by φ. Given a set V of variables, the formula

$$\mathit{disamb}(V) =_{\mathrm{def}} \bigvee_{X,Y \in V} X{=}Y \vee X{\lhd^+}Y \vee Y{\lhd^+}X \vee X{\perp}Y$$

disambiguates the relative positions of all variables in V without guessing labels. Let ς be a constraint in the clause set $\varphi \wedge \mathit{disamb}(\mathcal{V}ar(\varphi))$. In ς the relative positions of the root and hole variables of the segment terms occurring in the jigsaw parallelism literal are disambiguated. So we can put up alpha-gamma trees for the two jigsaw segment terms $C_0 \,/\, C_1, \ldots, C_m$ and $C'_0 \,/\, C'_1, \ldots, C'_m$ in ς in the way described in the proof of Lemma 7.11. Note that while that proof uses the fact that the left-to-right order of holes of a segment is known, this is not really necessary; it suffices to impose an arbitrary order on the holes of each of $C_0, \ldots, C_m$ and to impose the same order on the holes of C'_i as on C_i for each $0 \le i \le m$.

If there is an isomorphism between the two alpha-gamma trees that fulfills the conditions of Def. 7.13, we can read off one of the group parallelism literals from the set $\{\overline{A}_1 \sim \overline{B}_1, \ldots, \overline{A}_n \sim \overline{B}_n\}$. As the formula $disamb(\mathcal{V}ar(\varphi))$ exhaustively enumerates all possible relative positions of variables in $\mathcal{V}ar(\varphi)$, we find all of $\overline{A}_1 \sim \overline{B}_1, \ldots, \overline{A}_n \sim \overline{B}_n$ in this way. □

7.4 Modeling Ellipsis with Jigsaw Parallelism Constraints



$$\big((X_0/X_1)\ /\ (X_2/X_2)\big) \sim \big((Y_0/Y_1)\ /\ (Y_3/Y_4)\big)$$

Figure 7.13: Constraint for sentence (7.5): "Every colleague paid attention, but every student didn't."

Consider again the sentence (7.5), "Every colleague paid attention, but every student didn't." Above we used a disjunction of group parallelism literals to model the semantics of this sentence – see Fig. 7.7. Now we can use a single jigsaw parallelism literal instead, as shown in Fig. 7.13. For the source sentence, there are fragments for "every colleague and "paid attention", and for the target sentence, there are fragments for "every student" and "not". The relative scoping of these latter two fragments is left open. Furthermore there is the jigsaw parallelism literal that models the ellipsis. Intuitively it states that the source sentence semantics except for the contribution of "every colleague", and except for a singleton segment term X_2/X_2, is isomorphic to the target sentence semantics except for the contribution of "every student", and except for the contribution of "not". Additionally, the singleton segment term in the source sentence semantics must be "in the same position" as the segment term Y_3/Y_4 in the target sentence semantics. In this way, the singleton segment term in the source sentence semantics restricts the "not" fragment in the target sentence semantics to the correct positions: It may be situated either above the "every student" fragment, or between the "every student" fragment and the copy of the "paid attention" fragment. Note that we could also have formulated the jigsaw parallelism literal for Fig. 7.13 as $\big((X_0/)\,/\,(X_1/, X_2/X_2)\big) \sim \big((Y_0/)\,/\,(Y_1/, Y_3/Y_4)\big)$.

7.5 A Look at Other Formalisms

We now take a look at related approaches to modeling ellipsis to see whether they can handle the kind of cases that we have discussed in this chapter. An approach that is

especially interesting to compare to CLLS is the one by Dalrymple, Shieber and Pereira [30] (DSP). As sketched in Chapter 2, Sec. 2.5.2, they model ellipsis using higher order unification. The sentence we have been discussing above, "Dan left, but George didn't", is a variant of a sentence that they analyze, shown here as (7.7). They model the semantics of this sentence by the formula (7.8) together with the equation (7.9): The semantics of source and target sentence share some property P, which in the source sentence is stated of "Dan" and "not" and in the target sentence of "George" and the identity function (i.e. an empty element).

(7.7) Dan didn't leave, but George did.

(7.8) $neg(left(dan)) \wedge P(george)(\lambda x.x)$

(7.9) $P(dan)(neg) = neg(left(dan))$

The solution that is computed for (7.8) and (7.9) is $P = \lambda x \lambda Q.Q(left(x))$, which gives the correct semantics for the target sentence.

But now consider again the sentence (7.2), "Dan left, but George didn't", which we have studied above. For this sentence the DSP formalism derives the formula (7.10) and the equation (7.11).

(7.10) $left(dan) \wedge P(george)(neg)$

(7.11) $P(dan)(\lambda x.x) = left(dan)$

One solution we get is $P = \lambda x \lambda Q.Q(left(x))$, which indeed corresponds to the correct meaning of the sentence. But unfortunately we also get wrong solutions such as $P = \lambda x \lambda Q.left(Q(x))$. The core of the problem is that HOU performs silent beta conversions, to the effect that it can no longer easily distinguish between the different *occurrences* of $\lambda x.x$ in $P(dan)(\lambda x.x)$. We could say that what is missing is a way of fixing the *position* of the "neg". Interestingly, this is similar to the problem of fixing the position of the excluded tree segment in the CLLS approach, which we have solved by demanding "correspondence" of alpha-gamma trees in Def. 7.13.

This particular example could be saved by imposing well-typedness restrictions, but the general problem remains. (Lappin and Shih [82] comment on problems of DSP with cases where the target sentence contains additional adjuncts, as in example (7.6).)

The ellipsis analysis of Crouch [28] is closely related to DSP and cannot handle examples like sentence (7.2) for similar reasons. The approach of Schiehlen [106] does not share this problem, but he pays for this by having to explicitly specify all the parallel material.

7.6 Discussion

In this section we raise three points: We look at remaining problems with modeling ellipsis, we speculate on a procedure for processing jigsaw parallelism literals, and we consider the problem of automatically deriving the semantics of elliptical sentences.

7.6.1 Modeling Ellipsis

Jigsaw parallelism literals are a very flexible tool for modeling the semantics of elliptical sentences. However, I believe that with a further extension it can be made even more useful: Group parallelism and jigsaw parallelism could be combined by allowing group parallelism literals to incorporate jigsaw segment terms as well as normal segment terms. This yields literals

$$(A_1, \ldots, A_n) \sim (B_1, \ldots, B_n)$$

where the A_i and B_i are jigsaw segment terms. Again, this does not raise the expressivity of the formalism. These new extended group parallelism literals can be used, for example, to model sentences like the following:

(7.12) Every man kissed his wife before John did.

The point about this sentence is that there is a scope ambiguity between "every man" and "before". The sentence has three readings, which can be sketched as follows:

(7.13) (every man)($\lambda x.x$ kissed x's wife, then J kissed J's wife).

(7.14) (every man)($\lambda x.x$ kissed x's wife), then (every man)($\lambda x.$ J kissed x's wife)

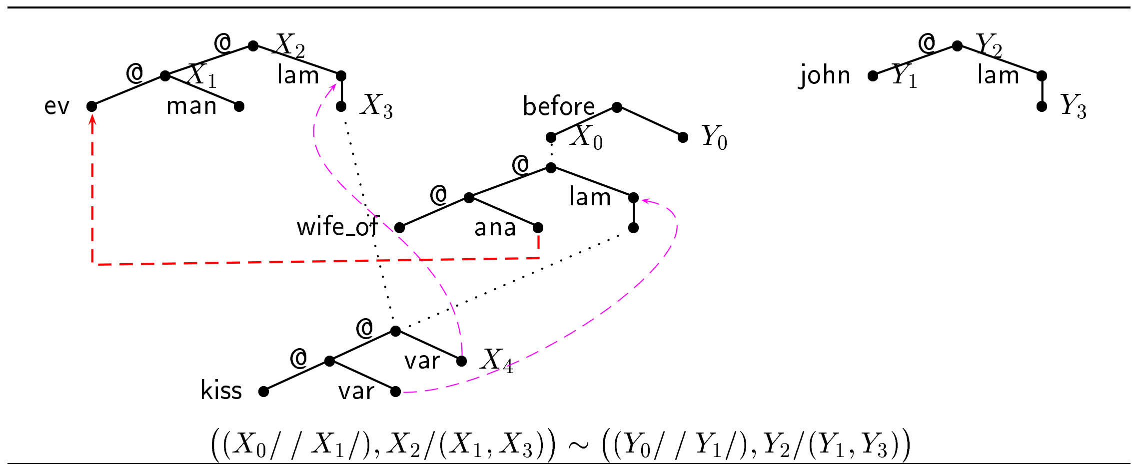
(7.15) (every man)($\lambda x.x$ kissed x's wife, then J kissed x's wife)

The reading (7.13) is sloppy, while (7.14) is strict. The third reading, (7.15), is strict too. It differs from the second reading in that it has "every man" outscope "before". That is, while in (7.14) John waits until all men have finished kissing their wives before he starts kissing them, (7.15) is an "interleaved" reading.

It is this third reading that makes it impossible to model this elliptical sentence with normal parallelism, because here the semantic contribution of the source contrasting element, "every man", outscopes the root of the source sentence semantics.

The analysis of (7.12) is shown in Fig. 7.14. We first take a quick look at the "simpler" strict reading (7.14). In this case, X_2 is dominated by X_0, which makes the jigsaw literal behave exactly like the ordinary parallelism literal $X_0/X_1{\sim}Y_0/Y_1$.

The most interesting reading, however, is (7.15), where "every man" outscopes "before", i.e. $X_3{\triangleleft^*}X_0$. As the denotation of $X_1/$ is not part of the alpha-gamma tree of the left-hand jigsaw segment, Y_1 cannot be below Y_0 either. That means that the two gamma



$$\big((X_0/\ /\ X_1/), X_2/(X_1, X_3)\big) \sim \big((Y_0/\ /\ Y_1/), Y_2/(Y_1, Y_3)\big)$$

Figure 7.14: Constraint for sentence (7.12): "Every man kissed his wife before John did."

segments $X_1/$ and $Y_1/$ do not overlap with $X_0/$ and $Y_0/$ at all, so the first pair of (jigsaw) segment terms in our extended group parallelism literal force the two subtrees below X_0 and Y_0 to be completely parallel. The second pair of segment terms ensures the correct lambda binding: the correspondent of X_4 must be bound at the right child of Y_2. This binding also forces Y_3 to dominate the correspondent of X_4. The group parallelism rules for anaphoric binding further ensure that the strict/sloppy ambiguity in our sentence (7.12) is modeled correctly.

7.6.2 Processing Jigsaw Parallelism Literals

How can jigsaw parallelism literals be processed? Jigsaw parallelism can be expressed by group parallelism. So one possible approach is to disambiguate the position of the gamma segment terms and then to solve a jigsaw parallelism literal by solving the appropriate group parallelism literal, using the procedure $\mathcal{P}_{\mathrm{gr}}$ of the previous chapter.

But a much more appealing solution would be to proceed in a similar way as the procedure $\mathcal{P}_\beta$ for underspecified beta reduction steps: to devise a jigsaw parallelism constraint procedure that copies a variable as soon as it is known that it cannot be inside any of the gamma segment terms, and likewise copies dominance, inequality, and labeling literals whenever it can safely do so. This procedure could again make use of the *underspecified correspondence literals* that we have introduced in the previous chapter. The reason why I think such a procedure would be sufficient is that in the linguistic application, gamma segment terms are usually fragments.

7.6.3 Deriving the Semantics of Elliptical Sentences

For the language CLLS, there exists a syntax/semantics interface [41]. Based on the syntactic structure of a sentence, it constructs a CLLS constraint that represents the

semantics of that sentence. For elliptical sentences, it generates the appropriate parallelism constraint linking the source and target sentence semantics (provided that the source and target sentences are known, as well as the contrasting elements). Can this approach be extended to generating jigsaw parallelism constraints? As the contrasting elements are known, it is not hard to generate the appropriate gamma segment terms. The only problem is to constrain the position of gamma segment terms, in the way that we did with the singleton gamma segment term X_2/X_2 in Fig. 7.13. One possibility is to impose less constraints on the position of the gamma segment terms – in the example in Fig. 7.13, this could be done by omitting the dominance $X_2 \triangleleft^* U$ – and to infer the proper position later.

7.7 Summary

In this chapter we have extended the definition of parallelism, turning it into a very flexible tool for replacing tree parts by other tree parts, within a fully declarative formalism.

We have pointed out that the semantics of a source or target sentence may consist not of a single segment but of a group of segments. This may be the case for example when negation or adjuncts are involved. Such cases can be handled using *group parallelism* instead of normal parallelism for modeling the ellipsis. Group parallelism is an extension to parallelism that we have introduced in the previous chapter.

Furthermore an ellipsis may possess contrasting elements partaking in scope ambiguities. In some cases this means that the sentence semantics cannot be described by a single group parallelism, only by a disjunction of group parallelism literals. To address this problem we have introduced *jigsaw parallelism.* In jigsaw parallelism, the excluded segments are explicitly stated, along with a maximal included segment. The jigsaw parallelism relation is strictly less expressive than the group parallelism relation. However in a jigsaw parallelism literal, the position of an excluded segment term may be more underspecified than is possible in a group parallelism literal, such that a single jigsaw parallelism literal can describe the semantics of elliptical sentences that would require a disjunction of group paralellism literals.

Chapter 8

Modeling Ellipsis: A Comparison of Approaches

The central topic of this thesis is parallelism, especially the processing of parallelism constraints. But while we concentrate on the formal aspects of parallelism, we also have to address the question of how it fares as a model.

In this chapter, we attempt an assessment of the CLLS approach to modeling ellipsis. First we need to clarify what exactly it is that we mean by "modeling ellipsis". We do this by asking three questions: First, what is the nature of ellipsis phenomena? Second, what problems need to be solved in connection with ellipsis phenomena? Third, which is the level of linguistic structure on which a formalism for modeling ellipsis should act?

We will use the third question to structure and group different ellipsis formalisms. We compare the analyses they propose and the classes of examples that they cover. On the basis of this comparison, we reach a tentative assessment of the CLLS approach to modeling ellipsis.

8.1 What is the Nature of Ellipsis?

Perhaps the earliest theory of the nature of ellipsis was to see it as *deletion* within the framework of generative syntax, as Sag [105] does. In this tradition, the surface form of a sentence is generated by a sequence of transformations on several underlying levels of representation. Ellipsis then arises from a removal of whole syntactic constituents, or just of their phonological features.

Another widely held view regards ellipsis as *reconstruction*: Some material is copied or generated into the place of an empty element. This element has been left empty throughout the syntactic analysis, up to the point at which reconstruction takes place. A comparison of these two views on ellipsis can be found in a paper by Williams [117].

A third possibility is to regard ellipses as a kind of anaphora, and thus to handle them in an anaphora resolution framework [59, 5]. This approach is similar to the reconstruction view, but the kind of identity that holds between the target sentence and its antecedent is a different one here: It is *referential* identity.

8.2 What Problems Need to be Solved in Connection with Ellipsis?

Johnson [63] asks three questions in connection with ellipsis (specifically VP ellipsis, the kind of ellipsis that we have mostly been considering throughout this thesis):

1. In which syntactic environments is VP ellipsis licensed?
2. What structural relation may a VP ellipsis and its antecedent have?
3. How is the meaning of the ellipsis recovered from its antecedent?

The first question is: Suppose we have a sentence (or a sequence of sentences) in which elements of similar meaning occur twice, but in one instance these elements are not expressed on the surface, under which circumstances do we get a well-formed sentence? An especially interesting question in this context is: Are there other phenomena with the same, or similar, licensing conditions? There is a large body of work on this, reviewed e.g. in Johnson's overview article [63].

The second question asks how, given an elliptical target sentence, we can determine the matching source sentence. There are only few papers on this topic; Gregory and Lappin [57, 81], Hardt and Romero [60, 61], Ginzburg and Cooper [53] have worked on it, as well as Egg and Erk [39] for the CLLS approach. The most comprehensive account is the one by Hardt [60], who uses heuristics to find the most suitable antecedent candidate. He stresses the need to take many different factors into account. The analysis by Egg and Erk, which uses the CLLS framework, determines source sentence candidates from strict syntactic conditions and then generates the appropriate parallelism constraint. But while this approach at the moment uses solely syntactic information, and only secure knowledge, it has been designed to be extensible to other sources of information and to an integration of preferences with secure knowledge.

The third question is the one that we have been concerned with in this thesis: Given both the source and the target of an ellipsis, how can the meaning of the target sentence be determined? It is with respect to this question that we will compare different ellipsis approaches in Sec. 8.4.

8.3 At Which Level of Linguistic Structure should an Ellipsis Theory be Situated?

Ellipsis theories can be distinguished by the types of *linguistic structure* that they access. In this section we first briefly list different types of linguistic structure that can be distinguished, then we recount arguments for why an account of ellipsis should have recourse to one type of structure or another.

There are several *levels* of linguistic structure, dimensions of linguistic information. They relate the *surface form* of an expression to the *meaning* that it conveys. For example,

"near the surface" one normally distinguishes at least prosody, morphology, and some kind of (surface) syntactic structure. The level of linguistic structure closest to the "meaning" side is taken up by the *proposition* that an expression conveys. Some theories add a level of *deep* syntactic structure between surface syntactic structure and the proposition level [17]. The difference between the two syntactic structures is that in deep syntactic structure, arguments are closer to the positions they occupy in the semantic structure than they are in surface syntactic structure. Another intermediate level posited in some theories is the level of *logical form* [19, 88], at which in particular scope is represented and disambiguated. The rules for deriving logical form from surface structure are subject to syntactic constraints, hence logical form, like deep and surface syntactic structure, is seen as a syntactic level of representation.

Basically, there are two ways in which these levels can interact. A traditional, Chomskyan perspective is that each level lives on a separate *stratum* of representation. In such a theory, each stratum can interact only with its immediate neighbors. An example of such a theory is Government & Binding [19]. The other possibility is to combine several levels into a single, multidimensional representation. That is, even though we still distinguish the different levels of linguistic structure, we only have a single stratum of representation. In such a monostratal approach, all levels can (in principle) interact. An example of such a theory is HPSG [97].

Before we get back to the subject of ellipsis, there is one further remark to be made about the notions of "syntax" and "semantics": We use the term *syntax* to talk about levels of linguistic structure that describe how surface form can be organized; by *semantics* we mean those levels of structure that describe how meaning is structured – where this is just meant to be a loose distinction, not an exact classification.

Now, coming back to the subject of ellipsis, there has been a long-standing dispute on whether ellipsis should be analyzed on a syntactic or a semantic level. Both sides can show examples to bolster their claim – on the one side sentences that point to *syntactic* constraints on the licensing of ellipsis, and on the other side sentences where (*semantic*) inferences are required to find the ellipsis antecedent. We briefly present a few examples for each of the two sides.

Sentences (8.1) through (8.3) have been used to argue that syntax must play a role in the resolution of ellipsis.[1] The point is that the unacceptability of these sentences is predicted by syntactic constraints, and that this only comes to bear if the treatment of an elliptic construction is tried within the syntactic structure and fails. For sentences (8.1) and (8.2) the syntactic constraints come from Binding Theory [19], which makes predictions on anaphoric binding based on relative positions of nodes in a syntactic tree: In (8.1) a pronoun gets bound in an inadmissible way, and in (8.2), a proper name gets bound, which is not allowed. The difference in acceptability between sentences (8.4) and (8.3) can be explained using the *subjacency* principle [18]: In this theory, some word order phenomena are explained by *movement* – elements are moved from the position they had

[1] The "*" at the beginning of an example sentence indicates that it is not well-formed.

in an underlying level of representation to the position they have in surface structure –, and by the subjacency principle movement is blocked if it crosses the boundaries of two or more designated classes of constituents. We only list the sentence here to give an impression of the kind of arguments raised for a syntactic analysis of ellipsis; see e.g. Kehler [70] for a detailed discussion of examples like these.

(8.1) * John_1 blamed himself_1, and Bill_2 did too. [blamed him_1] [73]

(8.2) * I hit Bill_1, and he_1 did, too. [hit Bill_1] [47]

(8.3) * John read everything which Bill believes the claim that he did. [58]

(8.4) John read everything which Bill believes he did. [58]

On the other hand, it has been argued that sentences (8.5) through (8.10) can only be properly analyzed by taking recourse to some level of semantic structure. The point is that in all these cases, there is no source sentence of appropriate form in the syntactic structure. In (8.5) the target sentence means "but he can't speak anymore", so the noun "speaker" is the antecedent. In (8.6) the target sentence is in the active voice, but we have a passive source sentence. Sentences (8.7) and (8.8) are examples of *split antecedents*: In (8.7) the target sentence must mean something like "neither of them can do what he or she wants to do", and in (8.8) the target sentence means "Gerry can talk and chew gum". In sentence (8.9) the preferred reading of the target sentence is something like "just as $[\text{all schoolboys}]_1$ give their_1 girlfriends their_1 school pictures." Kehler [71] suggests that this reading may be derived as part of an inference process of generalization.

(8.5) Harry used to be a great speaker, but he can't anymore, because he lost his voice. [59]

(8.6) This problem was to have been looked into, but obviously nobody did. [70]

(8.7) Wendy is eager to sail around the world and Bruce is eager to climb Kilimanjaro, but neither of them can because money is too tight. [116]

(8.8) I can walk, and I can chew gum. Gerry can too, but not at the same time. [116]

(8.9) Mary's boyfriend gave her his school picture, just as all schoolboys do.[71]

(8.10) Every linguist attended a workshop. Every computer scientist did, too.

Also, quantifier parallelism sentences like (2.6), repeated here as (8.10), have been put forward as an argument for an analysis on a semantic level, since they show an interaction of scope ambiguities with ellipsis. But this depends on where one would draw the dividing line between syntax and semantics; phenomena like quantifier scope and anaphoric binding have been assigned to semantic structure by some theories, by others to syntactic logical form.

8.4 Approaches to Modeling Ellipsis

In this section we discuss different theories of how the meaning of the elliptical target sentence can be determined. We first discuss approaches that treat ellipsis solely on a syntactic level, then approaches that analyze the phenomenon on a semantic level, finally approaches that take multiple sources of information into account.

8.4.1 Syntactic Approaches

Theories like the ones by Sag, Williams, Fiengo and May, and Lappin and Shih [105, 117, 47, 82] handle ellipsis at some level of syntactic structure, either at the surface syntactic level or at the level of syntactic logical form.

To give an example of a treatment of ellipsis at a level of logical form, Fiengo and May [47] study VP ellipsis in the context of a discussion of anaphoric binding and thus focus on strict/sloppy ambiguities. They consider the target sentence VP as a copy of the source sentence VP, where the anaphoric binding need not be the same (i.e. strict). For the sloppy reading, there is a *parallelism condition* on anaphoric binding: Index change is permitted between antecedent and ellipsis if the indexed elements participate in *parallel dependencies*, where a dependency is a sequence of syntactic categories connecting a dependent category with its antecedent. Furthermore Fiengo and May allow for *vehicle change*: Some syntactic properties may be different between matching source and target anaphora, for example "he" in the source sentence may change to "she" in the target.

Lappin and Shih [82], on the other hand, recover the missing material in the target sentence within the surface syntactic structure. They take the head verb of the source sentence, copy it to the target sentence and then fill all argument positions: if an argument of an appropriate type is present in the target sentence, they use that, otherwise the argument occupying the same slot in the source sentence is copied.

8.4.2 Semantic Approaches

Now we turn to theories that propose an analysis of ellipsis on some level of semantic structure. We have already discussed a number of such theories in Sec. 2.5.2, where we have listed ellipsis approaches related to the CLLS analysis. So now we just briefly recuperate that previous discussion.

The "classical" semantic analysis of ellipsis is the one by Dalrymple, Shieber and Pereira [30] (DSP). In this theory, the same property is expressed of the source contrasting elements and the target contrasting elements. For the sentence "John sleeps, and Mary does, too", for example, the meaning of the target would be $P(mary)$ for some property P of which we know (from the meaning of the source) that $P(john) = sleep(john)$. By solving this equation using higher-order unification (HOU), the meaning of the target sentence is retrieved.

Crouch [28] follows the same idea basic as DSP, but restricts the formalism to substitution, achieving a clean distinction of the modeling and the enumeration of readings (while in the DSP analysis the order of discharge of scope bearers determines the scope reading). Schiehlen [106] treats ellipsis in an UDRT setting, using index sequences to ensure the right interaction of scope and ellipsis. Schiehlen also discusses cases where inference is required to construct an adequate source sentence, like the split antecedent sentence (8.7). However in this framework scope bearers have to be explicitly included in the parallelism, while in DSP and the CLLS analysis any material inbetween the root and the holes of a parallel segment is automatically included in the parallelism.

The approach of Hardt [59] focuses on the similarities of anaphora and ellipsis. Using a DRT setting, this analysis places possible source sentences as referents into the universe. The target sentence can then refer to the appropriate source referent. In this theory examples where the source sentence can only be found by inference, like e.g. (8.5) or (8.7), play an important role. Asher [5] puts up a hierarchy of abstract entities that anaphora can refer to and discusses the kinds of abstract entities that can serve as ellipsis antecedents. He proposes an operation of *Concept Abstraction* within the DRT framework; this operation extracts abstract entities that are suitable antecedents.

8.4.3 Hybrid Approaches

In Sec. 8.3 we have seen a list of examples that seem to indicate that in determining the meaning of an ellipsis, access to some level of syntactic structure is needed, and likewise sentences that have been used to argue that access to semantic structure is necessary. Some theories suggested that this evidence means that multiple sources of information have to be taken into account.

Lappin [80] proposes to use completely different mechanisms for different types of ellipsis: a treatment within surface syntactic structure, a semantic HOU approach, and a ''quantifier storing" approach similar to Cooper storage [25].

But a uniform analysis for all kinds of ellipsis is aesthetically more pleasing; indeed, if ellipsis is perceived as a single phenomenon, there should be a uniform treatment for all its forms. In Kehler's [70] analysis, discourse structure plays a central role, in particular the *coherence relation* between source and target sentence. The question of whether syntactic or semantic information is needed for the treatment of an ellipsis becomes a question of discourse inference: If recourse to syntactic information is necessary during discourse inference, then the missing elements have to be recovered within the syntax, otherwise a semantic process of anaphora resolution suffices.

Kempson [72] takes a proof-theoretic approach to natural language interpretation. The formalism she uses is Labelled Deductive Systems [48]. The approach combines semantic and pragmatic aspects and can also incorporate syntactic information. For handling ellipsis, a higher-order variable is used in a similar way as in the DSP approach, but the variable is determined not by solving an equation but by inference – which should cover examples like (8.7).

8.5 A Tentative Assessment of the CLLS Approach to Modeling Ellipsis

In this section we attempt an assessment of the CLLS approach to ellipsis. First we position it with respect to the three questions we have raised in Sec. 8.1, 8.2, and 8.3. Then we reflect on the question which other approaches the CLLS analysis is most similar to. Finally, we look at possible frameworks in which this approach might be applied.

In Sec. 8.1, 8.2, and 8.3 we have raised three questions: What is the nature of the ellipsis phenomenon? Which problems need to be solved in connection with ellipsis? At which level of linguistic structure should an ellipsis theory be situated? So which positions does the CLLS approach take with respect to these questions?

- Concerning the nature of ellipsis, we have distinguished theories viewing it as either deletion, reconstruction, or reference. The CLLS approach is neutral with respect to this question. Even though the CLLS procedure of Chapters 3, 4 and 5 looks like it were performing reconstruction, this is not inherent in the modeling of ellipsis.
- The main problem with respect to ellipsis that we have been discussing here is: How can the meaning of the elliptical target sentence be determined? In the CLLS approach, the meaning of the target is determined by relating the semantics of source and target sentence by a parallelism literal, excluding the semantic contributions of the contrasting elements.
- CLLS, a formalism for underspecified semantics, models ellipsis within the semantic structure. But while this approach *recovers the meaning* of the target sentence within the semantic structure, it is clear that many factors contribute to *determining the antecedent.*

How can we position CLLS analysis in relation to other approaches? The CLLS analysis is similar to DSP: They are both unification-like approaches that determine the meaning of an ellipsis within the semantic structure. CLLS uses Kehler-style link chains to model the interaction of ellipsis and anaphora [69, 70]. Used in combination with dominance constraints, it integrates an underspecified treatment of scope ambiguities with an analysis of ellipsis, yielding the right results for quantifier parallelism cases like (8.10).

For a clear understanding of the formalism, we think the differences between DSP and the CLLS analysis are especially interesting. We repeat the most important differences (which we have noted in passing in different places):

- While DSP uses general higher-order variables to describe the structurally identical areas, the CLLS analysis uses a less expressive fragment, parallelism constraints, which are equally expressive as context unification.
- In DSP the integration of scope and ellipsis is procedural, depending on the order in which scope-bearers are discharged, in CLLS it is completely declarative: There

is a clean distinction between the description of all readings of a sentence in the form of a CLLS constraint on the one hand, and the enumeration of the readings using a procedure like $\mathcal{P}$ on the other hand.

- HOU and CLLS differ in their perspective on trees. HOU adopts the external perspective, talking about properties of trees, while CLLS takes the internal perspective, talking about relations between nodes of a single tree. This makes a difference in the way that the material excluded from parallelism is specified. For example, the equation $P(john) = see(john, john)$ can affect none, either one, or both occurrences of the subtree *john* in $see(john, john)$ (if we leave the issue of primary versus secondary occurrences aside for a moment). Parallelism $\pi_0/\pi_1, \ldots, \pi_n \sim \psi_0/\psi_1, \ldots, \psi_n$ on the other hand specifies that there are exactly n exceptions and gives their exact positions in terms of the tree nodes at which they start.
- Strict/sloppy ambiguities are handled differently: The mechanism that DSP uses is unification (which, as we have just said, can pick out all "john"-formed subtrees automatically) together with the primary/secondary occurrence restriction. CLLS, on the other hand, uses link chains as first introduced by Kehler [69] – see Sec. 2.3.4.
- Another difference is that in HOU, β and η conversions are built-in, but not so in CLLS. Underspecified beta reduction *can* be performed (we have shown a procedure in Chapter 6), but this does not happen automatically.

In the previous sections, we have raised some interesting issues about modeling ellipsis, which, however, go beyond the scenario of the CLLS approach as we have presented it in this text.

- In examples (8.5) through (8.9), some kind of inference is needed for deriving the source sentence. Could such examples be analyzed using parallelism constraints? In principle, yes, by delaying the statement of the parallelism constraint that models the ellipsis: First derive just a dominance constraint modeling the semantics of the sentence except for the ellipsis; use some form of inference to derive the source sentence (of course, this is the crucial point, just like in any other approach relying on inference within the semantic structure, but with CLLS the situation is exacerbated because we need to do direct deduction on an underspecified structure); then state a parallelism constraint to model the ellipsis.
- We have discussed ellipsis approaches that argue that access to multiple sources of information is needed, notably the approaches by Kehler [70] and Kempson [72]. Both approaches have at their core a process operating on semantic structure. So could the CLLS analysis form the core of such a multi-level approach to ellipsis? Yes – both Kempson and Kehler name HOU as one possible mechanism for recovering the target sentence semantics, and we have already discussed the similarity between the HOU approach and parallelism constraints. However, this would again require performing direct deduction on an underspecified semantic representation.

But why use CLLS in such a multi-level analysis? One advantage is the underspecified framework in which the interaction of scope ambiguity and ellipsis is modeled. Another advantage lies in processing: the parallelism constraint procedure of Chapter 4 performs well on constraints from the linguistic application, and there may even be a decidable fragment of CLLS that suffices for handling ellipsis – we take up both these points in the following chapter.

8.6 Summary

In this chapter we have first discussed issues connected with modeling ellipsis, in the form of three questions: First, what is the nature of ellipsis? Is it a phenomenon of deletion, reconstruction, or reference? Second, what problems need to be solved in connection with ellipsis? The problem we focus on is to determine the meaning of the elliptical target sentence. Third, at which level of linguistic structure should an ellipsis theory be situated? We have seen that there are arguments both for an analysis in syntactic structure and an analysis within semantic structure.

We have given a brief overview of approaches to modeling ellipsis, structured by whether they access some level of syntactic structure, semantic structure, or both, and we have stated a tentative assessment of the CLLS approach: It determines the meaning of the elliptical target sentence within the semantic structure (while for determining the *antecedent* multiple levels of linguistic structure have to be taken into account), in a style similar to DSP [30], but there are important differences in the perspective on trees as well as in the question of processing. CLLS is neutral with respect to the question of the nature of ellipsis, and it could in principle form the core of a multi-level analysis of ellipsis.

Chapter 9

Outlook

This chapter presents ideas for future work. We first briefly list a number of open questions, then we focus on the first two in the list.

- In this thesis we have presented an abstract semi-decision procedure for CLLS. How could it be turned into a practical procedure for CLLS?
- The decidability of CLLS is still an open problem. Can we find fragments of the language with good properties?
- We have presented two procedures for performing a single underspecified beta reduction step. They both have drawbacks: One procedure uses more distribution than necessary for just performing a beta reduction step, the other procedure is incomplete.

  One possibility would be to combine the two procedures but place strong restrictions on the application of distribution rules. The idea is to solve a beta reduction formula without distribution whenever that is possible, and to resort to a strongly controlled application of distribution otherwise.

  Another possibility would be to extend the incomplete procedure. The problematic cases are those with nonlinear redexes. The problem could maybe be solved by reducing *groups* of redexes at the same time, and requiring redexes generated as copies of the same original redex to be reduced simultaneously, as sketched in Chapter 6.
- We have presented two applications of the language CLLS. Which other areas can CLLS be applied to? In particular, are there other kinds of ambiguity for which CLLS can furnish an underspecified description?

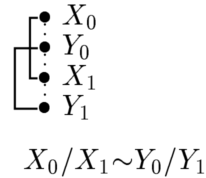
  One possible application is an underspecified description of discourse structure [107]. Furthermore, a recent application to parsing with resource-sensitive categorial grammar [45] uses not CLLS itself, but a variant based on finite set constraints that has previously been used for an implementation of dominance constraints [34]. The variant was chosen because it allows stronger statements about dominance.
- Among the questions on ellipsis that we have raised in Chapter 8, one was: Given an ellipsis, how do we determine its most likely antecedent?

> It seems that multiple sources of information have to be considered to determine the most suitable antecedent, and that certain knowledge is involved as well as preferences [60, 61]. First steps towards an analysis within the CLLS framework have been made [39], focusing on a subclass of ellipsis phenomena.

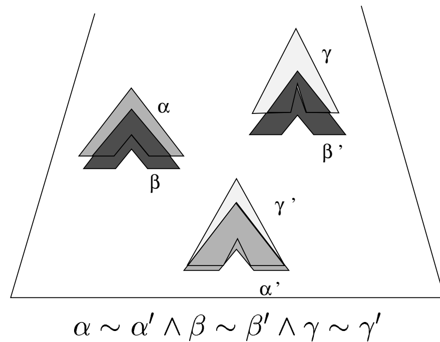
In the following two sections, we focus the first two questions in the list.

9.1 A Decidable Fragment of CLLS

Can we find decidable fragments of CLLS with good processing properties? Intuitively, the cases that are problematic for the CLLS procedure that we have presented are those with "self-overlapping" parallelism constraints. The simplest such constraint is shown in Fig. 4.7 (a), repeated here to the right. But self-overlap need not be as obvious as it is in that case. It can also occur in constellations as the one sketched in Fig. 9.1. In this picture, equal-colored segments are parallel. A subsegment of α also belongs to β and is parallel to a subsegment of β'. A sub-subsegment of this also belongs to γ and reappears in γ', where it overlaps with α'.



$X_0/X_1{\sim}Y_0/Y_1$



$$\alpha \sim \alpha' \wedge \beta \sim \beta' \wedge \gamma \sim \gamma'$$

Figure 9.1: Sketch of a more complex case of "self-overlap"

How can the idea of prohibiting "self-overlap" be exploited for a decidable fragment? We are currently working on a fragment of CLLS in which overlap is forbidden altogether: Any two tree segments involved in the parallelism relation (but not necessarily parallel to each other) may not overlap unless one is properly nested in the other. Parallelism constraints are then interpreted not over the parallelism relation in general but over this restricted parallelism relation. This fragment of CLLS is decidable, in fact satisfiability testing is only NP-complete, which is the same as for dominance constraints. Furthermore it seems that for all CLLS constraints that arise in modeling ellipsis, this restricted fragments suffices.

Can a bigger decidable fragment be defined in the same vein? Is it possible to prohibit solely "self-overlap", not overlap in general? The notion of "self-overlap", though intuitively clear, is not easy to define formally. One possibility could be to make use of the

notion of correspondence functions, employing the transitive closure of the correspondence relationship to detect "overlap cycles" like the one in Fig. 9.1.

Normal dominance constraints [76, 3] are a fragment of dominance constraints for which satisfiability can be tested in polynomial time. Here the concept of *fragments* (see p. 32) is central: In normal dominance constraints fragments cannot overlap, and satisfiability becomes a problem of arranging the fragments in such a way that all dominance constraints between them are satisfied. Can this language fragment, which is important in the linguistic application, be extended by parallelism constraints, or a decidable fragment of parallelism constraints? The answer is not obvious because some parallelism constraints (which also occur in the linguistic application) cancel normality.

9.2 Processing CLLS Constraints

For dominance constraints, the first solver was a high-level saturation algorithm [78]. Building on this, a solver based on constraint programming techniques [34] was proposed, a solver that already shows good average-case behavior. Then normal dominance constraints were introduced, along with a polynomial graph-based solver [3].

For parallelism constraints, and for CLLS in general, there now exists a high-level saturation procedure. So the next step is to develop practical procedures for this larger language.

One possibility of doing this is to interleave a dominance constraint solver with a copying step that makes explicit the structural isomorphism of one pair of segment terms. Preferably the language used should be a decidable fragment of parallelism constraints, a fragment that yields a partial order on parallel segment term pairs, in such a way that the copying step only needs to be applied exactly once to each segment term pair.

Chapter 10

Conclusion

This chapter summarizes the main contributions of this thesis. The main formalism that we have studied in this thesis are *parallelism constraints*, which are part of the Constraint Language for Lambda Structures. A parallelism constraint states that two *segments* of a lambda structure are structurally isomorphic and have parallel bindings. The main result that we have presented is a *procedure for parallelism constraints*, which we have extended to a procedure for all of CLLS. In the second part of the thesis, we have studied questions of the practical applicability of the formalism as well as the procedure. We have considered two applications: underspecified natural language semantics, and underspecified beta reduction.

10.1 A Procedure for CLLS Constraints

We have introduced the semi-decision procedure $\mathcal{P}$ for CLLS constraints. It is a high-level, rule-based saturation procedure, consisting of saturation rules that add more and more literals to a constraint until a saturation is reached. The procedure $\mathcal{P}$ has the following properties:

- It terminates for the linguistically relevant constraints. For these constraints, it computes saturations that correspond to the correct readings.
- It includes a solver for dominance constraints. Given a dominance constraint as an input, the procedure behaves exactly like the dominance constraint solver that it encompasses, which is important for the linguistic application.
- It is built in a modular fashion, such that different dominance constraint solvers can be incorporated.
- It never has to guess labels.
- It introduces *correspondence formulas* as a data structure for handling parallelism within partial tree descriptions.

The procedure makes explicit the information that is present only implicitly in a constraint. From the point of view of the linguistic application, which uses dominance

constraints for modeling scope ambiguity and parallelism constraints for modeling ellipsis, the procedure enumerates scope readings and recovers the meaning of an elliptical sentence from its antecedent.

The notion of *correspondence* features both in the definition of the parallelism relation, in the form of correspondence functions, and in the CLLS procedure, in the form of correspondence formulas. A correspondence function maps each node in one parallel segment to the node at the same position in the other parallel segment. A correspondence formula states that correspondence holds between the denotations of two variables. It is expressed in terms of *path parallelism literals*, which state that two tree paths are isomorphic. The properties of path parallelism literals, expressed as saturation rules, enforce the correct interaction between different correspondence formulas. In using correspondence formulas as the central data structure, the CLLS procedure benefits from the node-centered perspective of the language.

The CLLS procedure $\mathcal{P}$ is sound in the sense that all its rules are equivalence transformations. The saturations that it computes are satisfiable: A model can be directly read off each saturation. The procedure is complete in the sense that it computes all *minimal* saturations for a given input constraint, in fact it *only* computes minimal saturations. We have defined minimality in terms of a family of partial orders $\leq_{\mathcal{G}}$, parametrized by a set $\mathcal{G} \subseteq \mathcal{V}ar$ of variables. This family of partial orders can be described as subset inclusion modulo α-renaming of variables introduced during computation with $\mathcal{P}$ (where $\mathcal{V}ar - \mathcal{G}$ is the set of variables that may be renamed).

10.2 Applying Parallelism Constraints

We have discussed two applications of CLLS: underspecified natural language semantics, and underspecified beta reduction. In natural language semantics, parallelism constraints can be used to model ellipsis. In underspecified beta reduction, parallelism can be used for a declarative description of the result of a single underspecified beta reduction step. For these applications, we have added the following extensions to the formalism and the procedure.

Group parallelism. Group parallelism relates two *groups* of parallel segments instead of just two segments. It differs from "normal" parallelism in its weaker conditions on lambda and anaphoric binding. This extension to parallelism is needed both in the application to modeling ellipsis and in the application to modeling underspecified beta reduction steps.

Jigsaw parallelism. Ordinary parallelism relates pairs of segments, subtrees from which one or more subtrees have been cut out. Jigsaw parallelism relates pairs of jigsaw segments, segments from which one or more segments have been cut out. The position of the included as well as the excluded segments is specified in terms of an extended notion of correspondence. Jigsaw parallelism is needed in the application to modeling ellipsis. It is subsumed by group parallelism, however it allows for a more flexible and elegant

modeling of ellipsis: Some cases of ellipsis require partial disambiguation of scope before they can be modeled using group parallelism; with jigsaw parallelism, the ellipsis can be described without any preceding disambiguation.

A procedure for CLLS plus group parallelism constraints. We have extended the CLLS procedure $\mathcal{P}$ to handle group parallelism constraints as well as more expressive constraints for lambda binding, which are necessary for the application to underspecified beta reduction.

A procedure for computing the result of an underspecified beta reduction step. In computing the result of an underspecified beta reduction step it is desirable to keep the description of the lambda termas underspecified as it was before beta reduction. In particular, the procedure should not disambiguate quantifier scope. We have presented a procedure that can perform an underspecified beta reduction step without any disambiguation for many examples from underspecified semantics. The procedure, which is a modification of the CLLS plus group parallelism procedure, relies crucially on the specific layout of the segment terms in a reducing tree, along with underspecified correspondence literals.

Bibliography

[1] 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, 1994.

[2] Hiyan Alshawi and Richard Crouch. Monotonic semantic interpretation. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 32–39, Kyoto, 1992.

[3] Ernst Althaus, Denys Duchier, Alexander Koller, Kurt Mehlhorn, Joachim Niehren, and Sven Thiel. An efficient algorithm for the configuration problem of dominance graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 815–824, Washington, DC, 2001.

[4] Krzysztof Apt. Logic programming. In *Handbook of Theoretical Computer Science*. Elsevier Science Publishers, 1990.

[5] Nicholas Asher. *Reference to Abstract Objects in Discourse*. Kluwer, Dordrecht, 1993.

[6] Franz Baader and Jörg Siekmann. Unification theory. In *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, 1993.

[7] Rolf Backofen and Gert Smolka. A complete and recursive feature theory. In *Proceedings of the 31th Annual Meeting of the Association for Computational Lingustics (ACL)*, pages 193–200, 1993.

[8] Patrick Blackburn. Modal logic and attribute-value structures. In Maarten de Rijke, editor, *Diamonds and defaults*. Kluwer, 1993.

[9] Patrick Blackburn and Wilfried Meyer-Viol. Linguistics, logic and finite trees. *Bulletin of the IPGL (Interest Group in Pure and Applied Logics)*, 2:2–39, 1994.

[10] Patrick Blackburn, Wilfried Meyer-Viol, and Maarten de Rijke. A proof system for finite trees. In Hans Kleine Büning, editor, *Computer Science Logic. Selected Papers of the 9th International Workshop CSL '95*, volume 1092 of *LNCS*, pages 86–105. Springer-Verlag, Berlin, 1995.

[11] Manuel Bodirsky. Beta reduction constraints. Master's thesis, Fachbereich 14 Informatik, Saarland University, 2001.

[12] Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Beta reduction constraints. In *International Conference on Rewriting Techniques and Applications (RTA)*, Lecture Notes in Computer Science, Utrecht, The Netherlands, 2001. Springer-Verlag, Berlin.

[13] Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Underspecified beta reduction. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 74–81, Toulouse, France, 2001.

[14] Manuel Bodirsky and Martin Kutz. Pure dominance constraints. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, Antibes - Juan le Pins, 2002.

[15] Johan Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1995.

[16] John Chen, Robert Frank, and K. Vijay-Shanker. Dominance, precedence, and c-command in description-based parsing. In *Proceedings of the XII Congress on Natural Languages and Formal Languages*, La Seu D'Urgell, September 1996.

[17] Noam Chomsky. *Aspects of the theory of syntax.* The MIT press, Cambridge, Massachusetts, 1965.

[18] Noam Chomsky. Conditions on transformations. In Stephen Anderson and Paul Kiparsky, editors, *A Festschrift for Morris Halle.* Holt, Rinehart and Winston, New York, 1973.

[19] Noam Chomsky. *Lectures in Government and Binding.* Foris, Dordrecht, 1981.

[20] Noam Chomsky and Morris Halle. *The Sound Pattern of English.* Harper and Row, 1957.

[21] The CHORUS demo system. Web page: `http://www.coli.uni-sb.de/cl/projects/chorus/demo.html`.

[22] Hubert Comon. Completion of rewrite systems with membership constraints. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 623 of *LNCS*, 1992.

[23] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata – techniques and applications. Available from `http://www.grappa.univ-lille3.fr/tata/`.

[24] Hubert Comon, Mehmet Dincbas, Jean-Pierre Jouannaud, and Claude Kirchner. A methodological view of constraint solving. *Constraints*, 4(4):337–361, December 1999.

[25] Robin Cooper. Towards a general semantic framework. In Robin Cooper, editor, *DYANA-2 Deliverable R2.1.A*, The DYANA-2 Project Administrator, pages 49–97. IILC, University of Amsterdam, 1993.

[26] Ann Copestake, Dan Flickinger, and Ivan Sag. Minimal Recursion Semantics. An introduction. Available from `ftp://ftp-csli.stanford.edu/linguistics/sag/mrs.ps`, 1997.

[27] Thomas Cornell. On determining the consistency of partial descriptions of trees. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 163–170, 1994.

[28] Richard Crouch. Ellipsis and quantification: A substitutional approach. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 229–236, Dublin, 1995.

[29] Östen Dahl. On so-called 'sloppy identity'. In *Gothenburg Papers in Theoretical Linguistics*, volume 11, University of Gothenburg, Sweden, 1973.

[30] Mary Dalrymple, Stuart Shieber, and Fernando Pereira. Ellipsis and higher-order unification. *Linguistics & Philosophy*, 14:399–452, 1991.

[31] Nicolas de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[32] John E. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[33] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In Harry C. Bunt and Elias G.C. Thijsse, editors, *Third International Workshop on Computational Semantics (IWCS-3)*, pages 71–85, Tilburg, NL, 1999.

[34] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL)*, LNCS. Springer, 2000.

[35] Denys Duchier and Stefan Thater. Parsing with tree descriptions: a constraint-based approach. In *Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP)*, pages 17–32, Las Cruces, New Mexico, 1999.

[36] Markus Egg. Reinterpretation by underspecification. Habilitation thesis, Saarland University, 2000. To appear.

[37] Markus Egg. Reinterpretation from a synchronic and a diachronic point of view. In Regine Eckhart and Klaus van Heusinger, editors, *Meaning Change - Meaning Variation*, number 106 in Arbeitspapiere der FG Sprachwissenschaft. Universität Konstanz, 2000.

[38] Markus Egg. Semantic construction for reinterpretation phenomena. *Linguistics*, 40:579–609, 2002.

[39] Markus Egg and Katrin Erk. A compositional account of VP ellipsis. In *8th International Conference on Head-Driven Phrase Structure Grammar*, Trondheim, Norway, 2001.

[40] Markus Egg and Michael Kohlhase. Underspecification of quantifier scope. In *Proceedings der 6. Fachtagung der Sektion Computerlinguistik der DGfS*, Heidelberg, 1997.

[41] Markus Egg, Alexander Koller, and Joachim Niehren. The Constraint Language for Lambda Structures. *Journal of Logic, Language, and Information*, 10:457–485, 2001.

[42] Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda-structures in semantic underspecification. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and 36th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 353–359, Montreal, Canada, 1998.

[43] Katrin Erk and Alexander Koller. VP ellipsis by tree surgery. In *Proceedings of the 13th Amsterdam Colloquium*. University of Amsterdam, 2001.

[44] Katrin Erk, Alexander Koller, and Joachim Niehren. Processing underspecified semantic representations in the Constraint Language for Lambda Structures. *Journal of Language and Computation*, 2001. To appear.

[45] Katrin Erk and Geert-Jan M. Kruijff. A constraint programming approach to parsing with resource-sensitive categorial grammar. In *Proceedings of the 7th International Workshop on Natural Language Understanding and Logic Programming (NLULP)*, 2002. To appear.

[46] Katrin Erk and Joachim Niehren. Parallelism constraints. In *International Conference on Rewriting Techniques and Applications (RTA)*, pages 110–126, Norwich, U.K., 2000.

[47] Robert Fiengo and Robert May. *Indices and Identity*. MIT Press, Cambridge, 1994.

[48] Dov Gabbay. *Labelled Deductive Systems; principles and applications. Vol 1: Basic Principles*. Oxford University Press, 1996.

[49] L. T. F. Gamut. *Language, Logic and Meaning*. Chicago University Press, 1991.

[50] Claire Gardent and Michael Kohlhase. Focus and higher-order unification. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, Copenhagen, Denmark, 1996.

[51] Claire Gardent, Michael Kohlhase, and Noor van Leusen. Corrections and higher-order unification. In *Proceedings of KONVENS*, Bielefeld, Germany, 1996.

[52] Mark Gawron and Stanley Peters. *Anaphora and Quantification in Situation Semantics.* Number 19 in CSLI Lecture Notes. CSLI/University of Chicago Press, 1990.

[53] Jonathan Ginzburg and Robin Cooper. Resolving ellipsis in clarification. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 236–243, Toulouse, France, 2001.

[54] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[55] Paul Gorrell. *Syntax and Parsing.* Cambridge University Press, Cambridge, 1995.

[56] Sidney Greenbaum and Randolph Quirk. *A Student's Grammar of the English Language.* Longman, Harlow, 1990.

[57] Howard Gregory and Shalom Lappin. Antecentent contained ellipsis in HPSG. In Gert Webelhuth, Jean-Pierre Koening, and Andreas Kathol, editors, *Lexical and constructional aspects of linguistic explanation*, pages 331–356. CSLI, Stanford, 1998.

[58] Isabelle Haïk. Bound variables that need to be. *Linguistics and Philosophy*, 10:503–530, 1987.

[59] Daniel Hardt. *Verb Phrase Ellipsis: Form, Meaning, and Processing.* PhD thesis, University of Pennsylvania, 1993.

[60] Daniel Hardt. An empirical approach to VP ellipsis. *Computational Linguistics*, 23:525–541, 1997.

[61] Daniel Hardt and Maribel Romero. Ellipsis and the structure of discourse. In *Proceedings of Sinn and Bedeutung VI*, Osnabrück, 2001. to appear.

[62] Paul Hirschbühler. VP deletion and across the board quantifier scope. In *Proceedings of the 12th Conference of the North East Linguistic Society (NELS)*, 1982.

[63] Kyle Johnson. What VP ellipsis can do, what it can't, but not why. In Mark Baltin and Chris Collins, editors, *Handbook of Contemporary Syntactic Theory*, pages 439–479. Blackwell Publishers, 2001.

[64] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree adjunct grammars. *Journal Computer Systems Science*, 10(1), 1975.

[65] Ron Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. The MIT Press, Cambridge, MA, 1982.

[66] Robert Kasper and William Rounds. A logical semantics for features structures. In *Proceedings of the 24th Meeting of the Association for Computational Linguistics (ACL)*, pages 257–266, 1986.

[67] Robert Kasper and William Rounds. The logic of unification in grammar. *Linguistics & Philosophy*, 13:33–58, 1990.

[68] Martin Kay. Functional grammar. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics (ACL)*, Berkeley, CA, 1979.

[69] Andrew Kehler. A discourse copying algorithm for ellipsis and anaphora resolution. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 1993.

[70] Andrew Kehler. *Interpreting Cohesive Forms in the Context of Discourse Inference.* PhD thesis, Harvard University, 1995.

[71] Andrew Kehler. Another problem for syntactic (and semantic) theories of VP-ellipsis. *Snippets*, January 2002. Online Journal at `http://www.ledonline.it/snippets`.

[72] Ruth Kempson. Semantics, pragmatics and deduction. In Shalom Lappin, editor, *Handbook of Contemporary Semantic Theory.* Blackwell, 1997.

[73] Yoshihisa Kitagawa. Copying identity. *Natural Language and Linguistic Theory*, 9:497–536, 1991.

[74] Alexander Koller. Evaluating context unification for semantic underspecification. In *Proceedings of the Third ESSLLI Student Session*, pages 188–199, 1998.

[75] Alexander Koller. Constraint languages for semantic underspecification. Master's thesis, Saarland University, 1999.

[76] Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 368–375, Hong Kong, 2000.

[77] Alexander Koller, Joachim Niehren, and Kristina Striegnitz. Relaxing underspecified semantic representations for reinterpretation. *Grammars*, 3(2/3), 2000. Special Issue on MOL'99.

[78] Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints: Algorithms and complexity. In Michael Moortgat, editor, *Third International Conference on Logical Aspects of Computational Linguistics (LACL)*, Grenoble, 2001.

[79] Markus Kracht. Syntactic codes and grammar refinement. *Journal of Language, Logic and Information*, 4:41–60, 1995.

[80] Shalom Lappin. The interpretation of ellipsis. In Shalom Lappin, editor, *Handbook of Contemporary Semantic Theory*. Blackwell, 1997.

[81] Shalom Lappin. An HPSG account of antecentent contained ellipsis. In Shalom Lappin and Howard Gregory, editors, *Fragments: studies in ellipsis and gapping*, pages 68–97. Oxford University Press, New York, 1999.

[82] Shalom Lappin and Hsue-Hueh Shih. A generalized reconstruction algorithm for ellipsis resolution. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, Copenhagen, Denmark, 1996.

[83] Jean-Louis Lassez and Michael J. Maher. Closures and fairness in the semantics of programming logic. *Theoretical Computer Science*, 29:167–184, 1984.

[84] Jordi Lévy. Linear second order unification. In *International Conference on Rewriting Techniques and Applications*. Springer-Verlag, 1996.

[85] Jordi Lévy and Mateu Villaret. Linear second-order unification and context unification with tree-regular constraints. In *International Conference on Rewriting Techniques and Applications (RTA)*, pages 156–171, Norwich, UK, 2000.

[86] Gennadií S. Makanin. The problem of solvability of equations in a free semigroup. *Mat. Sbornik.*, 103(2):147–236, 1977. In Russian; English translation in: *Math. USSR Sbornik* 32, 129-198, 1977.

[87] Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 129–136, 1983.

[88] Robert May. *Logical Form: Its Structure and Derivation*. MIT Press, Cambridge, Massachusetts, 1985.

[89] Richard Montague. The proper treatment of quantification in ordinary English. In Jaako Hintikka and Julius M. E. Moravcsik, editors, *Approaches to Natural Language*. Dordrecht, 1973.

[90] Martin Müller and Joachim Niehren. Ordering constraints over feature trees expressed in second-order monadic logic. *Information and Computation*, 159(1/2):22–58, 2000. Special Issue on RTA 1998.

[91] Reinhard Muskens. Order-independence and underspecification. In Jeroen Groenendijk, editor, *Ellipsis, Underspecification, Events and More in Dynamic Semantics*, DYANA Deliverable R.2.2.C. IILC, University of Amsterdam, 1995.

[92] Joachim Niehren and Alexander Koller. Dominance constraints in context unification. In *Third International Conference on Logical Aspects of Computational Linguistics (LACL)*, Grenoble, France, 1998.

[93] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification and one-step rewriting. In *Procedings of the International Conference on Automated Deduction*, pages 34–48, Townsville, Australia, 1997. Full version available from http://www.ps.uni-sb.de/Papers/abstracts/fullContext.html.

[94] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In *Proceedings of the 35th Annual Meeting of the Association of Computation Lingustics (ACL)*, pages 410–417, Madrid, Spain, 1997.

[95] Manfred Pinkal. Radical underspecification. In Paul Dekker and Martin Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*. University of Amsterdam, 1995.

[96] Manfred Pinkal. Vagueness, ambiguity, and underspecification. In Teresa Galloway and Justin Spence, editors, *Proceedings from Semantics and Linguistic Theory VI*, Rutgerts University, 1996.

[97] Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar.* CSLI and University of Chicago Press, 1994.

[98] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[99] Owen Rambow, K. Vijay-Shanker, and David Weir. D-tree grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 151–158, MIT, Cambridge, 1995.

[100] Uwe Reyle. Dealing with ambiguities by underspecification: Construction, representation, and deduction. *Journal of Semantics*, 10(2), 1993.

[101] Uwe Reyle. Co-indexing labelled DRSs to represent and reason with ambiguities. In Stanley Peters and Kees van Deemter, editors, *Semantic Ambiguity and Underspecification.* CSLI Publications, Stanford, 1995.

[102] James Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions: an applications to Tree-Adjoining Grammars. *Computational Intelligence*, 10(4), 1994.

[103] William C. Rounds. Feature logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, volume 2: General Topics, pages 475–533. Elsevier Science Publishers B.V. (North Holland), 1997.

[104] Decidability of context unification. The RTA list of open problems, number 90, `http://www.lsv.ens-cachan.fr/~treinen/rtaloop`, 1998.

[105] Ivan Sag. *Deletion and logical form.* PhD thesis, MIT, Cambridge, 1976.

[106] Michael Schiehlen. *Semantikkonstruktion*. PhD thesis, IMS, University of Stuttgart, 1999.

[107] Frank Schilder. An underspecified discourse representation theory (USDRT). In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and of the 36th Annual Meeting of the Association for Computational Linguistics (ACL)*, Montréal, Canada, 1998.

[108] Manfred Schmidt-Schauß. Unification of stratified second-order terms. Technical Report 12/94, J. W. Goethe Universität, Frankfurt, 1994.

[109] Manfred Schmidt-Schauß. An optimised decision algorithm for stratified context unification. Technical Report 13/00, J.W. Goethe-Universität, Frankfurt, 2000.

[110] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.

[111] Gert Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, pages 324–343. Springer-Verlag, Berlin, 1995.

[112] Patrick Sturt and Matthew Crocker. Monotonic syntactic parsing: a crosslinguistic study of attachment and reanalysis. *Language and Cognitive Processes*, 11(5):449–494, 1996.

[113] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, August 1968.

[114] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Formal Models and Semantics*, volume B, chapter 4, pages 133–191. The MIT Press, 1990.

[115] K. Vijay-Shanker. Using descriptions of trees in a Tree-Adjoining Grammar. *Computational Linguistics*, 18(4):481–517, 1992.

[116] Bonnie Webber. *A formal approach to discourse anaphora*. PhD thesis, Harvard University, 1978.

[117] Edwin Williams. Discourse and logical form. *Linguistic Inquiry*, 8(1):101–139, 1977.

[118] Feiyu Xu. Underspecified representation and resolution of ellipsis. Master's thesis, Saarland University, 1998.

Index

$(\mathcal{L}^\theta, \sigma)$, 31
$\mathrm{co}(A, B)$, 80
u-co_i, 162
u-co_i^-, 163
$\mathrm{co}_k(\overline{A}, \overline{B})$, 149, 162
$\mathrm{co}_k^-(\overline{A}, \overline{B})$, 149, 162
$\mathrm{co}^-(\overline{A}, \overline{B})(S_1){=}S_2$, 152
$\mathrm{p}(\because)$, 79
Σ, 25
$\mathsf{ar}(f)$, 25
$\mathrm{con}_\varphi(.)$, 66
$\mathsf{distr}_{\overline{A}}(.)$, 151
$\mathsf{distr}_{\overline{A}}^-(.)$, 151
`false`, 31
$hs(.)$, 27
$\mathsf{js}(.)$, 182, 185
λ, 29, 31
$\lambda(.){\neq}.$, 152
λ^{-1}, 146
$\mathcal{L}^\theta$, 29
$\mu_\mathcal{G}(\varphi, \varsigma)$, 111
$\prec_\varphi$, 66
$\to^\beta$, 145
$r(.)$, 27
$\sim$, 30, 31, 144, 146
$\sim_\lambda$, 29, 144
$\sim^{\mathrm{sym}}$, 77, 148
θ, 26
$\mathcal{V}ar$, 30
$\mathcal{V}ar(.)$, 31
$/$, 31
$=^{ex}_\mathcal{G}$, 102
$\mathcal{C}_\mathrm{d}$, 32
$\mathcal{C}_\mathrm{p}$, 32
$\mathcal{C}_\mathrm{pp}$, 79
$\mathcal{G}$-measure, 111
$\mathcal{G}$-normal form, 103
$\mathcal{P}$, 119, 131
$\mathcal{P}_\beta$, 161, 163
$\mathcal{P}_\mathrm{d}$, 59
$\mathcal{P}_\mathrm{gr}$, 149
$\mathcal{P}_\mathrm{new}$, 111
$\mathcal{P}_\mathrm{o}$, 111
$\mathcal{P}_\mathrm{p}$, 80
$\mathcal{UB}$, 171
ante, 29, 31
(ante.out), 30
(ante.same), 30
$\mathrm{appc}_.$, 58, 76
$\in \mathsf{b}(.)$, 79
$\in \mathsf{b}(\ldots)$, 149
$\mathsf{b}(.)$, 27, 185
beta_n, 162
$\mathrm{beta}_{X_2,n}$, 162
(β.ante.diff), 142
(β.ante.out), 142
(β.ante.same), 142
($\beta.\lambda$.diff), 142
($\beta.\lambda$.out), 142
($\beta.\lambda$.same), 142
$\in \mathsf{b}^-(\ldots)$, 149
$\in \mathsf{b}^-(.)$, 79
$\mathsf{b}^-(.)$, 27
$\notin \mathsf{b}^-(\ldots)$, 149
$\notin \mathsf{b}^-(.)$, 79
$\notin \mathsf{b}(\ldots)$, 149
$\notin \mathsf{b}(.)$, 79
$\mathsf{diff}(\varphi_1, \varphi_2)$, 111
$\perp$, 26, 31
$\triangleleft^*$, 26, 31
$=$, 31
(gp.ante.diff), 144
(gp.ante.out), 144
(gp.ante.same), 144
(gp.λ.diff), 144
(gp.λ.hang), 144

(gp.λ.out), 144
(gp.λ.same), 144
$\mathsf{i}(.)$, 27
$\neq$, 31
$:f$, 31
(λ.hang), 29
(λ.out), 29
(λ.same), 29
$\mathsf{lc}(\mathcal{S},\varphi)$, 111
$\leq_{\mathcal{G}}$, 105
φ-disjointness set, 67
φ, 31
π, 26
ς, 31
ψ, 26
$\triangleleft^+$, 31
reductlike$(C, B, A_1, \ldots, A_n)$, 146
redtree$_{X_2}(C, B, A)$, 146
$\mathsf{seg}(.)$, 79
σ, 31, 91

alpha segment, 181
alpha-gamma tree, 187
ambiguity, 7
anaphor, 38
anaphoric binding, 12, 38, 123
antecedent, 35
application condition, 58, 76
argument segment, 138

beta reduction formula, 146
beta reduction literal, 145
beta reduction relation, 142
Binding Theory, 199
body segment, 138

capturing, 42
clash-free, 59
clause, 58
CLLS, 10, 31
$\mathrm{CLLS_j}$, 191
$\mathrm{CLLS_{gr}}$, 148
$\mathrm{CLLS_p}$, 79
completeness, 69, 111, 130, 160
connectedness set, 66
constraint, 2
constraint graph, 12, 32
Constraint Language for Lambda Structures, 10, 31
constraint solver, 2
constraint system, 2
constructor tree, 3, 43
context, 5, 12, 46
context function, 5, 46
context segment, 138
context unification, 5, 46, 50, 114
contrasting element, 9
copy set, 96
correction, 9
correspondence, 16
correspondence formula, 17, 77
correspondence function, 17, 27
correspondence literal, 162
correspondence-generated, 92, 156
CU, 5, 46, 50, 114
current term, 170

D segment term, 139
deletion, 197
deterministic rule, 58
disjointness, 26
distribution rule, 58
dominance, 10, 26
dominance constraint, 15, 32
DSP, 51, 53, 192, 201, 203

ellipsis, 8, 13, 35, 197
extension by labeling, 66
external perspective on trees, 3, 43

failed, 59
fairness, 89, 124, 155, 169
feature description language, 4, 45
feature tree, 4, 43
finite failure, 116
fragment, 32

gamma segment, 181
generatedness, 91, 125, 156
group parallelism, 18, 144

group parallelism literal, 146

inequality, 26
inequality set for φ, 111
internal perspective on trees, 3, 43
inverse lambda binding, 146, 155

jigsaw parallelism, 19, 181
jigsaw parallelism literal, 191
jigsaw parallelism relation, 189
jigsaw segment, 181, 182, 185

labeled, 64
labeling, 11
lambda binding, 11, 122
lambda calculus, 7
lambda structure, 10, 28, 29
licensing, 198
linear redex, 165
linguistic structure, 9, 198
literal, 11
logical form, 199

many-pronoun puzzle, 40
minimal saturated constraint, 17, 69, 102
minimal saturation, 17, 69, 102
model, 31
Montague Grammar, 6

non-simple, 66, 95, 127
nonlinear redex, 165, 167, 171
nontermination, 155
normal dominance constraint, 15, 72, 209

padded constraint, 108
parallelism, 10, 27, 30
parallelism constraint, 32
parallelism phenomena, 8
path literal, 79
path parallelism, 77
path parallelism literal, 79
projected, 106
proper overlap, 182

quantifier, 8
quantifier parallelism, 9, 14, 37, 84, 200

reconstruction, 197
redex, 138
reducing tree, 138, 141
reduct, 138
reductlike, 141
remainder set, 181, 182, 185
root, 27
root variable, 64

satisfiable, 31
saturated constraint, 59
saturated constraint, 87, 90, 125, 156, 169
saturation, 17, 57, 59
saturation, 87, 90, 125, 156, 169
saturation procedure, 57
saturation rule, 58, 76
saturation step, 58, 76
scope ambiguity, 7, 12, 33
search tree, 115
second-order monadic logic, 44
segment, 10, 12, 27
segment term, 16, 31
self-overlap, 85, 208
set operator, 33
simple, 64, 93, 125, 156
simplification, 2
singleton segment, 182
SkS, 3, 44
sloppy, 39
solved form, 63
soundness, 62, 89, 124, 154, 168
source contrasting element, 35
source parallel element, 35
source sentence, 9, 35
split antecedent, 200
stratum, 199
strict, 39
strict dominance, 26
strict/sloppy ambiguity, 39, 123
string unification, 5, 47
subjacency, 199
surface form, 198
symmetric group parallelism literal, 162

target contrasting element, 35
target parallel element, 35
target sentence, 9, 35
termination, 62
tree domain, 26
tree structure, 26

underspecification, 7
underspecified beta reduction, 14, 138, 170
underspecified correspondence, 19
underspecified correspondence literal, 162

V-variant, 76
VP ellipsis, 9

WSkS, 3, 44