MONOGRAFIES DE L'INSTITUT D´INVESTIGACIÓ EN INTEL·LIGÈNCIA ARTIFICIAL

On Some Variants of Second-Order Unication

Mateu Villaret i Auselle

Consell Superior d´Investigacions Científiques

MONOGRAFIES DE L'INSTITUT D'INVESTIGACIÓ
EN INTEL·LIGÈNCIA ARTIFICIAL
Number 22

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

Monografies de l'Institut d'Investigació en Intel·ligència Artificial

Num. 1 J. Puyol, *MILORD II: A Language for Knowledge–Based Systems*

Num. 2 J. Levy, *The Calculus of Refinements, a Formal Specification Model Based on Inclusions*

Num. 3 Ll. Vila, *On Temporal Representation and Reasoning in Knowledge–Based Systems*

Num. 4 M. Domingo, *An Expert System Architecture for Identification in Biology*

Num. 5 E. Armengol, *A Framework for Integrating Learning and Problem Solving*

Num. 6 J. Ll. Arcos, *The Noos Representation Language*

Num. 7 J. Larrosa, *Algorithms and Heuristics for Total and Partial Constraint Satisfaction*

Num. 8 P. Noriega, *Agent Mediated Auctions: The Fishmarket Metaphor*

Num. 9 F. Manyà, *Proof Procedures for Multiple-Valued Propositional Logics*

Num. 10 W. M. Schorlemmer, *On Specifying and Reasoning with Special Relations*

Num. 11 M. López-Sánchez, *Approaches to Map Generation by means of Collaborative Autonomous Robots*

Num. 12 D. Robertson, *Pragmatics in the Synthesis of Logic Programs*

Num. 13 P. Faratin, *Automated Service Negotiation between Autonomous Computational Agents*

Num. 14 J. A. Rodríguez, *On the Design and Construction of Agent-mediated Electronis Institutions*

Num. 15 T. Alsinet, *Logic Programming with Fuzzy Unification and Imprecise Constants: Possibilistic Semantics and Automated Deduction*

Num. 16 A. Zapico, *On Axiomatic Foundations for Qualitative Decision Theory - A Possibilistic Approach*

Num. 17 A. Valls, *ClusDM: A multiple criteria decision method for heterogeneous data sets*

Num. 18 D. Busquets, *A Multiagent Approach to Qualitative Navigation in Robotics*

Num. 19 M. Esteva, *Electronic Institutions: from specification to development*

Num. 20 J. Sabater, *Trust and Reputation for Agent Societies*

Num. 21 J. Cerquides, *Improving Bayesian Classifiers*

Num. 22 M. Villaret, *On Some Variants of Second-Order Unification*

Num. 23 M. Gómez, *Open, Reusable and Configurable Multi-agent Systems: Aknowledge Modelling Approach*

Num. 24 S. Ramchurn, *Multi-Agent Negotiation Using Trust and Persuasion*

On Some Variants of Second-Order Unification

Mateu Villaret i Auselle

Foreword by Jordi Levy

2005 Consell Superior d'Investigacions Científiques
Institut d'Investigació en Intel·ligència Artificial
Bellaterra, Catalonia, Spain.

Series Editor
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Foreword by
Jordi Levy
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Volume Author
Mateu Villaret
Departament d'Informàtica i Matemàtica Aplicada
Universitat de Girona

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

NIPO: 653-05-061-X
ISBN: 84-00-08319-9
Dip. Legal: B-36528-2005

Ordering Information: Text orders should be addressed to the Library of the IIIA, Institut d'Investigació en Intel·ligència Artificial, Campus de la Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain.

per la Gemma i en Mateu petit
i pels meus pares

Contents

Foreword **xiii**

Acknowledgements **xv**

Resum **xvii**

Abstract **xix**

1 Introduction **1**
1.1 Main Distinct Unification Problems 1
1.1.1 First-Order Unification 1
1.1.2 E-Unification 3
1.1.3 Word Unification 4
1.1.4 Context Unification 5
1.1.5 Higher-Order and Second-Order Unification 6
1.2 Higher-Order Applications 7
1.2.1 Automated Theorem Proving 7
1.2.2 Higher-Order Logic Programming 8
1.2.3 Program Synthesis and Transformation 9
1.2.4 Natural Language Semantics 12
1.3 Plan of the Thesis 16

2 Second-Order Unification **19**
2.1 Simply Typed λ-calculus 19
2.1.1 Types 19
2.1.2 Terms 20
2.1.3 Substitutions 21
2.1.4 λ-equivalence 22
2.1.5 Unification 25
2.2 Second-Order Unification Undecidability 27
2.3 Second-Order Unification Procedure 29
2.3.1 Pre-Unification 32
2.3.2 Regular Search Trees 33
2.4 Decidable Subcases 34

2.4.1 First-Order Unification 34
2.4.2 Pattern Unification 34
2.4.3 Monadic Second-Order Unification 35
2.4.4 Second-Order Unification With Linear Occurrences of Second-Order Variables 36
2.5 Higher-Order Matching 36
2.6 Summary 37

3 Linear Second-Order and Context Unification 39
3.1 Linear Second-Order Unification 39
3.1.1 Sound and Complete Procedure 40
3.2 Context Unification 42
3.2.1 Context Unification From the First-Order Unification Perspective 43
3.2.2 Context Unification From the Second-Order Unification Perspective 44
3.2.3 Comparison Between both Perspectives 45
3.2.4 Historical Notes 48
3.3 Known Decidable Fragments of Context Unification 48
3.3.1 Word Unification 48
3.3.2 Stratified Context Unification 50
3.3.3 The Two Distinct Context Variables Fragment 51
3.4 Bounded Second-Order Unification 52
3.5 Linear Second-Order, Linear Higher-Order and Context Matching 53
3.6 About Linear Second-Order and Context Unification Decidability 54
3.7 Summary 54

4 Currying Second-Order Unification Problems 57
4.1 Introduction 57
4.2 Preliminary Definitions 60
4.3 Currying Terms 61
4.4 Labeling Terms 62
4.5 When Variables do not Touch 63
4.6 About Currying Higher-Order Matching 68
4.7 Summary 69

5 Context Unification and Traversal Equations 71
5.1 Introduction 71
5.1.1 A Naive Reduction 71
5.2 Preliminary Definitions 74
5.3 Terms and Traversal Sequences 75
5.4 Traversal Equations 78
5.4.1 Rank-bound Traversal Systems 79
5.4.2 Permutation and Rank-bound Traversal Systems 82
5.5 The Rank-Bound Conjecture 83
5.6 Reducing Context Unification to Traversal Equations 86

5.7 Some Hints in Favor of the Rank-Bound Conjecture 91
5.7.1 First Hint 93
5.7.2 Second Hint 96
5.8 Summary 98

6 From LSOU to CU with TR-Constraints 99
6.1 Introduction 99
6.2 Preliminaries 101
6.3 Reducing LSOU to CU with TR-Constraints 103
6.4 Translating TR-Constraints to R-Constraints over Traversal Sequences 109
6.5 About Decidability 114
6.6 Extending the Results to Higher-Order Unification 115
6.7 Summary 116

7 Describing Lambda-Terms in CU with TR-Constraints 119
7.1 Introduction 119
7.2 The Parallelism and Lambda Binding Constraints Language . . . 121
7.2.1 Tree Structures and Parallelism 121
7.2.2 Lambda Structures and Parallel Lambda Binding 123
7.2.3 Constraint Languages 125
7.3 Example 127
7.4 The Non-intervenance Property 129
7.5 Elimination of Lambda Binding Constraints 131
7.6 The Monadic Second-Order Dominance Logic and TR-Constraints 137
7.6.1 Tree-Regular Constraints 137
7.6.2 Monadic Second-Order Dominance Logic 138
7.6.3 Extending Node Labels 139
7.6.4 Constructing Tree Automata 141
7.7 Extensions of Parallelism Constraints 143
7.8 Equivalence Between PC and CU when Considering TR-Constraints 146
7.8.1 Main Result 154
7.9 Limitations 154
7.10 Summary 155

8 Conclusion 157
8.1 Summary of the Thesis 157
8.2 Future Work 160

Bibliography 161

Index 173

List of Figures

3.1 Solution of the n-ary variables equation $F(G(a,b)) \stackrel{?}{=} G(F(a),b)$. 46
3.2 Solution of the unary variables equation $F(G_0(g(G_1(a),G_2(b)))) \stackrel{?}{=} G_0(g(G_1(F(a)),G_2(b)))$. 47

4.1 Common instance of the curried context unification equation of Example 4.1. 59

5.1 Examples of trees with ranks equal to 0, 1, 2 and ∞. 73
5.2 Representations of the function $f(i) = \text{width}(a_1 \cdots a_i)$, for some traversal sequences of $f(a, f(b, f(c, d)))$. 77

7.1 Representation of part of a tree that satisfies the relations of children-labeling: $\pi_0 : f(\pi_1, \pi_2)$, dominance: $\pi_0 \triangleleft^* \pi_3$ and disjointness: $\pi_1 \perp \pi_2$. 122
7.2 The segment $\pi/\pi_1, \pi_2$. 122
7.3 Parallelism relation between π_1/π_2 and π_3/π_4. 123
7.4 The lambda structure of $\lambda x.\,(f\ x)$. 124
7.5 Representation of the axioms of parallel lambda binding. 124
7.6 Logical languages for tree and lambda structures. 126
7.7 The graph of the constraint language for lambda structures formula for the semantics of the sentence: *John saw a taxi and so did Bill*. 127
7.8 Intervenance. 130
7.9 Non-intervenance and lambda binding. 132
7.10 Translation Literals. Naming variable binder for correspondence classes e. Auxiliary predicates in Figure 7.11. 133
7.11 Auxiliary predicates. 134
7.12 The tree τ' containing τ and its corresponding tree with extended labels. 144
7.13 Reduction of Parallelism with Tree-Regular Constraints to Context Unification with Tree-Regular Constraints. 151
7.14 Group parallelism between $(X_1/X_2, X_4/X_5) \sim (X_2/X_3, X_3/X_4)$. 155

8.1 Studied problems and their relations. 158

Foreword

At the beginning of 90's surged a problem in theorem proving called "context unification" that caught the interest of part of the community. This is the guideline of this book. Context unification is a variant of second-order unification, but contrarily to it, most people conjecture its decidability. However, the question about this point remains open after more than ten years of intense research. A lot of work has been done in such direction, proving decidability for a wide variety of cases, some of them known to be undecidable in the case of general second-order unification. It is indubitable that this book contains some of the most decisive of these works.

Evidently, a positive answer to the decidability of second-order unification would have had a great impact in theorem proving, because it would open the possibility of automatizing part of the second-order logic. What makes second-order unification undecidable is also an interesting question. It is known that the absence of lambda-bindings do not improve the situation. Neither the restriction on the number of variables, or on their number of occurrences. The limitation on the number of times that a function uses its arguments, just the limitation that defines context unification, seems to be the key. In this book it is proved that the restriction on the number of second-order constants does not make any difference, which allows us to concentrate in the case of just one second-order function symbol. Then, it is proved that context unification is decidable if a certain conjecture about the rank of the solutions is true. What is call "rank" here, is nothing else than the Strahler number of a tree, a measure of trees defined by a geologist, or the register number. The conjecture is not proved in the book, but some interesting hints supporting it are given.

Finally, decidability of context unification would have a positive impact in not only theorem proving, but also in some other areas of computer science, such as computational linguistics. The last part of the thesis relates context unification with parallelism constraints and the constraint language for lambda structures that are used to represent semantic underspecification in linguistics.

Bellaterra, December 2004

Jordi Levy
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Acknowledgements

I have many people to thank. First of all, I want to express my gratitude to Jordi Levy, the supervisor of this dissertation, who introduced me to the world of computer science research. By contaminating me with the virus of Context Unification he gave me a marvellous research line, although not free of suffering. We have worked together in a great part of the thesis and he has carefully read all the material and has always been ready to advise me whenever I asked. I appreciate his patience and joy of teaching, thanks to which I have been able to do this work. He has also become a good friend of mine.

I am especially indebted to Joachim Niehren, with whom after long e-mail discussions, I have started a nice research line, not only in computer science, but also in wine tasting. Some of the results of the former are also a part of the work.

The discussions and talks with Katrin Erk, Philippe de Groote, Manfred Schmidt-Schauß, Klaus U. Schulz and Ganesh Sittampalam, among others, have also been very helpful and enriching.

I also wish to thank my many colleagues in the department of *Informàtica i Matemàtica Aplicada* for their support. My office partner Miquel Bofill has always been ready to help and listen to my problems, and of course we have shared lots of hard and glorious moments. I hope there will be more beers to share. The rest of the juniors Maria Fuentes, Roel Martínez, Marc Massot, Gustavo Patow and Pere-Pau Vàzquez, with whom I've shared the challenge of writing a thesis, and the seniors Francesc Castro, Miquel Feixas, Xavier Pueyo, Mateu Sbert, Joan Surrell, Josep Suy, Jaume Rigau have also encouraged me thanks to their confidence in me and support. The rest of colleagues of the department have been equally friendly and encouraging.

I would like to thank my colleagues in the *Institut d'Inteligència Artificial*, for considering me as another member of the *Institut*. In particular to Jaume Agustí, Pedro Messeguer, Martí Sànchez, Marco Schorlemmer and Dani, with whom I shared some of my worries and illusions. Also to the staff Francesc Esteva, Ramon López de Màntaras, and the rest of the *IIIAencs*.

My colleagues in the department of *Llenguatges i Sistemes Informàtics*, Maria Bonet, Guillem Godoy, Roberto Niewenhuis, Horacio Rodríguez, Albert Rubio and Gabriel Valiente among others, have really been my other research family in Barcelona.

I thank Gert Smolka for welcoming me at the Programming Systems Lab. in Saarbrücken. Andreas Framke, Tim Priesnitz, and many others, made that month in Germany unforgettable.

My thanks to my wife Gemma Abío, who helped me to improve my poor english style and gave me much-needed support and encouragement throughout the project.

My friends from the faculty, from the volley club and from the chorus, have given me moral support at all times.

Some other people have also helped me in other respects like my cousin Ventura Passols, Rosa Sagristà, José Luís Balcázar, and many others.

This work has been founded by the *Universitat de Girona*, by the *Universitat Politècnica de Catalunya* and by the the CICYT research projects DENOC (BFM 2000-1054-C02-01) and CADVIAL (TIC2001-2392-C03-01).

And last but not least, I would like to thank all my family, especially my parents, brothers and sisters for their patience and understanding. I also wish to thank my wife's family for all their love and support.

All these people have contributed to make this dissertation possible. Of course, all remaining errors are my own responsibility.

Girona, December 2004

Mateu Villaret
Departament d'Informàtica i Matemàtica Aplicada
Universitat de Girona
villaret@ima.udg.es

Resum

En aquesta tesi presentem diversos resultats sobre el Problema de la Unificació de Segon Ordre. És ben sabut que la Unificació de Segon Ordre és, en general, indecidible, tot i que la frontera entre la decidibilitat i la indecidibilitat que dibuixen les seves subclasses, és fina i no està totalment definida. El nostre objectiu és aportar alguna pista més sobre aquest problema i estudiar algunes de les seves variants. De fet, ens hem concentrat en el Problema de la Unificació de Contextes i en el Problema de la Unificació Lineal de Segon Ordre. Aquests dos problemes són variants de la Unificació de Segon Ordre on els unificadors han de ser termes lineals. La Unificació de Contextes fa més de deu anys que es va definir i la seva decidibilitat (així com la de la Unificació Lineal de Segon Ordre) encara és un problema obert. En aquest treball aportem resultats significatius que poden ajudar a solucionar el problema.

El primer resultat que presentem és una simplificació d'aquests problemes gracies al que anomenem "currificació" (Levy i Villaret, 2002). Concretament demostrem que la Unificació de Contextes es pot NP-reduir al Problema de la Unificació de Contextes on només es poden usar un símbol de funció binari i constants, a més de les variables evidentment. També demostrem un resultat similar per al Problema de la Unificació de Segon Ordre.

El resultat central de la tesi és la definició d'una condició no trivial en els unificadors que és necessària i suficient per a demostrar la decidibilitat de la Unificació de Contextes. Aquesta l'anomenem *conjectura del rank acotat* (Levy i Villaret, 2001). La conjectura es basa en una mesura no trivial dels termes, el *rank*, i postula que sempre que una instància del Problema de la Unificació de Contextes sigui satisfactible, existirà un unificador amb un rank acotat per una cota en funció de la grandària del problema. Assumint aquest postulat, reduïm el problema de la satisfactibilitat per a la Unificació de Contextes al problema de la satisfactibilitat per a la Unificació de Paraules, que és decidible.

Finalment, tal i com s'ha fet per a la Unificació de Paraules, estudiem l'extensió "natural" de la Unificació de Contextes mitjançant restriccions d'arbres regulars en la instanciació de les variables. D'aquest estudi en surten un parell de resultats més:

- primerament definim una interrelació entre el Problema de la Unificació Lineal de Segon Ordre i la Unificació de Contextes (Levy i Villaret, 2000). Concretament hem reduït la Unificació Lineal de Segon Ordre a la Unifi-

cació de Contextes amb restriccions d'arbres regulars, aquestes restriccions les usem per a evitar la captura de variables.

- Per últim, també hem definit una interrelació precisa entre el Problema de la Unificació de Contextes i el Llenguatge de Restriccions per a Lambda Estructures (Niehren i Villaret, 2002, 2003). Aquest llenguatge és usat abastament en el tractament de sentències ambigües en llenguatge natural, i actualment hi ha molt interès en saber quina és la potència d'aquest formalisme així com quina és la seva naturalesa computacional. El fet d'haver relacionat aquest llenguatge amb el món de la unificació pot ajudar-nos a aplicar els resultats teòrics d'un costat cap a l'altre.

Al principi de la tesi també fem una breu descripció sobre el Problema de la Unificació en general, així com introduïm les seves principals variants. Encara que aquesta tesi no estigui directament enfocada a aspectes aplicats, també assenyalem quins han estat i quin és el principal paper de la Unificació en la lògica computacional i en les seves aplicacions, centrant-nos sobretot en les aplicacions de la Unificació d'Ordre Superior.

Abstract

In this thesis we present several results about Second-Order Unification. It is well known that the Second-Order Unification Problem is in general undecidable; the frontier between its decidable and undecidable subclasses is thin and it still has not been completely defined. Our purpose is to shed some light on the Second-Order Unification problem and study some of its variants. We have mainly focused our attention on Context Unification and Linear Second-Order Unification. Roughly speaking, these problems are variants of Second-Order Unification where second-order variables are required to be linear. Context Unification was defined more than ten years ago and its decidability has been an open question since then. Here we make relevant contributions to the study of this question.

The first result that we present is a simplification on these problems thanks to "curryfication" (Levy and Villaret, 2002). We show that the Context Unification problem can be NP-reduced to the Context Unification problem where, apart from variables, just a single binary function symbol, and first-order constants, are used. We also show that a similar result also holds for Second-Order Unification.

The main result of this thesis is the definition of a non-trivial sufficient and necessary condition on the unifiers, for the decidability of Context Unification. The condition is called *rank-bound conjecture* (Levy and Villaret, 2001) in order to enforce our belief about its truthness. It lies on a non-trivial measure of terms, the *rank*, and claims that, whenever an instance of the Context Unification problem is satisfiable, there exists a unifier with a rank not exceeding a certain bound depending on the size of the problem. Under the assumption of this conjecture, we give a reduction of the satisfiability problem for Context Unification to the (decidable) satisfiability problem of Word Unification with regular constraints.

Finally, in the same spirit of the extension of Word Unification with regular constraints, we also study the natural extension of Context Unification by means of tree-regular constraints on variable instantiations. We contribute with two more results:

- firstly, we establish a relationship between Linear Second-Order Unification and Context Unification (Levy and Villaret, 2000). Mainly, we reduce Linear Second-Order Unification to Context Unification with tree-regular constraints, these constraints are used to avoid the capture of variables in

this process.

- Then, we also establish a relationship between Context Unification and the Constraint Language for Lambda Structures (Niehren and Villaret, 2002, 2003). This last formalism is broadly used in the treatment of ambiguous sentences of natural language, and there is currently an effort to quantify its power, and define its computational nature. Relating this constraints language with the unification framework can help us to apply the theoretic results from one side to the other.

We also give a brief description of unification and introduce its main distinct kinds and variants. Although our thesis is not directly oriented to practical issues, we also illustrate what has been, and what is the main role of unification in computational logics and applications, mainly focusing on Higher-Order Unification.

Chapter 1

Introduction

The *Unification Problem* has several variants and has been studied in a number of fields of computer science (Knight, 1989; Baader and Snyder, 2001), including theorem proving, logic programming, automatic program transformation, computational linguistics, etc.

Abstractly, unification means: *given two descriptions d_1 and d_2, can we find an object o that fits both descriptions?*

In a more mathematical sense, unification consists of solving equations, i.e., given a pair of "terms" (*equation*) with some possibly common "unknowns" (*variables*), the problem is to decide whether or not there exists a possible assignation (*substitution*) to these unknowns that makes both objects "equal" modulo some equality theory. If such substitution exists, it is called *unifier*. When we have unknowns in only one of the terms, we talk about *matching* instead of unification. When we are not just considering a set (conjunction) of equations, but we allow to have more complex formulae combining equations and involving in particular negation, the problem is called *Disunification* (Comon, 1991).

Depending on what are the terms, the unknowns and the equality notion that we are considering, we get distinct kinds of unification. For instance, solving arithmetic equations can be seen as solving unification where the objects are arithmetic expressions, the unknowns are the variables that will be instantiated by numbers and the equality relation is the relation defined by the field structure of numbers. Therefore, the 10th Hilbert's problem, can be formalised as an (undecidable) unification problem.

1.1 Main Distinct Unification Problems

1.1.1 First-Order Unification

In First-Order Unification, terms are first-order terms, the unknowns are first-order variables, i.e. variables that can be instantiated by first-order terms, and the notion of equality corresponds to the syntactic equality.

The First-Order Unification Problem can be stated as: *Given two first-order terms s and t, does there exist a substitution σ of terms for the variables in s and t such that $\sigma(s) = \sigma(t)$?*

Note that $\sigma(s)$ and $\sigma(t)$ denote the application of substitution σ to terms s and t respectively i.e., $\sigma(s)$ and $\sigma(t)$ denote terms s and t respectively where all variables have been replaced by their corresponding values in σ. In general we will denote the unknowns by capital letters, constant symbols by lower case ones and substitutions by greek letters.

As a simple example, consider the following equation:

$$f(f(X_3, X_2), X_1) \stackrel{?}{=} f(X_1, f(X_2, X_3))$$

A possible unifier, could consist of assigning $f(a, a)$ to X_1 and a to X_2 and X_3. This assignment can be represented by means of the *substitution* σ:

$$\sigma = [X_1 \mapsto f(a, a),\ X_2 \mapsto a,\ X_3 \mapsto a]$$

Now, as we can see, the application of σ to both sides of the equation gives us the same term:

$$\sigma(f(f(X_3, X_2), X_1)) \ = \ f(f(a, a), f(a, a)) \ = \ \sigma(f(X_1, f(X_2, X_3)))$$

First-Order Unification was firstly studied by Herbrand in 1931, although its main crucial role in automated deduction was not considered until the 60's when J. A. Robinson invented the simple and powerful inference rule named *resolution* (Robinson, 1965). First-Order Unification is the cornerstone of the rule discovered by Robinson. In some sense this was also the beginning of automated logic. Robinson also showed that the First-Order Unification problem is decidable and that whenever a solution, or *unifier* exists, there always exists what is called a *most general unifier*, i.e. a unifier from which all other unifiers can be generated. Even more, in First-Order Unification whenever this most general unifier exists, it is unique up to variable renaming. Robinson's algorithm was quite inefficient requiring exponential time and space in the worst case. A great deal of effort has gone into improving the efficiency of unification. Among several other results, there are the ones of Venturini-Zilli (1975) reducing the complexity of Robinson's algorithm to quadratic time, and of Martelli and Montanari (1976) when they proved that a linear time algorithm for unification exists.

It was soon realised that resolution is better behaved in the restricted context of Horn clauses, a subset of first-order logic for which SLD-resolution is complete. Colmenauer and Kowalski considered this class of clauses and defined the elegant programming language PROLOG, which spurred the whole new field of logic programming (Kowalski, 1974).

First-Order Unification is not just used in resolution, but for many other purposes: in type inferencing for polymorphic programming languages (Milner, 1978), in expert systems, or in the calculation of critical pairs in the Knuth/Bendix completion procedure (Knuth and Bendix, 1967). Even in another theorem proving method that does not use resolution, like the so called *matings*, unification is required (Andrews, 1981).

Since the early 60's there have been many attempts to generalise the basic paradigm of theorem proving, and these attempts have stimulated research into more general forms of unification. On the one hand there was the goal of adding equality into the theorem proving procedure, and on the other hand there was the goal of automate higher-order logic. Both goals propitiated the two main generalisations of First-Order Unification: *E-Unification* and *Higher-Order Unification*.

1.1.2 *E*-Unification

The relevance of equational reasoning, i.e., the replaceability of equals by equals, in ordinary mathematical reasoning, and the expressive power of first-order logic to define algebraic structures, naturally lead to the introduction of equality into resolution. In this framework, Robinson and Wos introduced a new deduction rule dedicated to equality, *paramodulation* (Robinson and Wos, 1969; Nieuwenhuis and Rubio, 2001). Due to efficiency reasons, the best option is to split the deduction mechanism considering just the non-equational part in the refutation mechanism and using *E-Unification*, that is, First-Order Unification modulo an equational theory, instead of simply First-Order Unification to deal with the equational reasoning during the unification steps (Plotkin, 1972).

In E-Unification the terms are again first-order terms, and the unknowns first-order variables, but now the notion of equality is not just *syntactic equality* but equality modulo a given equality theory E.

The E-Unification Problem can be stated as: *Given a (finite) set E of equalities and two first-order terms s and t, does there exist a substitution σ of terms for the variables in s and t such that $\sigma(s)$ and $\sigma(t)$ are provably equal from the equations in E?*

As an example, consider the equation:

$$f(f(X_1, X_2), c) \stackrel{?}{=} f(X_1, f(X_2, c))$$

and the equational theory:

$$E = \{\ f(x, f(y, z)) = f(f(x, y), z)\ \}$$

a possible solution[1] is the substitution:

$$\sigma = [X_1 \mapsto a,\ X_2 \mapsto b]$$

$$\sigma(f(f(X_1, X_2), c)) \ =\ f(f(a, b), c) \ =_E f(a, f(b, c)) \ =\ \sigma(f(X_1, f(X_2, c)))$$

where here $=_E$ means the equality modulo the theory E.

Unlike First-Order Unification, E-Unification is undecidable in general. For instance, due to the undecidability of the word problem for semi-groups, E-Unification is undecidable when considering the semi-groups theory. Another

[1] Notice that if we were just considering first-order unification, this equation would have no solution.

major difference with respect to First-Order Unification, is that if an equation is solvable then there may not be a single most general unifier.

Nevertheless there are some theories known to be decidable, for instance $\emptyset$-Unification (i.e. First-Order Unification), AC-Unification (i.e. *Associative-Commutative* Unification), D-Unification (i.e. *Distributive* Unification), and A-Unification (i.e. *Associative* Unification). For a good summarisation of E-Unification see (Siekmann and Szabó, 1984) and (Baader and Siekmann, 1993).

A-unification is of special interest to us because it corresponds to the well known *Word Unification Problem*, shown to be decidable by Makanin (1977).

1.1.3 Word Unification

Word (String) Unification is unification where the terms are words, the unknowns can be instantiated by words and equality means to be the same word.

The Word Unification Problem can be stated as: *Given two words w_1 and w_2, does there exist a substitution σ of words for the variables in w_1 and w_2 such that $\sigma(w_1)$ and $\sigma(w_2)$ are the same word?*

Word Unification can also be seen as A-Unification where we use an associative symbol to form the words. Word Unification is decidable, but solvable Word Unification Problem instances do not always have just one most general unifier but possibly infinitely many independent unifiers, as illustrated by the following example:

$$aX \stackrel{?}{=} Xa$$

for which for all $n \geq 0$, any substitution of the form:

$$[X \mapsto \overbrace{a \ldots a}^{n}]$$

is a unifier, not comparable to any other.

The race for the decidability proof of Word Unification was long and full of little steps: firstly it was shown that the case when no variable occurs more than twice is decidable, then the three occurrences fragment was also proved decidable, and finally, thanks to the *exponent of periodicity* lemma, the general case was proved decidable by Makanin (1977). Since then, a lot of work has been made looking for lower upper bounds on the exponent of periodicity and trying to get the precise complexity of the problem (Kościelski and Pacholski, 1995, 1996). Two recent and independent works, due to Plandowski (1999a,b) and Gutiérrez (1998, 2000), show, with an alternative to Makanin's proof, that Word Unification is in the NEXP class and in the PSPACE class. These are the best complexity classes known for Word Unification. Nevertheless, some people believe that Word Unification is in NP.

It has also been proved that word equations, where variables can be constrained to belong to regular languages, is also decidable (Schulz, 1991). Several attempts to simplify Makanin's proof and trying to give a practical implementation of the algorithm, have been made (Jaffar, 1990; Schulz, 1993). Expressiveness of word equations has also been subject of study in (Karhumäki *et al.*, 1997).

Word Unification has applications, for instance, in deduction systems (Huet, 1976) and in constraint logic programming (Colmerauer, 1988).

Closely related to Word Unification there is the *Context Unification problem*. In fact, Word Unification can be seen also as "*Monadic Context Unification*", i.e. Context Unification considering just a monadic signature.

1.1.4 Context Unification

In Context Unification the terms are first-order terms and the unknowns are first-order and context variables. Substitutions assign first-order terms to first-order variables and contexts to context variables. *Contexts* are terms with "holes", i.e. terms with a special constant symbol called the *hole*, that is denoted by '•'. When a context is "applied" to some terms (arguments), the result is the term formed by the context where the holes have been replaced by the argument terms.

The Context Unification Problem can be stated as: *Given two terms s and t, does there exist a substitution σ of first-order terms and contexts for the variables in s and t such that $\sigma(s)$ and $\sigma(t)$ are the same term?*

As an example consider:

$$f(F(a), b) \stackrel{?}{=} F(f(a, b))$$

where F is a context variable. One of its solutions is the substitution:

$$\sigma = [F \mapsto \bullet]$$

which, when applied to the equation, identifies both sides:

$$\sigma(f(F(a), b)) = f(a, b) = \sigma(F(f(a, b)))$$

But as we can easily observe, as in the previous Word Unification example, there are infinitely many incomparable solutions for this equation, all of them having this form:

$$F \rightarrow \overbrace{f(\ldots f(}^{n} \bullet \overbrace{, b) \ldots , b)}^{n}$$

Therefore, solvable Context Unification equations can have infinitely many most general unifiers.

The Context Unification problem was firstly defined by Comon (1992a) and its decidability still remains unsolved. Context Unification has applications in rewriting (Comon, 1992a,b, 1998; Niehren *et al.*, 1997a, 2000), in unification theory (Schmidt-Schauß, 1996, 1998), and in computational linguistics (Pinkal, 1995; Niehren *et al.*, 1997b; Egg *et al.*, 1998; Niehren and Villaret, 2002, 2003).

Context Unification can also be seen as a variant of *Higher-Order Unification*, in fact it is closely related to Linear Second-Order Unification (Levy, 1996; Levy and Villaret, 2000, 2001), a variant of *Second-Order Unification*.

1.1.5 Higher-Order and Second-Order Unification

Higher-Order Unification serves for solving equations in the Simple Typed λ-Calculus (Church, 1940). In this unification problem, the terms considered are simply typed λ-terms, the unknowns are higher-order variables, i.e. variables that can be instantiated by simply typed λ-terms of the same type, and the equality relation is the congruence defined by the α, β and η congruencies of the λ-calculus.

The Higher-Order Unification Problem can be stated as: *Given two simply typed λ-terms s and t, does there exist a substitution σ of λ-terms for the variables in s and t such that $\sigma(s)$ and $\sigma(t)$ are λ-equivalent?*

For example, let s and t be:

$$F(\lambda y.\ y,\ b) \stackrel{?}{=} G(a)$$

one of its infinitely many solutions is:

$$\sigma = [F \mapsto \lambda x_1 x_2.\ x_1 x_2,\ G \mapsto \lambda x.\ b]$$

$$\sigma(s) = (\lambda x_1 x_2.\ x_1 x_2)(\lambda y.\ y,\ b) =_\beta (\lambda y.\ y) b =_\beta b\ _\beta = (\lambda x. b) a = \sigma(t)$$

where $=_\beta$ and $\ _\beta =$ denote β-equivalence in λ-calculus.

Like E-unification, Higher-Order Unification is undecidable in general and most general unifiers may not exist.

Second-order variables are variables that stand for functions on individuals. When variables are at most second-order, we talk about *Second-Order Unification*. Second-Order Unification is also undecidable (Goldfarb, 1981). There exists some recent work based on the number of distinct variables and the number of occurrences per variable, that draws a frontier between decidable and undecidable subclasses of Second-Order Unification (Levy, 1998; Levy and Veanes, 1998, 2000). Also the signature has been considered in the decidability question (Farmer, 1988, 1991).

There is a variant of Second-Order Unification problem called *Linear Second-Order Unification* (Levy, 1996). In this variant, it is imposed a limitation on the possible instances of variables: they are just allowed to be instantiated by linear terms, i.e. terms in normal form where each bound variable occurs in the body of the term once and only once. As we will show, this problem is closely related to Context Unification (Levy and Villaret, 2000). Although its decidability is still an open question, the fact that some subclasses that are undecidable in Second-Order Unification have been shown decidable in Context Unification (Levy, 1996; Schmidt-Schauß and Schulz, 1999), supports the common belief that Context Unification is decidable.

When variables are allowed to be instantiated by terms where bound variables can occur in the body of the term a bounded number of times, we talk about *Bounded Second-Order Unification*. Bounded Second-Order Unification is defined and shown decidable by Schmidt-Schauß (1999a, 2004). Adding the constraint that the number of lambdas in the unifiers is also bounded, and considering no just second-order variables, but variables of any order, the *Bounded*

Higher-Order Unification Problem is obtained. This last problem is also decidable (Schmidt-Schauß and Schulz, 2002a).

1.2 Higher-Order Applications

Despite of its undecidability, Higher-Order Unification is useful and necessary in many fields of computer science.

1.2.1 Automated Theorem Proving

Higher-Order Unification is required when automating higher-order logic. In this logic, quantification over sets or predicates and functions is allowed. This feature permits us, for instance, to axiomatise Peano arithmetic, which cannot be axiomatised just using first-order logic. But this increase on the expressive power is not for free. One of the major objections to the use of this logic comes from the Gödel first incompleteness theorem that states that no system that can formalise Peano arithmetic admits a complete deduction system. Nevertheless, Henkin generalised the notion of model theory with the so-called *general models* and proved that within this model theory, appropriate generalisations of first-order calculi to higher-order logics exist and are sound and complete. Since then, a wide range of methods for higher-order automated theorem proving has been proposed (Robinson, 1969; Darlington, 1971; Pietrzykowski, 1973; Huet, 1973a; Jensen and Pietrzykowski, 1976; Miller and Nadathur, 1987; Felty *et al.*, 1990; Paulson, 1990; Miller, 1991a; Paulson, 1993; Benzmüller and Kohlhase, 1998a,b). A textual cite from (Jensen and Pietrzykowski, 1976) illustrates the interest in higher-order logic despite its difficulty: *"...The attractiveness of higher-order methods in computational logic is not what you can or cannot prove, but rather that many proofs are more natural in a higher-order setting."*.

The first successful attempts to mechanise and implement higher-order logic were those of Pietrzykowski (1973); Jensen and Pietrzykowski (1976) and of Huet (1973a). They combined the resolution principle with Higher-Order Unification. As we have already said, Higher-Order Unification is undecidable, in fact semidecidable, and researchers were looking for procedures that were capable of completely enumerate all set of unifiers (notice that there can be infinitely many). The first implementation of a procedure for Higher-Order Unification already revealed that the search space for unifiers is far too large to be feasible for practical applications. Huet made a major contribution in showing that a restricted form of unification (also undecidable), called *preunification*, is sufficient for most refutation methods, and in defining a method for solving this restricted problem which is used by most current higher-order systems (Huet, 1975, 1976).

Nevertheless, since Higher-Order Unification is undecidable and when solutions exist there can be infinitely many, incorporating unification into the resolution inference rule would not result in an effectively computable rule. As a remedy, the unification process can be delayed by capturing the unification

equations as constraints and effectively interleaving the search for empty clauses by resolution with the search of unifiers.

But Higher-Order Unification is not only used in automated theorem proving but also in *higher-order logic programming*, in *program synthesis* and *program transformation* and in *computational linguistics* among other areas. We will illustrate now some of these applications.

1.2.2 Higher-Order Logic Programming

As in Prolog, unification plays a crucial role in higher-order logic programming. But now, the variables considered are not just first order but higher-order ones, therefore we can write programs which are parameterised not just by values but by functions. This feature is not just a privilege of higher-order logic programming, but of higher-order programming in general like functional programming.

According to the use of variables there are distinct approaches, on the one hand there is the logical framework *Isabelle* (Paulson, 1990, 1993) that just allow higher-order variables as functions but not as predicates, and on the other hand there is *λProlog* (Felty *et al.*, 1990; Miller, 1991a,b; Müller and Niehren, 1998) that allows both uses of variables.

One simple example to illustrate the higher-order features is the definition of a mapping function: a function that takes a function and a list as arguments and produces a new list by applying the given function to each element of the former list. We show this example from the perspective of quantifying over predicates, in a relational style.

We write the predicates in a higher-order logic program style, *a la* Prolog, as follows:

```
mappred(P, [], []).
mappred(P, [X|L], [Y|K]) :- P(X, Y), mappred(P, L, K).

age(mateu, 31).
age(gemma, 29).
```

We have also added two facts that define the predicate `age` over two elements. Using this program, now we could get the list of ages of `mateu` and `gemma` with the query:

```
?- mappred(age, [mateu, gemma], L).
```

the answer of which would be the substitution `[31, 29]` for `L`. Tracing a successful solution path for this query we can observe that Higher-Order Unification has been required; for instance, to shoot the first rule, `P` has been matched with:

```
\x y. age(x, y)
```

As we can also notice, when predicate variables get instantiated and after being supplied with appropriate arguments, they become new queries. In fact

the first new goal to solve becomes `age(mateu, X')`. Therefore, one needs to be careful because Prolog only considers Horn clauses, therefore the predicate variables when instantiated need to be correct *Horn goals*. Accordingly with this fact, we can only use conjunctions, disjunctions and existential quantifications to instantiate predicate variables that can become queries.

In (Miller *et al.*, 1991), it is proposed the use of *Hereditary Harrop Formulae*, a generalisation on the formulae considered to overcome these Horn clauses limitations.

The previous example illustrates how predicate variables can be used and the power increase that they provide. But not everything one could expect to obtain can be achieved. Consider the following query:

```
?- mappred( R, [mateu, gemma], [31, 29]).
```

that could be used to find out what is the relation that exists between the two lists `[mateu, gemma]` and `[31, 29]`. One could expect the answer to be:

```
R -> \x y. age(x, y)
```

but this is too much optimistic. In fact, there are infinitely many relations that satisfy this query, and enumerating these does not seem to be a meaningful computational task. The problem can be stated in the intensional/extensional role of predicate variables. A broad discussion about these problems can be found in (Nadathur and Miller, 1998).

In the next subsection we will show how higher-order logic programming has nice properties to perform program transformations. We will also see that there are some other techniques that use Higher-Order Unification and Matching for program synthesis and program transformation.

1.2.3 Program Synthesis and Transformation

Automatic program synthesis consists of generating programs from specifications in an automatic manner. One of the pioneers of these techniques was Darlington (1973). His technique consisted of generating SNOBOL programs given a set of axioms based on those of Hoare, and employing a resolution based theorem prover incorporating a restricted Higher-Order Unification algorithm.

Program transformation is the process of converting a piece of code from one form to another whilst preserving its essential meaning. We will show a couple of perspectives to this field.

For instance, there is the work of Miller and Nadathur in λ-Prolog. Considering the higher-order facilities that λ-Prolog provides, basically its Higher-Order Unification features, we can see that it is possible to give rules that apply to certain patterns only matchable using Higher-Order Unification, and that allows us to re-build programs. One of such pattern examples occur in tail-recursive programs. From this recognition we can translate such programs into equivalent imperative programs.

Consider for instance, the following tail-recursive program that sums two non-negative integers, written in a pseudo λ-calculus style, and using `fixpt` as a recursive combinator:

$$\texttt{fixpt}\ \lambda sum.\lambda n.\lambda m.\ \texttt{if (n=0) then}\ m\ \ \texttt{else (}sum\ \texttt{(}n\texttt{-1) (}m\texttt{+1))}$$

The tail-recursiveness of this program can be easily recognised by using Higher-Order Unification (Matching). The program is in fact, an instance of the term:

$$\texttt{fixpt}\ \lambda f.\lambda x.\lambda y.\ \texttt{if (C}\ x\ y\texttt{) then (H}\ x\ y\texttt{) else (}f\ \texttt{(F1}\ x\ y\texttt{) (F2}\ x\ y\texttt{))}$$

as substitution σ shows:

$$\left[\begin{array}{lcl} \texttt{C} & \mapsto & \lambda z_1 z_2.\ \texttt{(}z_1\ \texttt{= 0)} \\ \texttt{H} & \mapsto & \lambda z_1 z_2.\ z_2 \\ \texttt{F1} & \mapsto & \lambda z_1 z_2.\ \texttt{(}z_1\ \texttt{- 1)} \\ \texttt{F2} & \mapsto & \lambda z_1 z_2.\ \texttt{(}z_2\ \texttt{+ 1)} \end{array}\right]$$

In fact, any closed term that unifies with this last "second-order template" must be a representation of a recursive program of two arguments whose body is a conditional and in which the only recursive call appears as the head of the expression that constitutes the "*else*" branch of the conditional. Clearly any functional program that has such a structure must be tail-recursive.

Now we should use these matched parts to form the corresponding imperative version of the program that will return the result in variable `result`. We use an imperative style *a la* PASCAL:

```
done := false
while (not done) do
      if (C par1 par2) then
         begin
           done := true;
           result := (H par1 par2)
         end
      else
         begin
           par1 := (F1 par1 par2);
           par2 := (F2 par1 par2)
         end
```

Now, if we apply substitution σ to this imperative program template term, we obtain the imperative version of the summing program:

```
done:=false
while (not done) do
      if (par1 = 0) then
         begin
```

```
        done := true;
        result := par2
      end
  else
      begin
        par1 := par1 - 1;
        par2 := par2 + 1
      end
```

Notice also that this template does not recognise all tail-recursive programs but just the ones that have two parameters and just one conditional in their body. A deeper study and discussion of how to solve this problem, and some other nice examples about program transformation can be found in (Miller and Nadathur, 1987).

There is a more recent work due to de Moor and Sittampalam (2001); Sittampalam and de Moor (2001). As they notice themselves, the automatic program transformation field has its major impact in easing the tension between program efficiency and program abstraction. The purpose of these program transformers is to translate an abstract and readable human-written code into an efficient one. But in general this task cannot be fully automatised and some human annotations are required.

These annotations are made by means of conditional higher-order rewriting rules that lead the transformed program to the transformation intended by the programmer. These rules require Higher-Order Matching.

The work of de Moor and Sittampalam on the development of the system MAG for HASKELL program transformation provides some nice examples that illustrate the power of Higher-Order Matching. One of their examples is the `reverse` list function expressed by means of a `fold`:

$$\texttt{reverse = foldr } (\lambda x.\lambda xs.\ xs \texttt{ ++ } [x]) \texttt{ []}$$

In this definition, we first apply to each element of the list the "switching side" function and then the list concatenation function, therefore this `reverse` definition has quadratic time complexity. Our goal is to transform it into a linear time program using the so called "fold fusion" law:

$$\begin{gathered}\text{if} \\ \lambda x.\lambda y.\ (O_2)\ x\ (Fy) \ = \lambda x.\lambda y.\ F(O_1\ x\ y) \\ \text{then} \\ F\ \texttt{(foldr } (O_1)\ E\ \texttt{xs) = foldr } (O_2)\ (FE)\ \texttt{xs}\end{gathered}$$

The application of this law to our definition requires Higher-Order Matching and provides us with this substitution for the new fold operator:

$$[O_2 \mapsto \lambda x.\lambda g.\lambda xs.\ g\texttt{(}x\texttt{:}xs\texttt{)}]$$

Then, applying the resulting substitution[2] we obtain a linear time version of the `reverse` function:

$$\texttt{reverse l = foldr } (\lambda x.\lambda g.\lambda xs.\ g(x{:}xs))\ \texttt{((++) []) l []}$$

1.2.4 Natural Language Semantics

Now we will illustrate the Higher-Order Unification applications in computational linguistics, like scope ambiguity (Pinkal, 1995; Niehren and Koller, 2001). We will dedicate a particular attention to the field of characterising the interpretative possibilities generated by elliptical constructions in natural language. In contrast to the previously presented applications of Higher-Order Unification, a part of our work, mainly Chapter 7 which is based on (Niehren and Villaret, 2002, 2003), is closely related to these natural language semantics topic. Hence, we will introduce this field in more detail than the previous ones.

In computational semantics, the formal description of the meaning of an expression often requires the use of sets and higher-order notions. The task of representing and reasoning about meaning in a computational setting was dealt, for instance, by Montague (1988), or by Miller and Nadathur (1986) who showed how it is possible to integrate syntactic and semantic analysis with λ-Prolog.

We now illustrate scope ambiguity and ellipsis, the two main linguistic phenomena where Context Unification has been used. Then we will introduce the approach that we will study and relate with Context Unification, in the last part of the thesis.

Scope Ambiguity

Scope Ambiguity consists of having more than one possibility for determining the scope of some elements (usually quantifiers) of the sentence. One of the examples in (Pinkal, 1995) is this scope ambiguity example where Linear Second-Order Unification is used, hence substitutions of Second-Order variables are required to be linear. The following sentence:

`Every researcher visited a company`

which is represented by the following equation:

$$\begin{gathered} C_1(@(every\_researcher, \lambda_{x_1}.(C_3(@(visit, @(x_1, x_2)))))) \\ \stackrel{?}{=} \\ C_2(@(a\_company, \lambda_{x_2}.(C_4(@(visit, @(x_1, x_2))))))\end{gathered}$$

has the following two possible solutions corresponding to the two possible readings:

[2]Notice that in fact O_2 is a third-order term because its second abstraction is a bound variable of order two, i.e. a function.

1. *every researcher visited a company which is not necessarily the same as the one that the others researchers visited.* To obtain this reading we consider the following substitution:

$$\left[\begin{array}{lcl} C_1 & \mapsto & \lambda x.\ x \\ C_2 & \mapsto & \lambda x.\ @(every\_researcher, \lambda_{x_1}.(x)) \\ C_3 & \mapsto & \lambda x.\ @(a\_company, \lambda_{x_2}.(x)) \\ C_4 & \mapsto & \lambda x.\ x \end{array}\right]$$

   which, applied to the equation gives us the following term:

$$@(every\_researcher, \lambda_{x_1}.(@(a\_company, \lambda_{x_2}.(visit, @(x_1, x_2)))))$$

2. or *there exists a company (the same for all) that is visited by every researcher.* To obtain this reading we consider the following substitution:

$$\left[\begin{array}{lcl} C_1 & \mapsto & \lambda x.\ @(a\_company, \lambda_{x_2}.(x)) \\ C_2 & \mapsto & \lambda x.\ x \\ C_3 & \mapsto & \lambda x.\ x \\ C_4 & \mapsto & \lambda x.\ @(every\_researcher, \lambda_{x_1}.(x)) \end{array}\right]$$

   which, applied to the equation, gives us the following term:

$$@(a\_company, \lambda_{x_2}.(@(every\_researcher, \lambda_{x_1}.(visit, @(x_1, x_2)))))$$

As we can appreciate, all substitutions are linear[3].

The work of Pinkal has been extended by Niehren *et al.* (1997b) where it is shown how Context Unification also deals with Ellipses.

Ellipses

Ellipses consist of omitting from a sentence, words needed to complete the construction or sense. In (Dalrymple *et al.*, 1991) it is shown how Higher-Order Unification correctly predicts a wide range of interactions between ellipsis and other semantic phenomena such as quantifier scope and bound anaphora. As a particular example, we can reproduce the verb phrase ellipsis phenomenon example from (Dalrymple *et al.*, 1991):

`Dan likes golf, and George does too.` (1.1)

The intended meaning of the sentence is that that Dan and George both like golf: $like(dan, golf) \wedge like(george, golf)$. The *source* clause, "`Dan likes golf`", parallels the target "`George does too`", with the subjects "`Dan`" and "`George`" being parallel elements, and the verb phrase of the target sentence being represented by the target phrase "`does too`".

[3]Notice that there is a distinction between the lambdas and the bound variables of the object language like λ_{x_1} and x_1, and the lambdas and bound variables of the substitutions, which disappear when applied to the term.

Now, we know that the property, let's say P, being predicated of George in the second sentence is such that when it is predicated on Dan, it means that Dan likes golf. We might state this by means of a Higher-Order Unification equation as follows:

$$P(dan) \stackrel{?}{=} like(dan, golf) \tag{1.2}$$

where P is a predicate variable. A possible value for P in this equation is the property represented by the λ-term $\lambda x.\ like(x, golf)$. Applying this predicate to George, we obtain $like(george, golf)$, and the full sentence meaning becomes the intended one:

$$like(dan, golf) \wedge like(george, golf)$$

Nevertheless not all the solutions of equation 1.2 have a meaningful counterpart in the linguistic semantic world. Consider now the substitution of P by $\lambda x.\ like(dan, golf)$. This is also a solution for the equation but when applied to George we obtain $like(dan, golf)$ and the following meaning for our elliptical sentence:

$$like(dan, golf) \wedge like(dan, golf)$$

which is not an interesting semantic interpretation, in fact it is wrong because it is not the intended meaning of sentence 1.1.

The way to solve this problem is to forbid some kind of substitutions, basically those that instantiate variables by terms that contain *primary occurrences* of the parallel elements (Dalrymple *et al.*, 1991). In this example then, the proposed second substitution is not a valid substitution because it contains a primary occurrence: *dan*. The technique of Dalrymple *et al.* (1991), computes reasonably enough solutions in comparison with other systems. But this way of filtering substitutions was not fully satisfactory. The goal was not to filter among a huge set of generated solutions, but rather to filter beforehand those solutions which are correct from those which are not.

There have been several researchers who have approached this problem, for instance Gardent and Kohlhase (1996), who deal with the primary occurrence constraint, or the one that we have shown in the scope ambiguity example of Pinkal (1995), using Linear Higher-Order Unification.

The Underspecified Semantic Representation Approach of The Constraint Language for Lambda Structures

The use of Context Unification has been broadly studied and related with other formalisms like *Dominance Constraints* (Koller *et al.*, 1998) and *Parallelism Constraints* (Erk and Niehren, 2000), in the works of Egg *et al.* (1998, 2001); Erk *et al.* (2002). Although Parallelism Constraints are equivalent to Context Unification, the procedures used to solve these constraints have a nicer behaviour than the ones for solving Context Unification (Koller, 1998; Erk and Niehren, 2000; Erk *et al.*, 2002), for instance the implementation of an incomplete Context Unification procedure in (Koller, 1998), runs into combinatoric explosion when dealing with scope ambiguities, and it does not perform well enough on the Context Unification equivalent of Dominance Constraints.

Parallelism Constraints extended by means of lambda-binding constraints and anaphora bindings, forms the *Constraint Language for Lambda Structures* (Egg *et al.*, 2001; Erk *et al.*, 2002). This constraint language is currently an active framework for underspecified semantics. The idea of semantic underspecification is to postpone the enumeration of meanings of a semantically ambiguous sentence. Instead, one represents the set of all meanings by need.

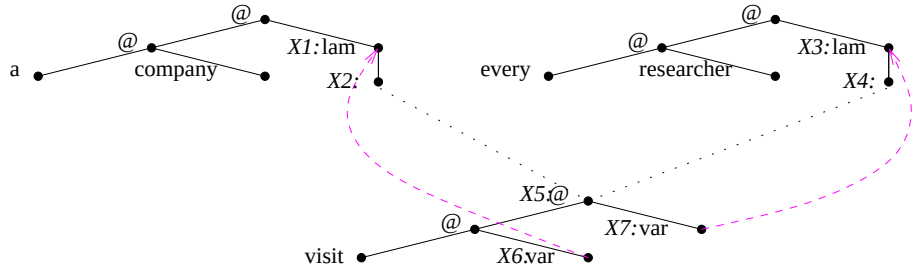
The last part of the thesis is devoted to establish a precise relationship between Context Unification and this Constraint Language for Lambda Structures language, therefore we will briefly describe it here.

The Constraint Language for Lambda Structure is a language for defining lambda-structures, i.e. terms with a special notion of λ-binding and of *anaphoric* binding. This language has also been extended by means of beta-reductions and group parallelism constraints. In the Constraint Language for Lambda Structures, the variables denote nodes of the tree, and the structure of this tree is described by stating the relations between its nodes and segments. The relation between nodes are stated by means of the literals: *labeling* to indicate what is the label of a node and what are its "mother-children" relations, *dominance* that establishes dominance between two nodes by stating that one is above the other, *lambda*-binding to indicate that a `var`-node is bound by a `lam`-node and *anaphoric*-binding to indicate that a node is an anaphora of another one. Segments are like contexts, and one can use the *parallelism* literal to indicate that two segments are parallel, i.e. that they have a similar[4] structure.

Consider again the following sentence:

`Every researcher visited a company`

Its Constraint Language for Lambda Structures description consists of: the labeling literals $X1$:`lam`$(X2)$, $X3$:`lam`$(X4)$, $X6$:`var`, $X7$:`var`, ... the dominance literals $X2 \triangleleft^* X5$, $X4 \triangleleft^* X5$ and the λ-binding literals $\lambda(X6) = X1$ and $\lambda(X7) = X3$. Its CLLS graphic representation counterpart is:



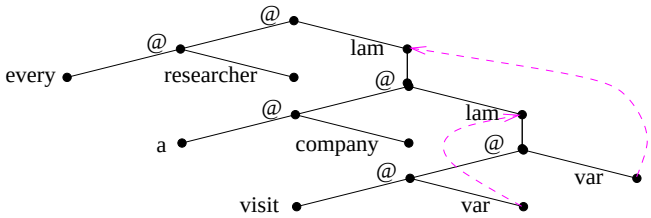
The nodes of the graph correspond to variables denoting nodes of a λ-structure, whereas labels, edges and arrows correspond to labeling, mother-children, dominance and λ-binding atomic constraints. Dotted edges signify *dominance*, where the upper node is required to be above the lower one in any λ-structure that satisfies the description. The dashed arrows, for λ-bindings, act like elastic bands, which can be stretched without breaking.

[4]The precise meaning of parallel will be defined more carefully later.

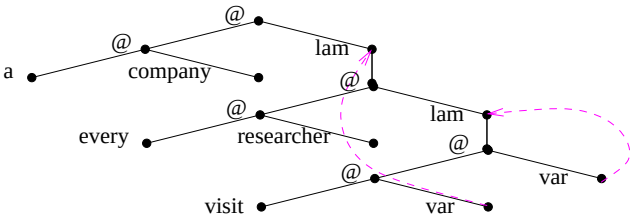
This underspecified description captures the two scope readings of the sentence by leaving the relative ordering between the two quantifier fragments (contiguous pieces of the graph that describe the `a` and the `every`) unspecified. But since both fragments dominate the same fragment, one must dominate the other. Such a situation is very common in scope underspecification.

Solving these constraints means finding a λ-structure which satisfies all the literals. In our example there are two "minimal" (in the sense that they do not introduce new non-strictly necessary nodes) solutions that correspond to, the already presented, two possible readings:

- Every researcher visited a company which is not necessarily the same as the one that the others researchers visited.



- There exists a company (the same for all) that is visited by every researcher.



There are currently, several solvers for Constraint Language for Lambda Structures formulae implemented in The Saarbrücken Programming Systems Lab. We will come back to Constraint Language for Lambda Structures later in our thesis to show how the stated relation between Context Unification and Parallelism Constraints can be extended to Constraint Language for Lambda Structures and Context Unification with regular constraints.

1.3 Plan of the Thesis

The thesis is organised as follows: Chapter 2 and Chapter 3 contain some preliminary definitions and the introduction of Higher and Second-Order Unification as well as the Linear Second-Order and Context Unification problems. It is also summarised the state of the art of these areas of research. Chapter 4 describes

a simplification on the Second-Order and the Context Unification problems by means of curryfication. This simplification serves for proving that decidability of Second-Order, and Context Unification, can be reduced to decidability of Second-Order, and Context Unification respectively, with just one binary function symbol (the application), and constants. In Chapter 5 we define the rank-bound property and prove that it is a sufficient and necessary condition for Context Unification decidability. In Chapter 6 and Chapter 7 we consider the natural extension of Context Unification by means of tree-regular constraints: in Chapter 6 we establish a precise relationship between Linear Second-Order Unification and Context Unification using tree-regular constraints to deal with λ-abstractions and bound variables, while in Chapter 7 we study and define the specific relationship between the Constraint Language for Lambda-structures and Context Unification, using also the tree-regular constraints to deal with the λ-bindings of the λ-structures, but in a different manner than in the Linear Second-Order Unification reduction. Finally, in Chapter 8 we conclude and we present the main lines of our future work.

Chapter 2

Second-Order Unification

In this chapter we introduce Second (and Higher)-Order Unification. We first introduce the base language for these problems: the simply typed λ-calculus, then we define Second (and Higher)-Order Unification, and sketch the undecidability proof of Second-Order Unification by Goldfarb (1981). We also present a sound and complete rule-based procedure for Second-Order Unification based on the one of Gallier and Snyder (1990). Then we enumerate the main known decidable fragments of Second-Order Unification.

2.1 Simply Typed λ-calculus

In this section, we give the definitions and elementary properties of simply typed λ-calculus which is the term-language of higher-order logic, hence the terms considered in Higher and Second-Order Unification. The "equality notion" required for Higher-Order Unification is the equivalence between terms under the conversion rules of the λ-calculus. The proofs and detailed explanations of this topic can be found, for instance, in (Barendregt, 1984; Hindley and Seldin, 1986).

2.1.1 Types

There are several varieties of λ-calculus (Barendregt, 1984). The one that is the basis of our study is the *simply typed λ-calculus*. In this language, λ-terms have "attached" a type. In some sense the type of a λ-term is a descriptor of its nature. Simply typed λ-calculus has nice computational properties in comparison to other λ-calculus variants, for instance simply typed λ-calculus is normalising while the untyped λ-calculus is not.

Definition 2.1 *Consider a finite set whose elements are called* atomic types. *The set of* types $\mathcal{T}$ *(for the simply typed λ-terms) is the smallest set inductively generated by the set of atomic types and the function type constructor $\rightarrow$, such that $(\tau_1 \rightarrow \tau_2)$ is a type whenever τ_1 and τ_2 are types.*
The order *of a type τ, noted by $o(\tau)$, is defined as follows:*

- *if* τ *is an atomic type then* $o(\tau) = 1$,
- *if* τ *has the form* $(\tau_1 \to \tau_2)$ *then* $o(\tau) = max\{1 + o(\tau_1), o(\tau_2)\}$

Remark: By convention, $\to$ associates to the right. We may think of type $\tau_1 \to \tau_2 \to \cdots \to \tau_n \to \tau$ as standing for the type of functions mapping n-tuples of type $(\tau_1 \times \tau_2 \times \cdots \times \tau_n)$ into entities of type τ.

2.1.2 Terms

Definition 2.2 *Let us assume given a* signature *of constants* $\Sigma = \bigcup_{\tau \in \mathcal{T}} \Sigma_\tau$, *such that, for every atomic type, there is at least a constant symbol. Similarly, for each type* $\tau \in \mathcal{T}$, *we assume given a denumerable set of variables of that type* $\mathcal{X}_\tau$, *and consider* $\mathcal{X} = \bigcup_{\tau \in \mathcal{T}} \mathcal{X}_\tau$.
The set of typed λ-terms *(or* λ-terms *for simplicity)* $\mathcal{T}(\Sigma, \mathcal{X})$ *is the smallest set inductively defined by:*

- *a constant or a variable of type* τ *is a* λ*-term of type* τ,
- *if* x *is a variable of type* τ_1 *and* t *is a* λ*-term of type* τ_2, *then* $\lambda x.\ t$ *is a* λ*-term of type* $\tau_1 \to \tau_2$. *This* λ*-term is a function where* λx *is the* λ-abstraction *and* t *is the* body.
- *If* u *is a* λ*-term of type* $\tau_1 \to \tau_2$ *and* v *is a* λ*-term of type* τ_1, *then* $(u\, v)$ *is a* λ*-term of type* τ_2. *This* λ*-term is an* application *where* function t *is* applied *over the* argument u.

The expression $\tau(t) = \tau_1 \to \ldots \to \tau_n \to \tau$ *denotes that the* λ*-term* t *has type* $\tau_1 \to \ldots \to \tau_n \to \tau$.
A λ*-term* s *is a* subterm *of* t *if* $s = t$ *or if, being* $t = (u\, v)$, s *is a subterm of* u *or a subterm of* v, *or if being* $t = \lambda x.\ u$, s *is a subterm of* u.

Definition 2.3 *The* size *of a* λ*-term is defined as follows:*

- $|x| = |c| = 1$, *for any variable* x *and constant* c,
- $|\lambda x.\, t| = |t|$, *for any* λ*-term* t *and variable* x,
- $|(u\, v)| = |u| + |v|$, *for any* λ*-terms* u *and* v.

The order *of a* λ*-term is the order of its type.*
Let τ *be an atomic type, and* $\tau(t) = \tau_1 \to \ldots \to \tau_n \to \tau$, *then,* t *is said to be of* arity n *(*n*-ary), noted* $arity(t) = n$.

Definition 2.4 *An occurrence of a variable* x *in a* λ*-term* t *is* bound, *if it occurs below a* λ*-abstraction (*a binder*)* λx *for it, i.e. if* t *has a subterm like* $\lambda x.\ t'$, *and* x *is a subterm of* t'. *Otherwise the variable is said to be* free. *If* x *is a free variable in* t, *it is said to be* bounded by *the external* λ*-binder in term* $\lambda x.\ t$.

Definition 2.5 *The set of free variables of a term t is noted $Var(t)$. A λ-term with no free variable is called a* closed λ-term.

Remark: By convention, application associates to the left, therefore, an expression like $(u\, v_1\, v_2\, \ldots\, v_n)$ is a notation for the λ-term $(\ldots((u\, v_1)\, v_2)\ldots v_n)$. We also may represent a sequence of λ-abstractions like the one of this λ-term: $\lambda x_1.\,(\lambda x_2.\,(\ldots(\lambda x_n.\,t)\,\ldots))$ in the form $\lambda x_1 \ldots x_n.\,t$ being t an application, a constant or a variable. We will often drop superfluous parentheses. We also follow the convention that the "dot" (of the λ-abstraction) includes as much right context as possible in the scope of its binder, so that, e.g., a λ-term $\lambda x.\, fab$ is to be interpreted as $(\lambda x.\,((fa)b))$.

From now on we will consider just one basic type (if nothing else is said), let's say ι, and we will often fail to specify types when they are clear from the context or when the specification adds nothing to the discussion. Notice also the usual convention that λ-terms of order one (*first-order terms*) denote individuals, λ-terms of order two (*second-order terms*) denote functions on individuals, etc.

2.1.3 Substitutions

Substitution is the main operation required for formulating the axioms (rules) that will define the convertibility relation of λ-calculus. But not only this, the notion of substitution is central to unification problems, in fact, unifiers are substitutions.

Definition 2.6 *A* substitution *is a finite mapping from variables to λ-terms, written as $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ where for each $i \in \{1..n\}$, x_i is a variable and t_i is a λ-term of the same type. Substitutions will be denoted by greek letters like $\sigma, \rho, \ldots$.*
Let σ be the following substitution: $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$:

- *the* domain *of σ is: $Dom(\sigma) = \{x_1, \ldots, x_n\}$,*
- *its* range *is: $Range(\sigma) = \{t_1, \ldots, t_n\}$,*

The substitution $\sigma|_A$ is the substitution σ restricted to the set of variables A. We say that σ is a restriction *of ρ if $\sigma = \rho|_{Dom(\sigma)}$. We say that σ is an* extension *of ρ if $\sigma|_{Dom(\rho)} = \rho$.*

Roughly speaking, the result of applying a substitution σ to a λ-term t is the λ-term t where the occurrences of its variables that are in $Dom(\sigma)$ have been replaced by the associated λ-term in the substitution. This replacements must be done carefully in order to avoid confusion between free and bound variables. There are two special situations that must be considered. See the following examples:

- let t be the λ-term $\lambda x.\, x$ and let σ be the substitution $[x \mapsto y]$, if we simply replace the occurrence of the variable x in t by y, we get $\lambda x.\, y$, while the variable x is bound in t, and thus we would rather expect the λ-term $\lambda x.\, x$,

- take now the same substitution σ, but now let t be the λ-term $\lambda y.\, x$, if we simply replace the occurrence of x in t by y, we obtain the λ-term $\lambda y.\, y$, capturing now the variable y, while we would rather expect to get the λ-term $\lambda z.\, y$.

These two situations are considered and treated correctly in the next definition.

Definition 2.7 *Let $\sigma = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ be a substitution and t a λ-term, then the* application of the substitution *σ to the λ-term t is noted as $\sigma(t)$ and is defined as follows:*

- *$\sigma(c) = c$, for any constant c,*
- *$\sigma(x_i) = t_i$, for any variable $x_i \in \{x_1, \ldots, x_n\}$,*
- *$\sigma(x) = x$, for any variable $x \notin Dom(\sigma)$,*
- *$\sigma(t\, u) = (\sigma(t)\, \sigma(u))$,*
- *$\sigma(\lambda x.\ t) = \lambda y.\ \sigma([x \mapsto y]\, u)$, where y is* a fresh variable*, with the same type as x, i.e. a variable that does not occur in t nor in $t_1, \ldots, t_n$ and that it is different from $x_1, \ldots, x_n$.*

Let the λ-term s be $\sigma(t)$ for some substitution σ, then s is said to be an instance *of t.*
The size of a substitution *$\sigma = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ is defined as:*

$$|\sigma| = |t_1| + \ldots + |t_n|$$

Definition 2.8 *A substitution is said to be* ground *if its range just contains closed terms.*

2.1.4 λ-equivalence

The theory of the λ-calculus is defined mainly by means of three equivalence (or convertibility) axioms: α, β and η-equivalence. The β and the η equivalences are often presented as oriented rules. These rules are the core of an strongly normalisable and confluent rewriting system that provides us with a normal form notion that we will use to decide equivalence between λ-terms.

Definition 2.9 *The λ-*equivalence *relation $=_\lambda$, is the minimal equivalent and congruent relation such that:*

$$\begin{array}{rcccl} \lambda x.\ t & =_\lambda & \lambda y.\ [x \mapsto y]\, t & \alpha & \text{(provided } y \text{ does not occur free in } t\text{)} \\ (\lambda x.\ t)\, s & =_\lambda & [x \mapsto s]\, t & \beta & \\ \lambda x.\ t\, x & =_\lambda & t & \eta & \text{(provided } x \text{ does not occur free in } t\text{)} \end{array}$$

i.e. that apart from the above equalities, for any λ-terms s, s', t and t', and for any variable x, the relation satisfies the congruence axioms:

$$\begin{array}{l} \textit{if } s =_\lambda s' \textit{ then } (t\,s) =_\lambda (t\,s') \\ \textit{if } t =_\lambda t' \textit{ then } (t\,s) =_\lambda (t'\,s) \\ \textit{if } t =_\lambda t' \textit{ then } \lambda x.\ t =_\lambda \lambda x.\ t' \end{array}$$

and the equivalence axioms:

$$\begin{array}{rcl} & & s =_\lambda s \\ \textit{if } s' =_\lambda s & \textit{then} & s =_\lambda s' \\ \textit{if } s =_\lambda s' \textit{ and } s' =_\lambda t & \textit{then} & s =_\lambda t \end{array}$$

Proposition 2.10 *Pairs of λ-equivalent λ-terms have the same type.*

Definition 2.11 *The* α-equivalence *relation is the minimum equivalent and congruent relation that satisfies the equivalence α of Definition 2.9. Similarly for the* β-equivalence *and the* η-equivalence *relations with respect to the β and the η equivalences.*

The α-equivalence relation captures the idea that bound variables can be renamed. The β-equivalence relation means that when applying a function $\lambda x.\ t$ over a term s, the formal parameter x of the function can be replaced by the argument s. The η-equivalence relation entails extensionality, which means that two functions are considered equal, if they behave equally for all arguments.
Remark: For simplicity, we assume that bound variables with different binders have different names. In the following we shall identify α-equivalent λ-terms, i.e. consider λ-terms as representatives of their α-equivalence classes.

Proposition 2.12 *Orienting the equivalences β and η, we obtain a confluent and terminating rewriting system between α-equivalent λ-terms formed by these two rules:*

$$\begin{array}{rcll} (\lambda x.\ t)\,u & \to_\beta & [x \mapsto u]\,t & \beta\textit{-reduction} \\ \lambda x.\ t\,x & \to_\eta & t & \eta\textit{-reduction (provided } x \textit{ does not occur in } t) \end{array}$$

Minimum subterms of a λ-term where the β-reduction rule can be applied, are called *β-redexes*. Similarly for the η-reduction rule and *η-redexes*.

Definition 2.13 *Let the λ-term t be of type $\tau_1 \to \tau_2$, then the* η-expansion *rule is defined as follows:*

$$t \to_{\eta^*} \lambda x.\ t\,x$$

provided, no β-redex is introduced, and x does not occur free in t neither.

Proposition 2.14 *The rewrite system formed by $\to_\beta$ and $\to_{\eta^*}$ is normalising. Moreover,*

$$(\to_\beta \cup \to_{\eta^*})^* = \to_\beta^* \circ \to_{\eta^*}^*$$

Proposition 2.15 *A λ-term t is in $\beta\eta$-*long normal form *if, and only if, it has the form:*

$$\lambda x_1 \ldots x_n.\ h\,t_1\ \ldots\ t_m$$

where $arity(h) = m$, t_i for $i \in \{1..m\}$ are λ-terms in $\beta\eta$-long normal form, and h (the head *of the λ-term) is either a constant or a variable.*

Example 2.16 Consider the λ-term $t = \lambda xyz.\ (z\,(x\,y))\,(\lambda x.\ x)\,f\,(\lambda x.\ x)$ where f has type $\iota \to \iota \to \iota$ (the remaining types can be inferred from the type of f). We can obtain its $\beta\eta$-long normal form by performing the following β-reduction and η-expansion steps:

$$\underbrace{\lambda xyz.\ (z\,(x\,y))\,(\lambda x.\ x)}_{\beta-redex}\ f\,(\lambda x.\ x) \quad \to_\beta \quad \underbrace{\lambda yz.\ (z\,((\lambda x.\ x)\,y))\,f}_{\beta-redex}\,(\lambda x.\ x)$$

$$\to_\beta \quad \lambda z.\ (z\,(\underbrace{(\lambda x.\ x)\,f}_{\beta-redex}))\,(\lambda x.\ x) \quad \to_\beta \quad \underbrace{\lambda z.\ (z\,f)\,(\lambda x.\ x)}_{\beta-redex}$$

$$\to_\beta \quad \underbrace{(\lambda x.\ x)\,f}_{\beta-redex} \quad \to_\beta \quad \underbrace{f}_{\eta^*-redex} \quad \to_{\eta^*} \quad \lambda x.\ \underbrace{f\ x}_{\eta^*-redex}$$

$$\to_{\eta^*} \quad \lambda x.\ \lambda y.\ (f\,x)\,y$$

■

Definition 2.17 *A* language has order n *if it is built over a signature where all constants are of order at most* $n+1$*, and a set of variables of order at most* n*.*

In the Unification perspective, as we will see, the important aspect of this last definition, is the bound on the order of free variables. As an alternative we can define order n languages as the ones where all terms in $\beta\eta$-long normal form may only contain variables of order at most n. Notice also that a term of order n can contain variables of any order as the following example shows:

Example 2.18 Let f be a unary fifth-order constant with type $\tau(f) = (((\iota \to \iota) \to \iota) \to \iota) \to \iota$ and let g be a unary fourth-order constant with type $\tau(g) = ((\iota \to \iota) \to \iota) \to \iota$, then the following term:

$$f\ g$$

that has type ι, hence is of order one, has a $\beta\eta$-long normal form that requires variables of order 3:

$$f(\ \lambda x.\ g(\lambda y.\ x(\lambda z.\ yz)))$$

where variable x has type $(\iota \to \iota) \to \iota$, thus order 3, variable y has type $\iota \to \iota$, thus order 2, and variable z has type ι. ■

Remark: If we keep the assumption that there is just an atomic type, whenever a λ-term is in $\beta\eta$-long normal form, we can infer the type of all the constants and variables occurring in it. In general we will assume that terms under discussion are in $\beta\eta$-long normal form.

We will also represent λ-terms in their *decurried form*, for instance, the λ-term $(\lambda xy.\ f\,x\,y)$, where x, y are ι-typed variables and f is a binary function symbol of type $\iota \to \iota \to \iota$, will be written as $\lambda xy.\ f(x, y)$.

As a convention and for the sake of clarity, in what follows we denote constants of atomic type by a, b, c, functions by f, g, h, bound variables of arbitrary type by x, y, z and free variables by capital letters, the first-order ones by X, Y, Z and the ones of functional type by F, G, H.

2.1.5 Unification

Once introduced the simply typed λ-calculus language, we can define the main operation of our study, *unification*.

Definition 2.19 *A* higher-order equation *is an unoriented pair $t \stackrel{?}{=} u$ of λ-terms with the same type and of arbitrary order.* Higher-Order Unification, *is the problem of, given a* system *(finite set) of higher-order equations $S = \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$, which* unknowns *are the free variables that occur in s_i and t_i for $i \in \{1..n\}$, to decide whether there exists a substitution σ such that $\sigma(s_i) =_\lambda \sigma(t_i)$, for all $i \in \{1..n\}$. Such σ is said to* unify *or* solve *S, and it is called* unifier.
Let σ be a unifier of the equation $s \stackrel{?}{=} t$, then we call the term $\sigma(s)$ and $\sigma(t)$ the common instance *of the equation.*

Definition 2.20 Second-Order Unification *is Higher-Order Unification restricted to second-order languages, therefore terms of the equations do not contain variables of order higher than* 2 *and constant symbols of order higher than* 3.

Definition 2.21 *The* size *of an equations system $S = \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$ is defined by $|S| = \sum_{i \in \{1..n\}} (|s_i| + |t_i|)$.*

Definition 2.22 Higher-Order Matching (Second-Order Matching) *is Higher-Order Unification (Second-Order Unification) where one of the λ-terms of each equation is a closed λ-term.*

Definition 2.23 *Let σ and σ' be two substitutions and $Dom(\sigma) \cup Dom(\sigma') = \{x_1, \ldots, x_n\}$, the* composition of substitutions *σ and σ' is defined as follows:*

$$\sigma \circ \sigma' = [x_1 \mapsto \sigma(\sigma'(x_1)), \ldots, x_n \mapsto \sigma(\sigma'(x_n))]$$

Definition 2.24 *A substitution σ is said to be* more general *than a substitution σ', noted as $\sigma \leq \sigma'$, if there exists a substitution ρ such that for every variable $x \in Dom(\sigma)$, $\sigma'(x) = \rho(\sigma(x))$, hence $\sigma'|_{Dom(\sigma)} = \rho \circ \sigma|_{Dom(\sigma)}$.*
A substitution σ is said to be a renaming *of σ', if $\sigma \leq \sigma'$ and $\sigma' \leq \sigma$.*

Definition 2.25 *Let S be an equations system. A unifier σ of S is a* most general unifier *of S when, for all unifier σ' of S, if $\sigma' \leq \sigma$ then $\sigma \leq \sigma'$.*

The notion of most general unifier of First-Order Unification has distinct interpretations in Higher-Order Unification. For instance, Baader and Snyder (2001) and Prehofer (1995), define most general unifier as "a unique substitution (modulo renaming of free variables) from which all other unifiers can be generated". This definition does not coincide with our definition of most general unifier in Definition 2.25.

On the other hand, the name of renaming comes from the First-Order Unification case, where the most general unifier is unique modulo renaming of free variables. As we will show in the following example, the treatment of bound variables is also relevant for *renamings* in Second-Order Unification.

Example 2.26 Consider the following equation:

$$F(a) \stackrel{?}{=} G(b)$$

for which substitution $\sigma = [F \mapsto \lambda x.\ H(x,b),\ G \mapsto \lambda x.\ H(a,x)]$ and substitution $\sigma' = [F \mapsto \lambda x.\ H'(x,b),\ G \mapsto \lambda x.\ H'(a,x)]$ are both most general unifiers. In fact, σ and σ' are renamings in the same sense as in First-Order Unification because H is a "renaming" of H' and viceversa. However there are still more renamings of σ, for instance, the following substitution:

$$\sigma'' = [F \mapsto \lambda x.\ H''(x,b,x),\ G \mapsto \lambda x.\ H''(a,x,a)]$$

is a renaming of σ because $\sigma''|_{\{F,G\}} = [H \mapsto \lambda xy.\ H''(x,y,x)] \circ \sigma|_{\{F,G\}}$, and also $\sigma = [H'' \mapsto \lambda xyz.\ H(x,y)] \circ \sigma''|_{\{F,G\}}$. As it is easy to see, the equation has infinitely many renamings. ■

Definition 2.27 *A set of substitutions Ω is a* minimal complete set of most general unifiers *of a unification equations system S, if and only if each element of Ω is a unifier of S incomparable to any other in Ω, and for every unifier σ of S, there exists a unifier σ' in Ω such that $\sigma' \leq \sigma$.*

Unification Problems are classified depending on the cardinality of minimal complete sets of most general unifiers[1].

Definition 2.28 *We call a Unification Problem:*

- unitary *if a minimal complete set of unifiers is either empty or a singleton,*
- finitary *if a finite minimal complete set of unifiers always exists,*
- infinitary *if a possibly infinite minimal complete set of unifiers always exists,*
- nullary *if no minimal complete set of unifiers may exist.*

Second-Order Unification is infinitary while Higher-Order Unification is nullary.

Remark: In what follows, we may sometimes use the word *problem* to refer to an equations system, then, for instance, a second-order unification problem will be an instance of the Second-Order Unification Problem. When the signature considered contains at least an n-ary function symbols (with $n \geq 2$), we can freely consider problems as just one equation instead of a set of equations.

[1] This notion does not only apply to Higher-Order Unification but to Unification in general.

2.2 Second-Order Unification Undecidability

Higher-Order Unification is undecidable, i.e. there is no algorithm that takes as argument a higher-order equations system and answers if it has a solution or not. This fact was shown independently by Huet (1973b) and by Lucchesi (1972). These proofs reduce the Post's Correspondence Problem to Third-Order Unification. Also Huet (1976), proved that Third-Order Unification is nullary because for a certain kind of equations there may exist an infinite chain of unifiers, each one more general than the previous one, without any most general one.

It was not until 1981, that the second-order case was shown undecidable by Goldfarb (1981). His proof is based on a reduction of the *Hilbert's tenth problem*: he shows how the problem of *given two polynomials* $P(X_1, \ldots, X_n)$ *and* $Q(X_1, \ldots, X_n)$ *whose coefficients are natural numbers, to answer if there exist natural numbers* $m_1, \ldots, m_n$ *such that* $P(m_1, \ldots, m_n) = Q(m_1, \ldots, m_n)$, can be reduced to Second-Order Unification.

We will now illustrate the main steps of the reduction. To encode polynomials, we need to be able to encode *natural numbers*, and the *addition* and the *multiplication* operators. We also need a mechanism to ensure that variables can only be instantiated by encodings of natural numbers.

Goldfarb's numbers are second-order terms of type $\iota \to \iota$ of the following form:

$$n^G = \lambda x.\ \overbrace{g(a, \ldots g(a, g(a, x))}^{n} \ldots)$$

where n^G stands for the representation of the natural number n. A normal term t of type $\iota \to \iota$ is a Goldfarb's number if and only if $[X \mapsto t]$ is a solution to the equation:

$$g(a, X(a)) \stackrel{?}{=} X(g(a, a))$$

The addition operation can be expressed by the third-order term:

$$add = \lambda xyz.\ x(y(z))$$

while multiplication requires an equation system. The following second-order problem:

$$\begin{aligned} Y(a, b, g(g(X_3(a), X_2(b)), a)) &\stackrel{?}{=} g(g(a, b), Y(X_1(a), g(a, b), a)) \\ Y(b, a, g(g(X_3(b), X_2(a)), a)) &\stackrel{?}{=} g(g(b, a), Y(X_1(b), g(a, a), a)) \end{aligned}$$

has a solution containing these mappings $[X_1 \mapsto m_1^G, X_2 \mapsto m_2^G, X_3 \mapsto m_3^G]$ if and only if $m_1 \cdot m_2 = m_3$.

Then we can encode the polynomial equations as second-order equations such that the second-order equations are solvable if and only if the polynomials equations are solvable.

Theorem 2.29 *[Goldfarb, 1981] Second-Order Unification is undecidable.*

One of the particularities to mention about Goldfarb's reduction, is that it does not require third-order constants, therefore, it allows to proof undecidability of Second-Order Unification even if we do not allow λ-bindings in equations.

This restriction is relevant for us because we will use it in the definition of terms for Context Unification (see Definition 3.9), and in the second-order language considered in Chapter 4, where the currying technique is used to reduce Second-Order Unification to Second-Order Unification with just one binary constant symbol. In both cases, constants have to be second-order typed.

Goldfarb's result shows that there are second-order (and therefore arbitrarily higher-order) languages where unification is undecidable. However there exist particular languages of arbitrarily high-order that have a decidable unification problem. For instance, Goldfarb's proof requires that the language to which the reduction is made contains at least one binary function symbol (the one required to codify Goldfarb's numbers). It has been shown by Farmer (1988) that the unification problem for *second-order monadic languages* (i.e., languages where function symbols are at most unary) is decidable, even more, it has been proved that it is NP-complete (Levy *et al.*, 2004). Also, Miller (1991a) defines a higher-order language for which unification is decidable, the so called *higher-order patterns*. Patterns are λ-terms in $\beta\eta$-long normal form, where the list of the arguments of any free variables is a list of pairwise distinct bound variables. We will come back to these decidable sub-cases in Section 2.4.

As we can see, the decidability/undecidability question for Second-Order Unification seems quite dependent on the syntactic characteristics of the language that we are considering. In this direction, the result of Goldfarb has been sharpened, just to mention some, by Farmer (1991), by Schubert (1998), by Levy (1998); Levy and Veanes (1998, 2000) and by Levy and Villaret (2002). The results of these works mainly consist of reductions of undecidable problems to second-order equations systems with languages that have particular syntactic requirements.

- The work of Farmer (1991) shows that there is an integer n such that Second-Order Unification is undecidable even if all second-order variables are unary and there are at most n of them, even more, first-order variables are not required to occur in equations.

- Schubert (1998) proves that Second-Order Unification is undecidable for systems of *simple equations*, i.e. equations where all arguments of free second-order variables do not contain free variables.

- The work of Levy (1998); Levy and Veanes (1998, 2000) is quite exhaustive and exhibits several reductions with very sharp results like, for instance, when each second-order variable occurs at most twice and there are only two second-order variables (as we will show, this case has been proved to be decidable for Linear Second-Order Unification (Levy, 1996)); or when there is only one second-order variable and it is unary, etc. Some of these results are obtained by a reduction from *Simultaneous Rigid E-Unification*

(see Degtyarev and Voronkov (1996) for an inverse reduction) to special fragments of Second-Order Unification.

- The work of Levy and Villaret (2002) shows that Second-Order Unification can be NP-reduced to Second-Order Unification where there is just one binary function symbol by means of currying (see Chapter 4). Applying this reduction to the results of Levy and Veanes (2000) proves that Second-Order Unification is undecidable for one binary function symbol and one second-order variable occurring four times.

Besides the undecidability of Second-Order Unification, another problem is that, unlike in First-Order Unification, most general unifiers may not be unique. In fact Second-Order Unification is infinitary, i.e. the minimal complete set of most general unifiers always exists, but can be infinite. For example the equation:

$$F(f(a,b)) \stackrel{?}{=} f(F(a),b)$$

has infinitely many incomparable most general unifiers of the form:

$$[F \mapsto \lambda x.\ \overbrace{f(\ldots(}^{n} x, \overbrace{b)\ldots b)}^{n}]$$

for $n \in \{0..\infty\}$.

2.3 Second-Order Unification Procedure

The problem of deciding if a given substitution is a unifier of a given problem is decidable: it suffices to apply the substitution to both sides of each equation, normalise the terms and check whether their normal forms are equal. Substitutions are denumerable, therefore Second-Order Unification is semidecidable. Obviously, a *generate and test* procedure is very inefficient and several authors have proposed unification procedures where the form of the terms in the equations is used to restrict somehow the search space (Darlington, 1973; Pietrzykowski, 1973; Huet, 1973a, 1975, 1976; Jensen and Pietrzykowski, 1976). The work of Pietrzykowski (1973) was the first in describing a sound and complete Second-Order Unification procedure that was later extended to the higher-order case by Jensen and Pietrzykowski (1976). We will now describe the second-order version of the procedure.

The main idea behind these algorithms is the same as in the Martelli Montanari algorithm for First-Order Unification. It simply consist of, at each step, trying to transform the equations into "more solved" ones from up to down, therefore the transformation applied depends on the "shape" of the heads of the chosen equation. The heads are *rigid* if they are constants or bound variables, and *flexible* if they are free variables. As at the end, all equations have to be solved, we can freely choose any equation to apply a transformation.

We will use the notation of transformations from (Gallier and Snyder, 1990) for describing unification processes. In this notation any state of the process is

represented by a pair $\langle S, \sigma \rangle$ where S is the equations system and σ is the substitution computed until that moment, i.e. the substitution leading from the initial problem to the actual one. The initial state is $\langle S_0, [] \rangle$ where S_0 is the original equations system. The procedure is described by means of *transformation rules* on states like $\langle S \cup E, \sigma \rangle \Rightarrow \langle \rho(S \cup E'), \rho \circ \sigma \rangle$ [2], where E is the chosen equation to be transformed into E' and ρ the substitution required for the transformation. The goal is to reach a *solved state* $\langle \emptyset, \sigma \rangle$ or to reach a search tree where no transformation rule can be applied anywhere.

Definition 2.30 *The transformations rules depending on the heads of the chosen equation are the following ones:*

- Rigid-rigid. *If we have an equation like*

$$E = \{\lambda x_1 \dots x_n.\ f(u_1, \dots, u_p) \stackrel{?}{=} \lambda x_1 \dots x_n.\ f(v_1, \dots, v_p)\}$$

  where f is a constant symbol or a bound variable, we can only apply the Simplification *rule. We need to propagate the equation over the arguments of both sides ensuring that no bound variable is "freed", therefore we transform E into*

$$E' = \bigcup_{i \in \{1 \dots p\}} \{\lambda x_1 \dots x_n.\ u_i \stackrel{?}{=} \lambda x_1 \dots x_n.\ v_i\}$$

  and the accumulated substitution does not change

$$\rho = []$$

- Flexible-rigid *(or Rigid-flexible, recall that equations are unoriented). If we have an equation like*

$$E = \lambda x_1 \dots x_n.\ F(u_1, \dots, u_p) \stackrel{?}{=} \lambda x_1 \dots x_n.\ g(v_1, \dots, v_q)$$

  we have two possibilities:

  - Projection. *We can guess that variable F projects on one of its arguments which must have the same type as $g(v_1, \dots, v_q)$, therefore we generate a substitution for it that guesses the argument*

$$\rho = [F \mapsto \lambda y_1 \dots y_p.\ y_i]$$

    where $i \in \{1 \dots p\}$, and y_i has the same type than $g(v_1, \dots, v_q)$. We transform E into

$$E' = \{\lambda x_1 \dots x_n.\ u_i \stackrel{?}{=} \lambda x_1 \dots x_n.\ g(v_1, \dots, v_q)\}$$

[2] The application of a substitution to an equation and to an equation system is the natural extension of the application of a substitution to a term.

– Imitation. *We guess that variable F imitates the beginning of the other side term g, then, we generate a substitution*

$$\rho = [F \mapsto \lambda y_1 \ldots y_p.\ g(F_1(y_1, \ldots, y_p), \ldots, F_q(y_1, \ldots, y_p))]$$

where $F_1, \ldots, F_q$ are new appropriately typed free variables. We transform E into

$$E' = \bigcup_{i \in \{1 \ldots q\}} \{\lambda x_1 \ldots x_n.\ F_i(u_1, \ldots, u_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ v_i\}$$

- Flexible-flexible. *If we have an equation like*

$$E = \{\lambda x_1 \ldots x_n.\ F(u_1, \ldots, u_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ G(v_1, \ldots, v_q)\}$$

we have four possibilities:

– Simplification. *When both head symbols are the same free variable, i.e. $F = G$, we can propagate the equation over the arguments of F ensuring that no bound variable is "freed", therefore we transform E into*

$$E' = \bigcup_{i \in \{1 \ldots p\}} \{\lambda x_1 \ldots x_n.\ u_i \stackrel{?}{=} \lambda x_1 \ldots x_n.\ v_i\}$$

and the accumulated substitution does not change

$$\rho = []$$

– Elimination. *We can eliminate the i-th parameter of one of the head variables by means of a substitution like*

$$\rho = [F \mapsto \lambda y_1 \ldots y_p.\ F'(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_p)]$$

and

$$E' = \left\{ \begin{array}{c} \lambda x_1 \ldots x_n.\ F'(u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_p) \\ \stackrel{?}{=} \\ \lambda x_1 \ldots x_n.\ G(v_1, \ldots, v_q) \end{array} \right\}$$

being F' a new appropriately typed free variable.

– Iteration. *We can also iterate the i-th parameter of one of the head variables by means of a substitution like*

$$\rho = [F \mapsto \lambda y_1 \ldots y_p.\ F'(y_1, \ldots, y_p, y_i)]$$

and

$$E' = \{\lambda x_1 \ldots x_n.\ F'(u_1, \ldots, u_p, u_i) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ G(v_1, \ldots, v_q)\}$$

being F' a new appropriately typed free variable.

– Identification. *when $F \neq G$ we can identify both variables, trying to fix a "maximal" common part. Therefore, we introduce a new free variable that denotes this "common part", being*

$$\rho = \left[\begin{array}{l} F \mapsto \lambda y_1 \ldots y_p . H(y_1, \ldots, y_p, F_1(y_1, \ldots, y_p), \ldots, F_q(y_1, \ldots, y_p)), \\ G \mapsto \lambda y_1 \ldots y_q . H(G_1(y_1, \ldots, y_q), \ldots, G_p(y_1, \ldots, y_q), y_1, \ldots, y_q) \end{array} \right]$$

and

$$E' = \left\{ \begin{array}{c} \lambda x_1 \ldots x_n . H(u_1, \ldots, u_p, F_1(u_1, \ldots, u_p), \ldots, F_q(u_1, \ldots, u_p)) \\ \stackrel{?}{=} \\ \lambda x_1 \ldots x_n . H(G_1(v_1, \ldots, v_q), \ldots, G_p(v_1, \ldots, v_q), v_1, \ldots, v_q) \end{array} \right\}$$

where $H, F_1, \ldots, F_q, G_1, \ldots, G_p$ are new appropriately typed free variables.

Notice that in all transformations, flexible heads are first-order variables when they have not arguments.

Theorem 2.31 *[Soundness] For any second-order equations system S, if there exists a derivation $\langle S, [] \rangle \Rightarrow^* \langle \emptyset, \sigma \rangle$, then σ is a unifier of S.*

Theorem 2.32 *[Completeness] If σ is a most general unifier of a second-order equations system S, then there exists a derivation $\langle S, [] \rangle \Rightarrow^* \langle \emptyset, \sigma \rangle$.*

It is easy to see that a procedure based on these rules is highly non-deterministic. A complete strategy for applying these rules could be: while the equations system contain rigid-rigid equations, apply simplification. The choice of the equation is *don't care*, i.e. we never need to backtrack to firstly try another equation since all of them must be solved, and simplification is the only applicable rule for rigid-rigid equations.

Then we could try to solve the flexible-rigid equations but now, the choice of the rule to apply, and even, the projected argument in the projection rule, introduces a *don't know* non-determinism, i.e. this choice could require some backtrack step.

Once there are no flexible-rigid nor rigid-rigid equations in the system (i.e. the system is in *presolved form*), we could solve the flexible-flexible ones. Unfortunately, the don't know non-determinism of the flexible-flexible rules is even more explosive and makes implementations impracticable.

2.3.1 Pre-Unification

Huet noticed that a flexible-flexible equation is always trivially solvable (we can always guess a common constant symbol and make the head variables of both sides of the equation to be instantiated by it[3]) and defined a method in

[3]Remember that we are under the assumption that we have a constant in each atomic type, therefore we can always find such a closed term as solution.

(Huet, 1975), named *preunification*[4]. This method consist of applying just the rigid-rigid and flexible-rigid rules until an equations system with just flexible-flexible equations is obtained, thus a solvable system is reached. Whenever an equations system is solvable, this method reaches a unifier. Therefore this method can be used to decide solvability of an equations system and some of the search branching explosion due to the don't know non-determinism of the flexible-flexible rules is avoided. Unfortunately, the imitation rule is enough to make this method non-terminating.

The fact that in higher-order logic, testing for unifiability is much simpler than enumerating unifiers, motivates the design of proof-search methods such as *constrained resolution* (Huet, 1973a), that require only the testing of unifiability and not the enumeration of solutions.

2.3.2 Regular Search Trees

An interesting approach to Huet's algorithm is due to Zaionc (1986), who remarks that when the number of equations system that we can generate from a given equations system is finite, i.e. when the number of distinct nodes in the search tree is finite modulo free variables renaming, we can compute this set of equations systems. If this finite set does not contain any solved state, then we know that the problem is unsolvable. In this way he sharpened Huet's algorithm by proposing an algorithm that reports failure more often that Huet's. For instance, for the equation:

$$X(a) \stackrel{?}{=} f(X(a))$$

Huet's algorithm would construct an infinite tree with no solved state, while after an imitation step with substitution $[X \mapsto \lambda x.\, f(X'(x))]$, we obtain after simplification, the equation:

$$X'(a) \stackrel{?}{=} f(X'(a))$$

that is the same equation, modulo free variable renaming, as the original one. The other chance is to apply projection and obtain:

$$a \stackrel{?}{=} f(a)$$

that is trivially unsolvable. Thus we obtain a search tree with just two "distinct states" and where none of them is a solved one, therefore Zaionc algorithm would report a failure.

Moreover, when the number of equations generated by a given equation is finite and it is solvable, the set of minimal unifiers may still be infinite, but it can be described by a grammar. For instance, for the equation:

$$F(f(a,b)) \stackrel{?}{=} f(F(a),b)$$

[4]In fact preunification was defined for Higher-Order Unification, therefore it also applies to the second-order case.

being σ the elementary substitution $[F \mapsto \lambda x.\ x]$ and being τ the elementary substitution $[F \mapsto \lambda x.\ f(F(x), b)]$, all the most general unifiers have the form $\sigma \circ \tau \circ \tau \circ \ldots \circ \tau$. Such substitutions can be represented by the words $\sigma\tau\tau\ldots\tau$. These words can be produced by the grammar:

$$\begin{array}{lcl} s & \rightarrow & \sigma \\ s & \rightarrow & s\tau \end{array}$$

2.4 Decidable Subcases

A lot of effort has also been made in identifying decidable subcases of Higher and Second-Order Unification. In this subsection we present some of them which are obtained by restricting the order, the arity or the number of occurrences of variables, or by just considering terms of a special form.

Besides these decidable cases, there remain three prominent subclasses of problems: Higher-Order Matching, Linear Second-Order Unification and Context Unification, whose decidability is still an open question. At the end of this section, we will present the main advances about Higher-Order Matching decidability, but we left Context Unification and Linear Second-Order Unification to be introduced in detail in next chapter, as well as the main results about them.

2.4.1 First-Order Unification

The first decidable subcase of Higher-Order Unification is obviously First-Order Unification. When all the variables of a problem have atomic type, i.e. they are first-order variables, all the constants have at most second-order types and the terms in the equations have first-order types, then the problem is simply First-Order Unification, that as we have already said in the Introduction Chapter, it is decidable (Martelli and Montanari, 1976; Robinson, 1965; Venturini-Zilli, 1975).

2.4.2 Pattern Unification

In mathematics, one often define functions by an equation of the following shape:

$$F(x, y) = x \cdot x + y \cdot y$$

and we actually mean that F is a function that has two arguments that are raised to the square and added:

$$F = \lambda xy.\ (x \cdot x + y \cdot y)$$

The particular shape of the first equation shows us that the function, the free variable, is applied to distinctly named arguments (that in fact are distinct bound variables). This motivates the study of unification problems where the higher-order free variables can only be applied to distinct bound variables.

A *pattern* is a term t such that for every subterm of the form $F(u_1, \ldots, u_n)$, where F is a free variable, the terms $u_1, \ldots, u_n$ are distinct bound variables of t. For instance, the following equation:

$$\lambda x_1 x_2 x_3.\ F(x_1, x_2) \stackrel{?}{=} \lambda x_1 x_2 x_3.\ f(\lambda y_1.\ G(x_1, y_1), F(x_2, x_3))$$

is a Pattern Unification equation.

Like First-Order Unification, Pattern Unification is decidable in polynomial time and when a unification problem has a unifier, it has only one most general unifier (Miller, 1991a; Nipkow, 1993). The algorithms for both problems have also some similarities, in particular, the occur-check plays an essential role in both cases. The relation between both problems is better understood when considering quantifier permutation in mixed prefixes (Miller, 1992).

Pattern Unification is used in higher-order logic programming (Nadathur and Miller, 1998). Prehofer (1995) studies some decidable extensions of patterns and shows their usefulness in functional and logic programming.

2.4.3 Monadic Second-Order Unification

As we have already shown, Goldfarb's undecidability proof requires a language with a binary constant symbol. Thus, a natural problem to investigate is Second-Order Unification where the language contains only unary functions, i.e. constants with a single argument. This problem, called *Monadic Second-Order Unification* has been proved decidable by Farmer (1988). Recently is has been problem that in fact it is NP-complete (Levy *et al.*, 2004).

Farmer's proof exploits the similarity between closed monadic terms of atomic type and words. A term like $f_1(f_2(\ldots f_n(c)\ldots))$ can be represented by the word $f_1 f_2 \ldots f_n c$. Then, a unification equations system in such a language can be reduced to a satisfaction equivalent Word Unification equation system, and Word Unification is known to be decidable (Makanin, 1977).

In Monadic Second-Order Unification, the set of minimal unifiers may be infinite, for instance, the following equation:

$$\lambda y.\ f(X(y)) \stackrel{?}{=} \lambda y.\ X(f(y))$$

which is equivalent to the word equation $fX \stackrel{?}{=} Xf$, has an infinite number of minimal solutions like $[X \mapsto \lambda x.\ x], [X \mapsto \lambda x.\ f(x)], [X \mapsto \lambda x.\ f(f(x))], \ldots$ corresponding to the solutions of the word problem: $[X \mapsto \epsilon], [X \mapsto f], [X \mapsto ff], \ldots$ In fact, for any natural number n the following substitution:

$$[X \mapsto \lambda x.\ f^n(x)]$$

is a unifier of the previous monadic equation. Farmer proposes to describe minimal unifiers using so called *parametric terms*. These parametric terms, remind Zaionc's descriptions of unifiers by means of grammars.

2.4.4 Second-Order Unification With Linear Occurrences of Second-Order Variables

In a second-order equation like:

$$\lambda x_1 \dots x_n.\ F(u_1, \dots, u_p) \stackrel{?}{=} \lambda x_1 \dots x_n.\ f(v_1, \dots, v_q)$$

we could perform a projection and replace the variable X by a closed term like $\lambda x_1 \dots x_n.\ x_i$ for some $i \in \{1..p\}$, thus the number of variables in the problem decreases. And we could also apply imitation rule and simplify the equation, obtaining the following equations instead of the original one:

$$\begin{gathered}\lambda x_1 \dots x_n.\ F_1(u'_1, \dots, u'_p) \stackrel{?}{=} \lambda x_1 \dots x_n.\ v'_1 \\ \dots \\ \lambda x_1 \dots x_n.\ F_q(u'_1, \dots, u'_p) \stackrel{?}{=} \lambda x_1 \dots x_n.\ v'_q\end{gathered}$$

which seems to be smaller than the original equation because the equations seem smaller, but this is not necessarily true because the variable F may have occurrences in terms $u_1, \dots, u_p, v_1, \dots v_q$, and therefore, terms $u'_1, \dots, u'_p, v'_1, \dots v'_q$ could be bigger than the original arguments of F and f. Nevertheless, when second-order variables are restricted to occur just once, these occurrences of the substituted variable in the arguments are not possible, therefore terms $u'_1, \dots, u'_p, v'_1, \dots v'_q$ are in fact the same original arguments $u_1, \dots, u_p, v_1, \dots v_q$ and the resulting equations are effectively smaller, thus a terminating process can be achieved (Dowek, 1993).

2.5 Higher-Order Matching

Matching is the particular case of unification where one side of each equation does not contain any unknown. Higher-Order Matching decidability is conjectured since Huet (1976), and still remains unproved.

The first positive result is the decidability of Second-Order Matching (Huet, 1976). The proof of this result is based in the measure $(SizeG, \sharp Vars)$ where $SiseG$ is the sum of the sizes of the original ground side of each equation and $\sharp Vars$ is the number of variables in the equations system. This measure decreases at each application of the rules of Huet's preunification algorithm, in fact, imitation will always be followed by a simplification. Notice that the flexible-flexible case will never occur. Second-Order Matching equations systems have a finite set of minimal solutions and has been proved to be NP-complete by Baxter (1977).

As soon as we have a free third-order variable (unknown), the required algorithm is more complex and may produce an infinite number of minimal solutions. For instance:

$$\lambda x.\ F(x, \lambda y.\ y) \stackrel{?}{=} \lambda x.\ x$$

has an infinite number of solutions of the form $[F \mapsto \lambda x f.\ f(f(\dots f(x)\dots))]$. Thus, we cannot use Huet's algorithm as a terminating algorithm for this case. Nevertheless, terminating algorithms exist for third (Dowek, 1992, 1994), and

even for fourth-order matching (Padovani, 1995, 2000). There exists also a decidability proof for third and fourth-order matching that uses tree-automata, in a similar way as Zaionc with grammars (see Subsection 2.3.2), to recognise solutions of a given problem (Comon and Jurski, 1997).

Recently, it has been proved that Higher-Order β-Matching, (matching of λ-terms not considering η-equivalence), is undecidable (Loader, 2003). But this result does not shed any light about the full Higher-Order Matching problem. The same applies to the positive result about decidability of Linear Higher-Order Matching (Salvati and de Groote, 2003; Dougherty and Wierzbicki, 2002), see Section 3.5.

2.6 Summary

In this chapter we have introduced simply typed λ-calculus, Unification and its related concepts, Second (and Higher)-Order Unification and the main results related with these problems, such as their undecidability, the existence of sound and complete procedures for them, and the existence of several decidable fragments. About these fragments, we have seen, that the frontier between decidability and undecidability is not completely defined and seems to depend on particular syntactic restrictions like the number of occurrences of each variable, the signature considered or the shape of the terms.

Chapter 3

Linear Second-Order and Context Unification

In this chapter, we define Linear Second-Order Unification and Context Unification, and introduce the main known results about these problems. Roughly speaking, Linear Second-Order Unification is Second-Order Unification where second-order variable instantiations must be linear λ-terms while, Context Unification is an extension of First-Order Unification where variables can denote not only first-order terms, but also context, i.e. terms with a *hole* or *distinguished position*. Context Unification can also be seen as a Linear Second-Order Unification restriction where neither third-order constants nor λ-bindings are allowed in the equations.

3.1 Linear Second-Order Unification

Definition 3.1 *A λ-term is said to be a* linear *if, written in $\beta\eta$-long normal form, every bound variable occurs just once. In other words, every subterm of the form $\lambda x.\ t$, contains only one occurrence of x free in t.*

Fact 3.2 *Linearity of λ-terms is preserved under β and η-reduction, and η-expansion. Therefore, if t and u are linear, $[X \mapsto u](t)$ is also linear.*

Definition 3.3 *A substitution is said to be* linear *if it maps variables to linear λ-terms.*

Definition 3.4 *The* Linear Second-Order Unification Problem *is the problem of deciding, given a second-order equation system, whether there exists a linear substitution that solves it.*

As can be seen from the definition, the presentation of linear second-order equations is the same as the one of "general" second-order equations. Notice that λ-terms of the equations are not required to be linear, linearity is only

required in variable instantiations. Obviously any equation that is solvable as a linear second-order equation is also solvable as a second-order equation, but the inverse is not true.

Example 3.5 The following equation:

$$F(a) \stackrel{?}{=} g(\lambda y.\ f(a))$$

considered as a Second-Order Unification equation has these two unifiers:

$$\begin{array}{lcl} \sigma & = & [F \mapsto \lambda x.\ g(\lambda y.\ f(a))] \\ \sigma' & = & [F \mapsto \lambda x.\ g(\lambda y.\ f(x))] \end{array}$$

but as we can observe none of these substitutions is linear. On the one hand, σ is not linear because x does not occur in the body of $\lambda x.\ g(\lambda y.\ f(a))$ and y does not occur in the body of $\lambda y.\ f(a)$ either. On the other hand, σ' is not linear because the variable y does not occur in the body of $\lambda y.\ f(x)$. Hence, the equation is unsolvable when considered as a Linear Second-Order Unification equation. ■

The interest of Linear Second-Order Unification arises in fields like automated deduction (Levy and Agustí, 1996) or computational linguistics (Pinkal, 1995). Decidability of Linear Second-Order Unification is a prominent open question (as well as decidability of Context Unification) that remains open after more than 10 years. Nevertheless, as well as in Second-Order Unification, a semi-decision procedure exists (Levy, 1996; Cervesato and Pfenning, 1997), and some decidable fragments have been found (Levy, 1996).

3.1.1 Sound and Complete Procedure

A naive semi-decision procedure could be the same "*generate and test*" proposed for Second-Order Unification but now considering just linear solutions. We could also use the Pietrzycowski semi-decision procedure (see Definition 2.30) and then accept just the linear unifiers.

Levy (1996), proposes an accurate modification of the transformation rules of the Pietrzycowski procedure, obtaining a less redundant one because it avoids the use of the prolific *elimination* and *iteration* rules. Even more, Levy's procedure only computes unifiers of the form $\rho \circ \sigma$ where σ is a most general unifier and ρ instantiates some variables by $(\lambda x.\ x)$.

This procedure has the same presentation as the one of Definition 2.30. Recall that we are always considering normalised terms, therefore any equation $s \stackrel{?}{=} t$ will always have the following form:

$$\lambda x_1 \ldots x_n.\ h(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ h'(t_1, \ldots, t_q)$$

since if s and t have the same type, then they must have the same number of more external λ-bindings. We also give the same name to these bound variables.

Definition 3.6 *All transformation rules have the form:*

$$\langle S \cup \{s \stackrel{?}{=} t\}, \sigma\rangle \Rightarrow \langle \rho(S \cup R), \rho \circ \sigma\rangle$$

where for each rule, the transformation $\{s \stackrel{?}{=} t\} \Rightarrow R$ *and the linear second-order substitution* ρ *are defined as follows:*

1. Simplification. *If* h *is a constant or a bound variable then:*

$$\begin{array}{c} \lambda x_1 \ldots x_n.\ h(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ h(t_1, \ldots, t_p) \\ \Downarrow \\ \bigcup_{i \in \{1..p\}} \lambda x_1 \ldots x_n.\ s_i \stackrel{?}{=} \lambda x_1 \ldots x_n.\ t_i \end{array}$$

   and $\rho = []$.

2. Imitation. *If* f *is a constant and* F *is a free variable then:*

$$\begin{array}{c} \lambda x_1 \ldots x_n.\ f(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ F(t_1, \ldots, t_q) \\ \Downarrow \\ \bigcup_{i \in \{1..p\}} \{\lambda x_1 \ldots x_n.\ s_i \stackrel{?}{=} \lambda x_1 \ldots x_n.\ F_i(t_{i_1}, \ldots, t_{i_{q_i}})\} \end{array}$$

   and $\rho = [F \mapsto \lambda y_1 \ldots y_q.\ f(F_1(y_{i_1^1}, \ldots, y_{i_{r_1}^1}), \ldots, F_p(y_{i_1^p}, \ldots, y_{i_{r_p}^p}))]$.

   Where F_i *(for* $i \in \{1..p\}$*), are appropriately typed fresh free variables and the set of indices* $\{\{i_1^1 \ldots i_{r_1}^1\} \ldots \{i_1^p \ldots i_{r_p}^p\}\}$ *is a partition of the indices* $\{1 \ldots q\}$ *ensuring that* $\rho(F)$ *is a linear term. Notice that when* $r_j = 0$ *then* F_j *is a first-order variable.*

3. Projection. *If* h *is a constant or a bound variable and* F *is a free variable, and* $h(s_1, \ldots, s_p)$ *and* t' *have the same type, then:*

$$\begin{array}{c} \lambda x_1 \ldots x_n.\ h(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ F(t') \\ \Downarrow \\ \lambda x_1 \ldots x_n.\ h(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ t' \end{array}$$

   and $\rho = [F \mapsto \lambda y.\ y]$.

4. Flexible-Flexible rule with equal-heads (or Simplification rule in Flex-Flex case). *If* F *is a free variable, then:*

$$\begin{array}{c} \lambda x_1 \ldots x_n.\ F(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ F(t_1, \ldots, t_p) \\ \Downarrow \\ \bigcup_{i \in \{1..p\}} \lambda x_1 \ldots x_n.\ s_i \stackrel{?}{=} \lambda x_1 \ldots x_n.\ t_i \end{array}$$

   and $\rho = []$.

5. Flexible-Flexible rule with distinct-heads. *If* F *and* G *are free distinct variables,* $p' < q$ *and* $q' < p$*, then*

$$\lambda x_1 \ldots x_n.\ F(s_1, \ldots, s_p) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ G(t_1, \ldots, t_q)$$
$$\Downarrow$$
$$\{\ \bigcup_{j\in\{1..p'\}} \lambda x_1 \ldots x_n.\ F_j(s_{i_1^j}, \ldots, s_{i_{r_j}^j}) \stackrel{?}{=} \lambda x_1 \ldots x_n.\ t_{k_j}\ \}$$
$$\cup$$
$$\{\ \bigcup_{j\in\{1..q'\}} \lambda x_1 \ldots x_n.\ s_{k'_j} \stackrel{?}{=} \lambda x_1 \ldots x_n.\ G_j(t_{l_1^j}, \ldots, t_{l_{r_j}^j})\ \}$$

and $\rho =$

$[F \mapsto \lambda y_1 \ldots y_p.\ H(F_1(y_{i_1^1}, \ldots, y_{i_{r_1}^1}), \ldots, F_{p'}(y_{i_1^{p'}}, \ldots, y_{i_{r_{p'}}^{p'}}), y_{k'_1}, \ldots, y_{k'_{q'}}),$

$G \mapsto \lambda z_1 \ldots z_q.\ H(z_{k_1}, \ldots, z_{k_{p'}}, G_1(z_{l_1^1}, \ldots, z_{l_{r_1}^1}), \ldots, G_{q'}(z_{l_1^{q'}}, \ldots, z_{l_{r_{q'}}^{q'}}))].$

Where F_j *(for* $j \in \{1..p'\}$*),* G_j *(for* $j \in \{1..q'\}$*) and* H *are appropriately typed fresh free variables and where* $\{\{i_1^1 \ldots i_{r_1}^1\} \ldots \{i_1^{p'} \ldots i_{r_{p'}}^{p'}\}\{k'_1 \ldots k'_{q'}\}\}$ *is a partition of the indices* $\{1 \ldots p\}$ *ensuring that* $\rho(F)$ *is a linear term, and* $\{\{k_1 \ldots k_{p'}\}\{l_1^1 \ldots l_{r_1}^1\} \ldots \{l_1^{q'} \ldots l_{r_{q'}}^{q'}\}\}$ *is a partition of the indices* $\{1 \ldots q\}$ *ensuring that* $\rho(G)$ *is a linear term. Notice that when* $r_j = 0$ *then* F_j *(or* G_j*) is a first-order variable.*

The key point of this algorithm is that the *elimination* rule and the *iteration* rule of the general second-order procedure, are not required. In order to ensure linearity of the substitutions the restrictions made in the set of indices of bound variables, are crucial.

Theorem 3.7 *[Soundness, (Levy, 1996)] For any second-order unification equation system* S*, if there exists a derivation* $\langle S_0, []\rangle \Rightarrow^* \langle \emptyset, \sigma\rangle$*, then* σ *is a minimal linear second-order unifier of* S_0*.*

Theorem 3.8 *[Completeness, (Levy, 1996)] If* σ *is a minimum linear second-order unifier of the second-order unification equation system* S_0*, then there exists a transformation sequence* $\langle S_0, []\rangle \Rightarrow^* \langle \emptyset, \sigma\rangle$*.*

One of the decidability fragments described by Levy (1996) is the one where no free variable occurs more than twice. The proof is based on the fact that each application of the transformation rules does not increase certain *size* measure for the equations. Then, noticing that for any finite signature and given size there are finitely many unification equation systems modulo renaming of free variables, it is enough to control loops to ensure the termination of the previous procedure. This argument resembles the one of Zaionc (1986) (see Section 2.3.2).

The decidability of this particular fragment of Linear Second-Order Unification contrasts with the undecidability of the same fragment considered as general Second-Order Unification (Levy and Veanes, 2000).

3.2 Context Unification

Context Unification can be defined from two perspectives. The main difference between both is the arity of free variables or unknowns but, as we will show in

Subsection 3.2.3, this characterisation does not imply any difference with respect to decidability.

1. From the "*First-Order Unification*" perspective, equations are defined over the algebra of terms and contexts. Terms are first-order terms without any constant of order higher than two, hence without bound variables. Context variables are used as unary second-order variables and can be instantiated by contexts. Contexts are terms with one occurrence of an special symbol $\bullet$ that denotes the *hole* of the context. This hole is the place where, when a context is "applied" to a term, the "argument" of the context must be placed. This perspective is the one followed, for instance, in (Comon, 1992a), where Context Unification is used in completion for some rewrite systems, or in (Schmidt-Schauß, 1996; Schmidt-Schauß and Schulz, 1998, 1999, 2002b), where, just to mention one, Schmidt-Schauß (1996) uses a decidable fragment of Context Unification to prove decidability of Distributive Unification.

2. From the "*Second-Order Unification*" perspective, contexts are linear second-order λ-terms without any constant of order higher than two. Terms in the equations are first-order terms without any constant of order higher than two and hence without bound variables either, in other words, second-order λ-terms as the ones used by Goldfarb (1981) (see Section 2.2). Therefore second-order (context) variables can have any arity (greater than one), and nor λ-abstraction nor bound variable occur in the equations but in substitutions. This is mainly the perspective of Levy and Villaret (2000, 2001, 2002), and comes from the study of Linear Second-Order Unification initiated by Levy (1996).

The following two Subsections provide the definition of *Context Unification* from both perspectives.

3.2.1 Context Unification From the First-Order Unification Perspective

Definition 3.9 *Let us assume a given finite* first-order signature $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$ *where constants in* Σ_0 *are first-order typed and constants in* Σ_i*, for* $i \geq 1$*, are second-order typed and have arity* i*. Similarly, we assume a given denumerable set of variables* $\mathcal{X} = \mathcal{X}_0 \cup \mathcal{X}_1$ *where* $\mathcal{X}_0$ *are first-order variables and* $\mathcal{X}_1$ *are* context variables.
A term *is a first-order* λ*-term in* $\mathcal{T}(\Sigma, \mathcal{X})$*, hence apart from the usual first-order terms, the application of a context variable to a term is also a term.*
A Context C *is a special term over the extended signature* $\Sigma \cup \{\bullet\}$*, where* $\bullet$ *is a constant symbol of arity* 0 *named* hole*, such that the hole occurs just once in* C*. The application of a context* C *to a term* t *(argument), denoted as* $C(t)$*, is the term resulting from replacing the occurrence of the constant symbol* $\bullet$ *with the term* t*, in the context* C*.*

Remark: Notice that this definition of terms coincides with the one of the terms used by Goldfarb (1981) (see Section 2.2), with the restriction that context variables are unary second-order variables. We will often not distinguish explicitly between *terms* and λ-*terms*, when the distinction is not relevant or can be inferred from the context.

Definition 3.10 *A* Context Substitution *is a substitution that maps first-order variables to terms and context variables to contexts.*
The application of a context substitution to a term is the term resulting from replacing first-order variables with the corresponding terms mapped by the substitution and context variables with their corresponding contexts.

Definition 3.11 *The* Context Unification Problem *is the problem of deciding, given an equation system over terms, whether there exists a context substitution that solves it.*

Example 3.12 Consider the following context equation:

$$F(F(a)) \stackrel{?}{=} f(f(G(a), b), X)$$

where F and G are context variables and X is a first-order variable. Substitution $\sigma = [X \mapsto b, F \mapsto f(\bullet, b), G \mapsto \bullet]$ solves the equation, hence, it is a unifier:

$$\begin{array}{rlll} \sigma(F(F(a))) = & f(\bullet, b)(f(\bullet, b)(a)) & = & \\ = & f(f(a, b), b) & = & \\ = & (f(f(\bullet(a), b), b)) & = \sigma(f(f(G(a), b), X)) & \end{array}$$

■

3.2.2 Context Unification From the Second-Order Unification Perspective

Contexts can also be seen as linear second-order λ-terms with just an external λ-abstraction where the "hole" is a variable occurrence bound by that λ-abstraction. Then, the application of a context substitution becomes the application of the linear second-order substitution followed by the reduction of the introduced β-redexes.

Definition 3.13 *We assume a given finite* second-order signature $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$ *where constants in* Σ_0 *are first-order typed and constant symbols in* Σ_i*, for* $i \geq 1$*, are second-order typed and have arity* i*. Similarly, we assume a given denumerable set of variables* $\mathcal{X} = \bigcup_{i=0}^{n} \mathcal{X}_i$ *where* $\mathcal{X}_0$ *are first-order variables and, for* $i \geq 1$*,* $\mathcal{X}_i$ *are second-order variables.*
A context equation *is a pair of first-order* λ*-terms over this signature.*

Definition 3.14 *The* Context Unification Problem *is the problem of deciding, given a context equations system, whether there exists a linear second-order substitution that solves it.*

Example 3.15 Following this definition, the substitution σ of Example 3.12 has the following form $\sigma = [X \mapsto b, F \mapsto \lambda x.\ f(x,b), G \mapsto \lambda x.\ x]$ and it is, of course, a unifier also:

$$\begin{array}{rll} \sigma(F(F(a))) = & \lambda x.\ f(x,b)(\lambda x.\ f(x,b)(a)) & =_\beta \\ =_\beta & f(f(a,b),b) & _\beta = \\ _\beta = & (f(f((\lambda x.\ x)(a),b),b)) & = \sigma(f(f(G(a),b),X)) \end{array}$$

■

3.2.3 Comparison Between both Perspectives

In the following we will consider Context Unification from the Second-Order Unification perspective, hence we will allow n-ary ($n \geq 1$) context (second-order) variables. Next we will show why this assumption does not affect to the decidability of the problem. However, Salvati and de Groote (2003) show that in the case of matching, this distinction affects to the complexity of the problem (see Section 3.5).

The use of the Second-Order Unification perspective of Context Unification is motivated by means of the following example.

Example 3.16 The following problem:

$$F(a) \stackrel{?}{=} G(b)$$

has the following most general unifier when allowing n-ary context variables:

$$\sigma \quad = \quad [F \mapsto \lambda x.\ H(x,b),\ G \mapsto \lambda x.\ H(a,x)]$$

where H is a binary context variable. However, if we restrict ourselves to unary context variables, σ could not be represented. Even worse, to find a possible unifier for this problem, we need to use an n-ary ($n \geq 2$) constant symbol:

$$\sigma' \quad = \quad [F \mapsto \lambda x.\ f(x,b),\ G \mapsto \lambda x.\ f(a,x)]$$

but notice that f does not occur in the equation. Therefore, the solvability of this equation depends on the signature that we are considering. ■

In order to prove that the arity of context variables does not play any role in decidability of Context Unification, we reduce the Context Unification with n-ary context variables to Context Unification with unary variables (Levy and Villaret, 2000). Similar ideas are also used by Schmidt-Schauß (1999a, 2004).

Given a context unification problem S with n-ary context variables over a signature Σ, if Σ does not contain any constant with arity $n \geq 2$ and a first-order constant, we include them in Σ. In the case of n-ary Context Unification this modification of the signature does not affect to the solvability of the problem. Then, we construct a new context unification problem S' without n-ary context variables by iteratively applying the following unarise rule, until all non-unary context variables of the problem disappear.

Definition 3.17 *The rule* unarise *consists of, given a context unification problem S, for any n-ary context variable F in S with $n \geq 2$, guessing a p-ary constant symbol g with $p \geq 2$, from the (previously enlarged if required) signature. In a second step, we guess a partition of $\{1, \ldots, n\}$ into $p \leq n$ many* disjoint *subsets such that $\bigcup_{i \in [1..p]} \{c_1^i, \ldots, c_{q_i}^i\} = \{1, \ldots, n\}$, and at least two of them are non-empty. We instantiate F in S by the following substitution:*

$$[F \mapsto \lambda x_1 \cdots \lambda x_n.\ F_0(g(F_1(x_{c_1^1}, \ldots, x_{c_{q_1}^1}), \ldots, F_p(x_{c_1^p}, \ldots, x_{c_{q_p}^p})))]$$

where F_0 is a fresh unary context variable and $F_1, \ldots, F_p$ are (maybe non-unary) fresh context or first-order variables.

Example 3.18 Consider the following n-ary context unification problem:

$$F(G(a, b)) \stackrel{?}{=} G(F(a), b) \tag{3.1}$$

One of its infinitely many solutions is (see Figure 3.1):

$$\sigma = [\quad \begin{array}{l} F \mapsto \lambda x.\ H(H(x, b), b), \\ G \mapsto \lambda x, y.\ H(H(H(H(x, b), y), b), b)] \end{array} \tag{3.2}$$

where H is a fresh context variable.

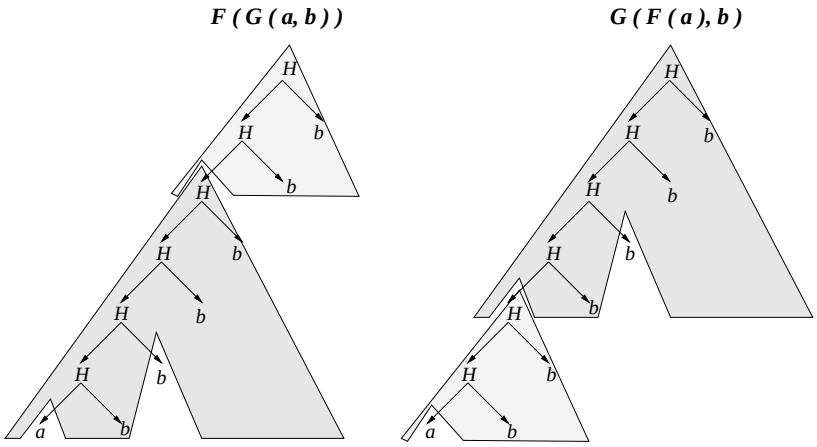


Figure 3.1: Solution of the n-ary variables equation $F(G(a, b)) \stackrel{?}{=} G(F(a), b)$.

We use the *unarise* rule and we enlarge our signature to $\Sigma' = \{a, b, g\}$, where g is a new binary constant. Now, we can guess a partition of $\{1, 2\}$ into two disjoint subsets $\{1\}$ and $\{2\}$, where both are non-empty, and instantiate G by:

$$\tau = [G \mapsto \lambda x_1, x_2.\ G_0(g(G_1(x_1), G_2(x_2)))]$$

We obtain a new problem:

$$F(G_0(g(G_1(a), G_2(b)))) \stackrel{?}{=} G_0(g(G_1(F(a)), G_2(b))) \tag{3.3}$$

which is also solvable (see Figure 3.2), and only contains (unary) context variables.

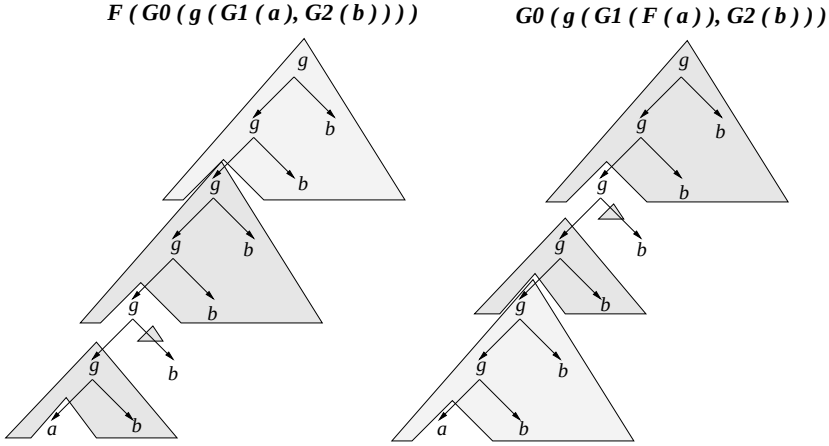


Figure 3.2: Solution of the unary variables equation $F(G_0(g(G_1(a), G_2(b)))) \stackrel{?}{=} G_0(g(G_1(F(a)), G_2(b)))$.

■

Theorem 3.19 *[Levy and Villaret (2000)] The Context Unification Problem where n-ary variables are allowed, is NP-reducible to Context Unification where all context variables are unary.*

Proof: We prove that a context unification problem S with n-ary context variables is unifiable if and only if there exists a translated context unification problem S' (where n-ary context variables with $n > 1$ have been removed by the described method) that is unifiable.

($\Rightarrow$) Let σ be a context unifier of S. Substitution σ shows us how parameters are split into the instantiation of any n-ary context variable. It is not difficult to prove that any splitting can be conjectured by the translation method. Therefore the corresponding context unification problem S', where context variables are unary, can be constructed and one can also construct a context substitution σ' that solves S' from σ.

($\Leftarrow$) It is easy to see that polynomially many steps of this non-deterministic rule, transform an n-ary context unification problem S, into another one S' where all the context variables are unary, that is a context unification problem where context variables are unary. Moreover, since S' is an instance of S, if S' is solvable then, S is also solvable. ■

3.2.4 Historical Notes

Context Unification was introduced by Comon (1992a, 1998) who studied this problem to solve membership constraints. He proves that Context Unification is decidable when any occurrence of the same context variable is always applied to the same term. Levy (1996) also proves decidability of Linear Second-Order Unification under the same condition applied to second-order variables. This condition is close to the linearity restriction on the occurrences of second-order variables (Dowek, 1993), described in Subsection 2.4.4.

It is easy to see that any Context Unification equation system is also a Linear Second-Order Unification equation system. Hence Levy's semi-decision procedure also applies to Context Unification. On the other hand, the presence of "*bindings*" in Linear Second-Order Unification seems to increase the expressivity of the problem and it is still not known if this difference could imply a distinct decidability nature between both problems. We will discuss this in Chapter 6.

Context Unification has applications in rewriting (Comon, 1992a, 1998; Levy and Agustí, 1996; Niehren *et al.*, 1997a, 2000), in unification theory (Schmidt-Schauß, 1996, 1998), and in computational linguistics (Egg *et al.*, 2001; Erk *et al.*, 2002; Koller, 1998; Niehren *et al.*, 1997b; Niehren and Villaret, 2003). Context Unification decidability is unknown but several decidable fragments have been found.

3.3 Known Decidable Fragments of Context Unification

As we have already said, when all occurrences of the same context variable are always applied to the same term, also known as *Comon's restricted case*, Context Unification is decidable (Comon, 1992a, 1998). Moreover, Levy and Villaret (1998) prove that it in NP-complete. This result constrasts with the fact that Second-Order Unification under the same restriction on second-order variables has linear time complexity (Levy and Villaret, 1998). Obviously, the decidable fragment of Linear Second-Order Unification, of *at most two occurrences per second-order variable* (Levy, 1996), also applies to Context Unification. But there are still more known decidable fragments in Context Unification.

3.3.1 Word Unification

When the signature considered for the terms is restricted to be at most unary, we are dealing with the well known decidable problem of *Word Unification*.

Decidability of Word Unification, also known as String Unification and as A-Unification, was an open problem for many years. The problem was raised by Markov in the late 1950s who hoped to prove the undecidability of Hilbert's tenth problem by showing undecidability of the Word Unification problem. In this context, Matiyasevich in 1968 gave a simple decision procedure for Word Unification problems where each variable occurs at most twice. Later, J. I.

Hmelevskii in 1971, proved decidability of Word Unification where there are three variables with an arbitrary number of occurrences. The general case of Word Unification was proved to be decidable by Makanin (1977).

Word Unification is a very important problem on his own. The close relation with Context Unification suggests to us presenting it as a fragment of Context Unification where constant symbols and variables are at most unary.

Example 3.20 Let a, b be constants and X, Y word variables, then, the following word unification equation:

$$abXY \stackrel{?}{=} YabX$$

with unifier $\sigma = [X \mapsto ab, Y \mapsto ab]$, can be represented as this A-unification equation[1] where $\cdot$, is an associative symbol:

$$a \cdot b \cdot X \cdot Y \stackrel{?}{=} Y \cdot a \cdot b \cdot X$$

The previous equation can also be represented as the following context unification equation where functions and variables are unary, but where we add an special constant: $\sharp$, that denotes the end of the word:

$$a(b(X(Y(\sharp)))) \stackrel{?}{=} Y(a(b(X(\sharp))))$$

■

Makanin's algorithm was not thought to be implemented but rather to prove decidability of Word Unification. Several authors have tried to simplify it and obtain implementable versions like Jaffar (1990); Schulz (1993), who also give an algorithm which complexity is in 4-NEXPTIME class (i.e. composition of four exponential functions). Then Kościelski and Pacholski (1995, 1996) improved the result and obtain an algorithm that is in 3-NEXPTIME class.

All these algorithms are based on the existence of a theorem that ensures that whenever a word unification equation system is solvable, then there exists a solution σ that has an *exponent of periodicity* bounded by the size of the equations.

Definition 3.21 *An* exponent of periodicity *of a word w is a maximal number $p(w)$ such that $u^{p(w)}$, for a nonempty word u, is a subword of w. An exponent of periodicity $p(\sigma)$ of a solution σ of a word equation $u \stackrel{?}{=} v$ is $p(\sigma(u))$.*

Proposition 3.22 *[Kościelski and Pacholski (1996)] There is a constant c such that for each minimal solution σ of a word equation e, $p(\sigma) \leq 2^{c|e|}$.*

Recently the complexity of the problem has been improved. Gutiérrez (1998) shows that the problem is in EXPSPACE class, and then in Gutiérrez (2000), that it is in PSPACE class. Plandowski (1999a), with an alternative to Makanin's

[1] In the translation, variables that will be mapped to the empty word, must be guessed a priori, and removed in the resulting translated equation.

proof, proves that the problem is in NEXPTIME class an then, in Plandowski (1999b), that it is in PSPACE class. Up to now these are the best known complexity classes for Word Unification. Nevertheless, some people think that Word Unification is in NP.

An important extension of Word Unification is *Word Unification with Regular Constraints*. These constraints enforce instances of word variables in the solutions to belong to desired regular languages. Schulz was the first in proving its decidability (Schulz, 1991). This result for words suggested to us studying the corresponding extension for trees in Context Unification (see Chapter 6 and Chapter 7).

Decidability of Word Unification has been used several times to prove decidability of some fragments of Context Unifications, also, our attempt to prove decidability of full Context Unification relies on the decidability of Word Unification. Schmidt-Schauß and Schulz have also used several ideas of the proof of Word Unification decidability to obtain important results on Context Unification. One of these results is the existence of an *Exponent of Periodicity Lemma* for Context Unification (Schmidt-Schauß and Schulz, 1998).

One of the fragments shown decidable thanks to Word Unification (although an alternative proof that does not rely on Word Unification decidability has been found) is *Stratified Context Unification*.

3.3.2 Stratified Context Unification

Stratified Context Unification is a fragment of Context Unification where terms considered must be *second-order stratified terms*. A term is said to be second-order stratified if for any first and second-order (context) variable V, the sequence of second-order (context) variables from the root of the term to any occurrence of V is always the same.

Example 3.23 The following term:

$$f(F(G(H(a)),b),F(h(G(b)),b))$$

is second-order stratified because all occurrences of variable G have always an F above them, and all occurrences of variable F do not have any variable above them. Variable H also fulfils the stratified condition because it occurs only once. On the other hand, the following term:

$$F(F(a,b),b)$$

is not a stratified term because the outermost occurrence of F has no variable above it while the innermost one has one F above it. ■

Schmidt-Schauß (1996, 1998, 1999b, 2002), proves decidability of Distributive Unification. His proof is based on Stratified Context Unification decidability. Originally the proof of Stratified Context Unification decidability was based on the decidability of Word Unification but a later work avoids the call to Makanin's

decision algorithm. The algorithm given by Schmidt-Schauß provides a terminating strategy to solve *cycles* between equations. Cycles in equations establish a *cyclic dependence between variable instantiations*, for instance, the following set of equations:

$$\{F(X) \stackrel{?}{=} f(G(a), b),\ G(Y) \stackrel{?}{=} g(F(a), b)\}$$

forms a cyclic set of equations because the instantiation of F depends on the instantiation of G which, in its turn, depends on the instantiation of F. Notice that the same situation in First-Order Unification would produce an *occur-check failure*, but Context Variables can be projected. Therefore, it is required that at least one of the variables in the cycle is a context variable.

The termination ordering is of course dependent of the stratifiedness property and does not seem easy to extend to the full Context Unification. The exponent of periodicity bounds for context equations (Schmidt-Schauß and Schulz, 1998), is also used in the proof. Further work on this fragment proves that the problem is in PSPACE class, (Schmidt-Schauß, 2001). The proof requires the use of a compression technique based on the use of powers to represent iterations of contexts.

Decidability of Stratified Context Unification is not only relevant in Unification theory but also in rewriting (Niehren *et al.*, 2000) and in computational linguistics (Niehren *et al.*, 1997b) because it subsumes the *Dominance Constraints* language. This language is used to represent scope underspecification (see Chapter 7).

3.3.3 The Two Distinct Context Variables Fragment

Another decidable fragment of Context Unification also relaying on the decidability of Word Unification is the one where there are at most two distinct context variables and an undefined number of the first-order ones (Schmidt-Schauß and Schulz, 1999, 2002b). In these works, Schmidt-Schauß and Schulz translate context equations into *generalised context equations* guessing the parts of the original terms that must coincide to solve the equation. Then there is a translation from this generalised context equations to word equations with regular constraints which, as we have already said, is decidable. Unfortunately, termination of this second translation relies on the existence of just two distinct context variables.

The decidability of this Context Unification problem contrasts with the undecidability of Second-Order Unification where there is just one second-order variable and it is unary (Levy, 1998; Levy and Veanes, 1998, 2000).

The decidability of the equivalent Linear Second-Order Unification fragment is still not known.

3.4 Bounded Second-Order Unification

Bounded Second-Order Unification is a variant of Second-Order Unification. It is Second-Order Unification under the restriction that only a bounded number of bound variables is allowed in the instantiating terms for second-order variables. However, the size of the instantiation is not restricted. This variant is introduced and proved decidable by Schmidt-Schauß (1999a, 2004). It is the first non-trivial decidable variant of Second-Order Unification where there are no restrictions on the occurrences of variables nor in the shape of the terms. Terms of the equations are the terms defined in Definition 3.9.

The first step of Schmidt-Schauß 's proof is to reduce Bounded Second-Order Unification to Z-Context Unification, where second-order variables are unary and the number of occurrences of every bound variable, in an instantiation of a second-order variable, may be zero or one. Then he proposes an algorithm that resembles the one of Stratified Context Unification because both deal with cycles in a similar way. Nevertheless, there is an important difference, in Z-Context Unification, when all equations are flexible-flexible, we can say that these are in a presolved form because we can solve these equations like it is done in the *preunification method* from (Huet, 1975) for Second-Order Unification (see Subsection 2.3.1).

It is easy to see that the decidability of Context Unification would trivially imply the decidability of Z-Context, and hence, of Bounded Second-Order Unification: notice that we only would need to guess a priori what variables are not going to "use" their argument and then we could simply replace them by first-order variables. On the other hand, the decision algorithm for Bounded Second-Order Unification cannot be trivially used as a decision algorithm for Context Unification, since context variables must "use" their argument, and thus the presolved forms for Z-Context Unification do not need to be solvable when considering Context Unification equations.

Another unification problem related with Bounded Second-Order Unification is Monadic Second-Order Unification. In fact, Monadic Second-Order Unification is simply Bounded Second-Order Unification where the signature is restricted to be at most unary, thus Schmidt-Schauß provides an alternative method for solving Monadic Second-Order Unification. In fact, Schmidt-Schauß (1999a, 2004) also shows that Bounded Second-Order Unification and Monadic Second-Order Unification are NP-hard.

Recently, it has also been proved that when variables are not restricted to be second-order but higher, and the number of lambdas in the unifiers is also bounded, the problem, called Bounded Higher-Order Unification, is decidable (Schmidt-Schauß and Schulz, 2002a).

3.5 Linear Second-Order, Linear Higher-Order and Context Matching

Second-Order Matching is decidable, thus, Linear Second-Order and Context Matching are also decidable. There are several results on the complexity of these problems and other variants of Higher-Order Matching:

- Schmidt-Schauß and Schulz (1998), show that Context Matching is NP-complete, by reducing *1-in-3-SAT* to Context Matching.

- In (de Groote, 2000) it is shown the NP-completeness of Linear Higher-Order Matching, i.e. Higher-Order Matching restricted to the set of linear λ-terms. The proof relies on the fact that in each β-reduction, a certain notion of *size* decreases thanks to linearity, then, a polynomial bound on the *size* of the solutions can be given and hence, solutions could be enumerated and tested in non-deterministic polynomial time.

- Schmidt-Schauß and Stuber (2002), study the complexity of some restrictions on Context Matching like, stratifiedness and the *at most two occurrences per variable* restriction, which are NP-complete, and the Comon's restriction that is shown to be in P. In that paper they also illustrate the possible applications of Context Matching in information extraction from XML documents.

- Dougherty and Wierzbicki (2002), generalise the result of de Groote (2000), and show that Higher-Order Matching where instances of variables are allowed to contain a bounded number of bound variable occurrences, is decidable.

- Salvati and de Groote (2003), show that Linear Second-Order Matching under Comon's restriction is NP-complete, contrasting with the fact that Context Matching under the same restriction is in P (Schmidt-Schauß and Stuber, 2002). The main reason for this difference is that the order in which the arguments of second-order variables are abstracted in Linear Second-Order Matching, does not need to correspond to the order in which these variables occur in the body of the term. In some sense, this difference between Linear Second-Order and Context Matching, suggests that some significant complexity difference must exist between Linear Second-Order and Context Unification.

Unfortunately, none of these results sheds any light on the general Higher-Order Matching case.

3.6 About Linear Second-Order and Context Unification Decidability

Goldfarb's reduction of the 10th Hilbert's Problem to Second-Order Unification (Goldfarb, 1981) (see Section 2.2), does not apply to Context Unification nor to Linear Second-Order Unification. The linearity condition of the solutions for these last two problems forbids a naive reutilisation of that reduction.

As we have already said, the most extended belief is that Context and Linear Second-Order Unification are both decidable. There are some clues that support this belief.

On the one hand, the similarities between Word Unification and Context Unification, suggest that either

- an adaptation of the decidability proof of Word Unification could be made for Context Unification; in this sense the existence of a bound on the exponent of periodicity for Context Unification is a very important step,
- or some kind of reduction from Context Unification to Word Unification could be done. Again the work of Schmidt-Schauß and Schulz (1999, 2002b) showing that the fragment of the two distinct context variables can be reduced to Word Unification, is important in supporting this possibility (see Section 3.3); as well as our work (Levy and Villaret, 2001), where we present a reduction from Context Unification to Word Unification plus Regular Constraints that depends on the *rank bound* conjecture (see Chapter 5).

On the other hand, the fact that some Context Unification fragments like the Stratified one or the one of two distinct context variables, which when considered as Second-Order Unification are undecidable (Levy, 1998; Levy and Veanes, 1998, 2000; Schubert, 1998) (see Section 2.2), and when considered as Context Unification become decidable (Schmidt-Schauß, 1996, 1998, 1999b, 2002; Schmidt-Schauß and Schulz, 1999, 2002b) (see Section 3.3), also supports this generalised belief.

Concerning the possibility that one of both problems were decidable and the other undecidable, only makes sense in one direction, (because as we have already said, Linear Second-Order Unification subsumes Context Unification). We think that this should not be the case, as we will argue in Chapter 6.

3.7 Summary

In this chapter we have introduced the main topic of this thesis: Linear Second-Order Unification and Context Unification, two variants of Second-Order Unification that enforce linearity in instantiations of second-order or context variables. We have argued that permitting the use of λ-abstractions and bound variables in the first one but not in the second one, is the main difference between both problems.

We have also discussed on the two "approaches" to Context Unification:

- as an extension of First-Order Unification by means of allowing variables that denote contexts, i.e. terms with a hole,
- or as a restriction of Linear Second-Order Unification where neither λ-bindings nor third-order constants are allowed to occur in the equations.

One of the differences is the arity of the variables: while in the first approach context variables are just unary, in the second approach context variables are allowed to be of any arity. We have shown that this difference is not significant in terms of decidability, but it has some consequences with respect to solvability depending on the signature that we consider.

We have also enumerated the several known decidable fragments of Context and Linear Second-Order Unification, and presented the main results about the Matching Problem.

Finally we have argued why we think that that Context Unification and Linear Second-Order Unification can be decidable.

Chapter 4

Currying Second-Order Unification Problems

In this chapter we show how the signature for Second-Order Unification and Context Unification can be simplified to contain only one binary function symbol and first-order constants.

This result shows us that in fact the importance of the signature, in terms of decidability, lies in the difference between having at most unary constant symbols (Monadic Second-Order Unification) or having at least a binary symbol that allows branching. Apart from this theoretical reading of the result, this simplification of the signature applied to the results of Levy (1998), allows us to draw the frontier between decidability and undecidability of Second-Order Unification problems more precisely. On the other hand, it also helps us to simplify the study of Context Unification.

The work presented in this chapter is basically based on (Levy and Villaret, 2002).

4.1 Introduction

The Curry form of a term, like $f(a, b)$, allows us to write it, using just a single binary symbol, as $@(@(f, a), b)$, where @ denotes the explicit application. This helps to solve unification problems. In first-order logic, this transformation reduces a unification equation system to another one containing a single binary symbol. The size of the new equation system, and of the unifier, is similar to the size of the original equation system, and of the original unifier. So, from the point of view of complexity there is not a significant difference, but in practical implementations this allows us representing terms as binary trees, and contexts as subterms, and this has been used in term indexing data structures (Ganzinger *et al.*, 2001).

In second-order logic the transformation is not so obvious. We can currify constant symbol applications and second-order variable applications, obtaining

a first-order term. For instance, for $f(F(a), Y)$, where F is a second-order variable, we obtain $@(@(f, @(F, a)), Y)$, where both X and Y are now first-order variables. However, solvability of unification equation systems is not preserved by such transformation, unless we consider some form of First-Order Unification modulo β-reduction for solving the new problem. For instance, the Second-Order Unification equation $F(G(a), b) \stackrel{?}{=} g(a)$ is solvable, whereas its first-order Curry form $@(@(F, @(G, a)), b) \stackrel{?}{=} @(g, a)$ is unsolvable. Moreover, the right-hand side of the β-equivalence $(\lambda x.\ t_1)t_2 = t_1[t_2/x]$ is a meta-term, unless we make substitution explicit (Abadi *et al.*, 1998). Roughly speaking this is what is done in the so called *Explicit Unification* (Dowek *et al.*, 2000; Bjorner and Muñoz, 2000).

Here, we propose to currify function symbol applications, but not variable applications. Therefore, the new equation system we get is also a second-order unification equation system. For instance, for $F(G(a), b) \stackrel{?}{=} g(a)$, we get $F(G(a), b) \stackrel{?}{=} @(g, a)$, that is also solvable. In this case, we do not reduce the order of the unification equation system, but we reduce the number of function symbols to just one: the application symbol @. It can be argued that this reduction is useless, since Second-Order Unification (Goldfarb, 1981) was already known to be undecidable for just one binary function symbol (Farmer, 1991), although applying the reduction to the results of Levy and Veanes (1998), proves that Second-Order Unification is undecidable for one binary function symbol and one second-order variable occurring four times. Moreover, the same reduction is applicable to Context Unification, for which decidability is still unknown and it allows us concentrating the efforts in a very simple signature. We also think that currying could help to simplify the signature used in Higher-Order Matching, and this could help to prove its decidability (or undecidability).

If we currify function applications in a second-order (or context) unification equations system, it is easy to prove that, if the original equations system is solvable, then its Curry form is also solvable: we can currify the unifier of the original equation system to obtain a unifier of its Curry form. However, the converse is not true and, in general, solvability is not preserved by currying, as the following examples prove.

Example 4.1 The following Context Unification equation

$$\begin{array}{c} g(\ F(G(a)),\ F(a),\quad G(a)\quad) \stackrel{?}{=} \\ \stackrel{?}{=} g(\ f(a,b),\quad H(a,b),\ H(X,a)\ ) \end{array}$$

is unsolvable. However, its Curry form

$$\begin{array}{c} @(@(@(g,\ F(G(a))\quad),\ F(a)\quad),\ G(a)\quad) \stackrel{?}{=} \\ \stackrel{?}{=} @(@(@(g,\ @(@(f,a),b)\ ),\ H(a,b)\ ),\ H(X,a)\ ) \end{array}$$

is solvable and has the following unifier

$$\begin{array}{l} \sigma(F) = \lambda x.\ @(x, b) \\ \sigma(G) = \lambda x.\ @(f, x) \\ \sigma(H) = \lambda xy.\ @(x, y) \\ \sigma(X) = f \end{array}$$

Similarly, the following Second-Order Unification equation

$$\begin{array}{l} g(\ F(G(a)), F(G(a')), F(a), \quad F(a'), \quad G(a), \quad G(a')\quad) \stackrel{?}{=} \\ \stackrel{?}{=} g(\ f(a,b), \quad f(a',b), \quad H(a,b), H(a',b), H(X,a), H(X,a')\ ) \end{array}$$

is also unsolvable, whereas its Curry form is solvable. ■

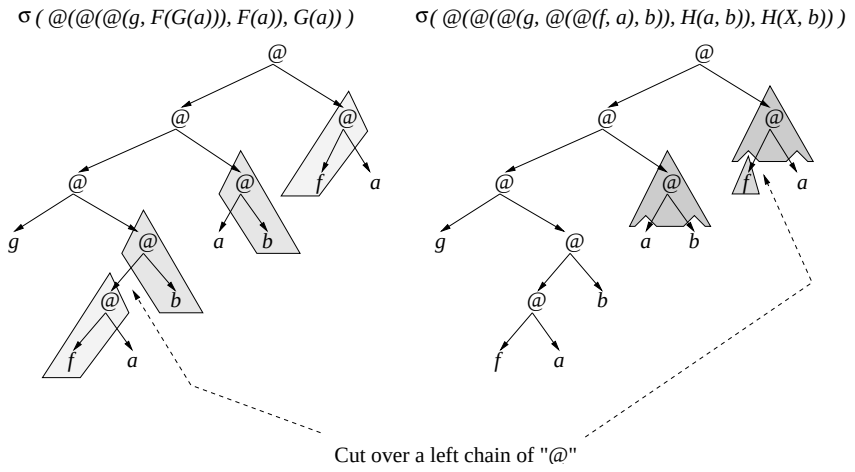


Figure 4.1: Common instance of the curried context unification equation of Example 4.1.

In the previous example, $\sigma(F)$, $\sigma(G)$, $\sigma(H)$ and $\sigma(X)$ are not "well-typed", i.e. they are not the Curry form of any well-typed term. For instance, $\sigma(F) = \lambda x.\ @(x, b)$ is the Curry form of $\lambda x.\ x(b)$, but this term is third-order typed (and F is a second-order typed variable), and $\sigma(G) = \lambda x.\ @(f, x)$ is the Curry form of $\lambda x.\ f(x)$, but f has two arguments. This disallows us to reconstruct a unifier for the original equation system from the unifier we get for its Curry form.

We can also see that the original unification equations contain variables that "touch". For instance, F touches G in $F(G(a))$, and H touches X in $H(X, a)$. We will prove, for Second-Order and for Context Unification, that, if no variable touches any other variable, then solvability of the equations is preserved in both directions by our partial currying. It is easy to reduce Second-Order and Context Unification equation systems to equation systems accomplishing such property. Therefore, we conclude that Second-Order and Context Unification can be both reduced to the partial Curry form, where only a binary function symbol @ is used.

In Subsection 3.3.1, we have already introduced Word Unification as an special case of Context Unification. Plandowski (1999b), proves that if σ is a most general unifier of a Word Unification equation $t \stackrel{?}{=} u$, then any substring of $\sigma(t)$ "is over a cut", i.e. there exists an occurrence of the substring in $\sigma(t)$ that is not

completely inside the instance of a variable. Something similar can be proved for Second-Order and for Context Unification. The pathology of Example 4.1 is due to the existence of a cut over a left chain of @ ended by a constant. For instance, in the example, the left chain $@(@(f,\ldots),\ldots)$ is "cut" by $F(G(\ldots))$, i.e. one piece is inside $\sigma(F)$ and another inside $\sigma(G)$ (see Figure 4.1). If variables "do not touch" this situation is avoided, and satisfiability is preserved. Our main result could be proved using a version of Plandowski's theorem for Second-Order Unification, but the proof would be longer than the one we present here.

This chapter proceeds as follows. In Section 4.2 we introduce some assumptions and considerations of the chapter. Most of our results hold for Second-Order and for Context Unification, and sometimes we do not make the distinction explicit. In Section 4.3 we define the partial Curry forms where only function symbol applications are made explicit. In Section 4.4 we define a labeling on Curry forms that is used to characterise "well-typed" terms, i.e. terms that are the Curry form of some well-built term. In Section 4.5 we prove our main result: Second-Order and Context Unification can be reduced to a simplified form where only a single binary function symbol and unary constants are used. We conclude in Section 4.6 with a discussion about the difficulties to extended these results to Higher-Order Matching.

4.2 Preliminary Definitions

The terms we consider are first-order typed λ-terms over a restricted second-order signature (see Definition 3.13). If nothing is said, the signature of a problem is given by the set of constants that it contains and a denumerable infinite set of variables, for every arity. For technical reasons we also assume that the signature contains, at least, a binary function symbol and a constant (that can be added if the problem does not contain any).

The following is a basic property of most general second-order [and context] unifiers that will be required in some proofs. It ensures that the signature does not play an important role with respect to the decidability of the problem.

Property 4.2 *Let $t \stackrel{?}{=} u$ be a second-order or a context unification problem, and σ be a most general unifier. Then, for any variable X, $\sigma(X)$ does not contain constants not occurring in the problem $t \stackrel{?}{=} u$.*

Proof: Suppose that a most general unifier σ introduces a constant f not occurring in the problem. Then we can replace this constant by a fresh variable F of the same arity everywhere and get another unifier that is more general than σ (we can instantiate F by $\lambda x_1 \ldots x_n.\ f(x_1,\ldots,x_n)$, but not vice versa). This contradicts the fact that σ is most general. ■

Notice that this property is true for Context Unification thanks to the fact that we allow n-ary context variables (see Subsection 3.2.3), otherwise it would not be true, as Example 3.16 shows.

4.3 Currying Terms

Definition 4.3 *Given a second-order signature* $\Sigma = \bigcup_{n\geq 0} \Sigma_n$*, the* curried signature $\Sigma^c = \bigcup_{n\geq 0} \Sigma_n^c$ *is defined by*

$$\begin{aligned}
&\Sigma_0^c = \textstyle\bigcup_{n\geq 0} \Sigma_n \\
&\Sigma_2^c = \{@\} \\
&\Sigma_n^c = \emptyset \qquad\qquad \text{for } n \neq 0, 2
\end{aligned}$$

The currying function $\mathcal{C} : \mathcal{T}(\Sigma, \mathcal{X}) \to \mathcal{T}(\Sigma^c, \mathcal{X})$ *is defined recursively as follows:*

$$\begin{aligned}
&\mathcal{C}(a) = a \\
&\mathcal{C}(x) = x \\
&\mathcal{C}(f(t_1, \ldots, t_n)) = @(\overset{n}{\cdots} @(f, \mathcal{C}(t_1)) \overset{n}{\cdots}, \mathcal{C}(t_n)) \\
&\mathcal{C}(F(t_1, \ldots, t_n)) = F(\mathcal{C}(t_1), \ldots, \mathcal{C}(t_n)) \\
&\mathcal{C}(\lambda x\,.\,t) = \lambda x\,.\,\mathcal{C}(t)
\end{aligned}$$

for any constant $a \in \Sigma_0$*, bound variable* x*, function symbol* $f \in \Sigma_n$*, and variable* $F \in \mathcal{X}_n$*.*

The currying function is injective, but it is not onto, as suggested by the following definition.

Definition 4.4 *Given a term* $t \in \mathcal{T}(\Sigma^c, \mathcal{X})$*, we say that it is* well-typed *(w.r.t.* Σ*), if* $\mathcal{C}^{-1}(t)$ *is defined, i.e. if there exists a term* $u \in \mathcal{T}(\Sigma, \mathcal{X})$ *such that* $\mathcal{C}(u) = t$*.*

Lemma 4.5 *If the second-order [context] unification problem* $t \stackrel{?}{=} u$ *over* Σ *is solvable, then the second-order [context] unification problem* $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ *over* Σ^c *is also solvable.*

Proof: Let σ be a unifier of $t \stackrel{?}{=} u$, then it is easy to prove that the substitution $\sigma_{\mathcal{C}}$ defined as, for each variable $F \in Dom(\sigma)$, $\sigma_{\mathcal{C}}(F) = \mathcal{C}(\sigma(F))$, is a unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$. ■

In fact, we have proved a stronger result: given a unifier σ of $t \stackrel{?}{=} u$, we can find a unifier $\sigma_{\mathcal{C}}$ of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ that satisfies the commutativity property $\mathcal{C}(\sigma(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(t))$. This commutativity property is represented by the following diagram:

$$\begin{array}{ccc}
t \stackrel{?}{=} u & \xrightarrow{\mathcal{C}} & \mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u) \\
\Big\downarrow \sigma & \Longrightarrow^{\mathcal{C}} & \Big\downarrow \sigma_{\mathcal{C}} \\
\sigma(t) & \xrightarrow{\mathcal{C}} & \sigma_{\mathcal{C}}(\mathcal{C}(t))
\end{array}$$

Unfortunately, as it is shown in Example 4.1, the converse is not true. Given a unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ it is not always possible to obtain a unifier of $t \stackrel{?}{=} u$. In the next Section, we describe sufficient conditions to ensure that the inverse construction is possible.

4.4 Labeling Terms

The first step to find a sufficient condition ensuring that the currying function preserves satisfiability is to characterise well-typed curried terms. This is done by labeling application symbols @ with the "arity" of their left argument, and using a "hat"to mark the roots of right arguments. If left arguments always have positive arity, and right arguments always have arity zero, then the term is well-typed.

Definition 4.6 *Given a signature* $\Sigma = \bigcup_{n\geq 0} \Sigma_n$*, the* labeled signature $\Sigma^L = \bigcup_{n\geq 0} \Sigma_n^L$ *is defined by:*

$$\begin{aligned} \Sigma_0^L &= \textstyle\bigcup_{n\geq 0} \Sigma_n \\ \Sigma_2^L &= \{@^l, \widehat{@^l} \mid l \in \{\ldots, -1, 0, 1, \ldots\}\} \\ \Sigma_n^L &= \emptyset \qquad\qquad \text{for } n \neq 0, 2 \end{aligned}$$

The labeling functions $\mathcal{L}, \widehat{\mathcal{L}} : \mathcal{T}(\Sigma^c, \mathcal{X}) \to \mathcal{T}(\Sigma^L, \mathcal{X})$ *are defined by the following rules:*

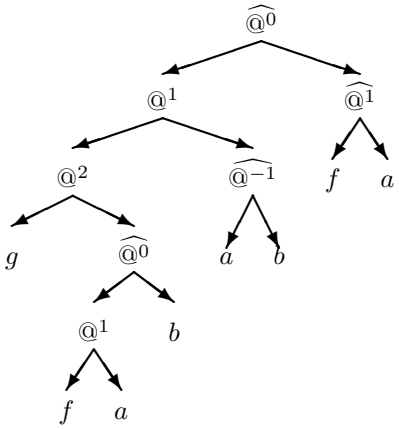
1. *If the left child of an @ is an n-ary symbol* $f \in \Sigma_n$*, then it has label* $l = arity(f) - 1 = n - 1$*.*
2. *If the left child of an @ is a variable* $X \in \mathcal{X}$*, or a bound variable, then it has label* -1*, regardless what the arity of the variable is.*
3. *If the left child of an @ is another @ with label n, then it has label* $n - 1$*.*

In the case of $\widehat{\mathcal{L}}$ *we also use the following rule:*

4. *If an @ is the right child of another @, or it is the child of a variable, or it is the root of the term, then, apart from the label, it also has a hat.*

Example 4.7 The $\widehat{\mathcal{L}}$-labeling of the term $\sigma(\mathcal{C}(t))$, used in Example 4.1 and shown in Figure 4.1, is as follows.

$$\widehat{@^0}(@^1(@^2(g, \widehat{@^0}(@^1(f, a), b)), \widehat{@^{-1}}(a, b)), \widehat{@^1}(f, a))$$

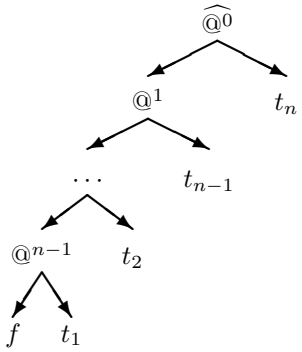


Notice that labels can be negative numbers. These negative labels do not appear in labelings of "well-typed" terms. ■

Based on these labels, it is easy to characterise well-typed terms.

Lemma 4.8 *A term $t \in \mathcal{T}(\Sigma^c, \mathcal{X})$ is well-typed if, and only if, $\widehat{\mathcal{L}}(t)$ does not contain application symbols with negative labels ($@^{-n}$, for $n > 0$) or with hat and non-zero labels ($\widehat{@^n}$ with $n \neq 0$).*

Proof: The "only if" implication is obvious. For the "if" implication, assume that the labeling $\widehat{\mathcal{L}}(t)$ does not contain $@^{-n}$, with $n > 0$, or $\widehat{@^n}$, with $n \neq 0$. Then, any @ symbol is in a sequence of the form:



where the node $\widehat{@^0}$ is a right child of another @, or it is the child of a variable F, or it is the root of the term. We can prove that this is the currying of $f(\mathcal{C}^{-1}(t_1), \ldots, \mathcal{C}^{-1}(t_n))$ which is a well constructed term, because f has n arguments and arity n. ■

4.5 When Variables do not Touch

In this section, we try to find sufficient conditions ensuring that, when we have a unifier for $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, we can find a unifier for $t \stackrel{?}{=} u$. The strategy to prove this result is summarised in the following diagram:

$$\begin{array}{ccccc}
t \stackrel{?}{=} u & \xrightarrow{\mathcal{C}} & \mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u) & \xrightarrow{\widehat{\mathcal{L}}} & \widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u)) \\
\Big\downarrow \sigma & \xLeftarrow{\mathcal{C}^{-1}} & \Big\downarrow \sigma_{\mathcal{C}} & \Longrightarrow^{\widehat{\mathcal{L}}} & \Big\downarrow \sigma_{\widehat{\mathcal{L}}} \\
\sigma(t) & \xleftarrow{\mathcal{C}^{-1}} & \sigma_{\mathcal{C}}(\mathcal{C}(t)) & \xrightarrow{\widehat{\mathcal{L}}} & \sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t)))
\end{array}$$

We will find a condition that makes the right square commute (Lemma 4.12). Then we will prove that when the right square commutes, then the left one also

commutes (Lemma 4.13). This second commutativity property ensures that the currying transformation preserves satisfiability.

The sufficient condition we have found is based on the following definition.

Definition 4.9 *Given a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$, we say that two variables $F, G \in \mathcal{X}$* touch, *if t contains a subterm of the form $F(t_1, \ldots, G(u_1, \ldots, u_m), \ldots, t_n)$.*

In the context unification problem of Example 4.1, the variable F touches G, and the variable H touches X.

For technical reasons, before proving that the first square commutes (see Lemma 4.12 below) we prove the same result using a variant of the labeling function where hats are not considered (notice that in Lemma 4.10, the labeling function $\mathcal{L}$ has no hats).

Lemma 4.10 *If the variables of $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then the substitution $\sigma_{\mathcal{L}}$ defined as follows: for each variable $F \in Dom(\sigma_{\mathcal{C}})$*

$$\sigma_{\mathcal{L}}(F) = \mathcal{L}(\sigma_{\mathcal{C}}(F))$$

is a most general unifier of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$, and satisfies

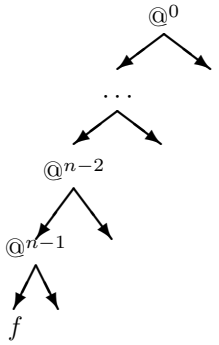
$$\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t))) = \mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$$

Proof: First, we prove that

$$\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t))) = \mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$$

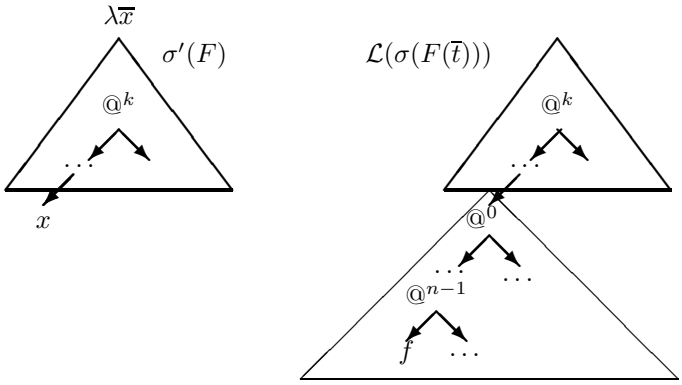
As far as $\sigma_{\mathcal{C}}$ and $\sigma_{\mathcal{L}}$ only differ in the introduction of labels, both terms have the same form, except for the labels. Therefore, we only have to compare the labels of the corresponding @'s in both terms. There are two cases:

- If the occurrence of the @ is outside the instance of any variable, then this @ already occurs in $\mathcal{C}(t)$, and it is in a sequence of the form:



  where the f and all the @'s in between, already occur in $\mathcal{C}(t)$ (they have not been introduced by an instantiation either). Thus, the @ gets the same label in $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(t)))$ as in $\mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$, because this label only depends on the left descendants, and they have not been introduced by $\sigma_{\mathcal{C}}$ or $\sigma_{\mathcal{L}}$.

- If the @ is inside the instance of a variable F, we have to prove that it gets the same label in $\sigma_{\mathcal{L}}(\mathcal{L}(F(t_1,\ldots,t_n))) = \sigma_{\mathcal{L}}(F)(\sigma_{\mathcal{L}}(\mathcal{L}(t_1)),\ldots,\sigma_{\mathcal{L}}(\mathcal{L}(t_1)))$ as in $\mathcal{L}(\sigma_{\mathcal{C}}(F(t_1,\ldots,t_n)))$. In the first case we label $\sigma_{\mathcal{C}}(F)$ before instantiating (so we have bound variables in the place of the arguments), whereas in the second case we label $\sigma_{\mathcal{C}}(F)$ after instantiating (so we already have the arguments t_i). As we will see, in both cases the labels we get are the same. The root of one of the arguments t_i can be a left descendant of the @, and its label will depend on such argument. However, if variables do not touch, the head of any argument t_i of F is a constant, and the head of $\mathcal{C}(t_i)$ is either a 0-ary constant a or an @ with label 0. Therefore, the labels of the ancestors of the argument inside $\sigma_{\mathcal{C}}(F)$ will be the same if we replace the argument by a bound-variable, and the label of the corresponding @ inside $\sigma_{\mathcal{L}}(F)$ will be the same.



Similarly, we can prove $\sigma_{\mathcal{L}}(\mathcal{L}(\mathcal{C}(u))) = \mathcal{L}(\sigma_{\mathcal{C}}(\mathcal{C}(u)))$. As $\sigma_{\mathcal{C}}(\mathcal{C}(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(u))$, we can conclude that $\sigma_{\mathcal{L}}$ is a unifier of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$.

Given a unifier of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$, we can find a unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$ by removing labels. Using this idea, it is easy to prove that, if $\sigma_{\mathcal{C}}$ is most general for $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then $\sigma_{\mathcal{L}}$ is also most general for $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$. Otherwise, there would be a unifier more general than $\sigma_{\mathcal{L}}$, and removing labels we could obtain a unifier more general than $\sigma_{\mathcal{C}}$. ■

The following is a technical lemma that we need in the proof of Lemma 4.12.

Lemma 4.11 *If the variables of $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then the arguments t_i of any variable F never occur as left child of an @ in $\sigma_{\mathcal{C}}(\mathcal{C}(t))$.*

Proof: As $\mathcal{C}(t)$ and $\mathcal{C}(u)$ are trivially well-typed, by Lemma 4.8, $\mathcal{L}(\mathcal{C}(t))$ and $\mathcal{L}(\mathcal{C}(u))$ will not contain @'s with negative labels. Let $\sigma_{\mathcal{L}}$ be the most general unifier of $\mathcal{L}(\mathcal{C}(t)) \stackrel{?}{=} \mathcal{L}(\mathcal{C}(u))$ given by Lemma 4.10. Now, by Property 4.2, as $\sigma_{\mathcal{L}}$ is a most general unifier, for any variable F, $\sigma_{\mathcal{L}}(F)$ will not contain @'s with negative labels, either. We can conclude then that the head of any argument t_i of F cannot be a left child of an @. As far as the heads of $\sigma_{\mathcal{L}}(t_i)$ have zero label

or are 0-ary constants, this situation would introduce a negative label in some @ inside $\sigma_{\mathcal{L}}(F)$. ■

Lemma 4.12 *If the variables of $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then the substitution $\sigma_{\widehat{\mathcal{L}}}$ defined as follows: for each variable $F \in Dom(\sigma_{\mathcal{C}})$*

$$\sigma_{\widehat{\mathcal{L}}}(F) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(F))$$

is a most general unifier of $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$, and satisfies

$$\sigma_{\widehat{\mathcal{L}}}(\widehat{\mathcal{L}}(\mathcal{C}(t))) = \widehat{\mathcal{L}}(\sigma_{\mathcal{C}}(\mathcal{C}(t)))$$

Proof: We already know that both terms have the same form and the same labels, thus we only have to prove that they have the same hats. Again, there are two cases:

- If the occurrence of the @ is outside the instance of any variable, then the only situation we have to consider is the following. If the @ has as father a variable F in $\mathcal{C}(t)$, and after instantiation, it becomes a left child of an @ inside $\sigma_{\widehat{\mathcal{L}}}(F)$, then it could loose the hat. However, if variables do not touch, this situation is not possible because, by Lemma 4.11, arguments t_i of F never occur as a left child of an @ in $\sigma_{\widehat{\mathcal{L}}}(F(t_1, \ldots, t_n))$.
- If the occurrence of the @ is inside the instance of a variable F, then we have to prove that the fact that @ has a hat or not, does not depend on the arguments of F. This is obvious because this fact does not depend on the descendants of the @. As in Lemma 4.10, this allows us to replace arguments by bound variables and get a unifier $\sigma_{\widehat{\mathcal{L}}}$ for our problem.

Using the argument of Lemma 4.10, we conclude that $\sigma_{\widehat{\mathcal{L}}}$ is a most general unifier of $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$. ■

Lemma 4.13 *If the variables of $t \stackrel{?}{=} u$ do not touch, and $\sigma_{\mathcal{C}}$ is a most general unifier of $\mathcal{C}(t) \stackrel{?}{=} \mathcal{C}(u)$, then there exists a most general unifier σ of $t \stackrel{?}{=} u$ that satisfies*

$$\mathcal{C}(\sigma(t)) = \sigma_{\mathcal{C}}(\mathcal{C}(t))$$

Proof: Let $\sigma_{\widehat{\mathcal{L}}}$ be the most general unifier of the equation $\widehat{\mathcal{L}}(\mathcal{C}(t)) \stackrel{?}{=} \widehat{\mathcal{L}}(\mathcal{C}(u))$ given by Lemma 4.12. As $\mathcal{C}(t)$ and $\mathcal{C}(u)$ are well-typed, by Lemma 4.8, they do not contain negative labels nor hats over non-zero labeled @'s. Then, by Property 4.2, $\sigma_{\widehat{\mathcal{L}}}$ does not introduce such kind of labels or hats. Therefore, as $\sigma_{\widehat{\mathcal{L}}}(F)$ is defined as the labeling of $\sigma_{\mathcal{C}}(F)$, using again Lemma 4.8, $\sigma_{\mathcal{C}}(F)$ will be well-typed, and we can define:

$$\sigma(F) = \mathcal{C}^{-1}(\sigma_{\mathcal{C}}(F))$$

for each variable $F \in Dom(\sigma_{\mathcal{C}})$. ■

Theorem 4.14 *Decidability of Second-Order Unification can be NP-reduced to decidability of Second-Order Unification with just one binary function symbol, and constants.*

Proof: By Lemmas 4.5 and 4.13, we know that, when variables do not touch, satisfiability of second-order problems is preserved by currying. Now, we will prove that we can NP-reduce solvability of Second-Order Unification to solvability of the corresponding problems without touching variables.

For every n-ary variable F of the original unification problem (notice that n can be 0), we conjecture one of the following possibilities:

- Project $F \mapsto \lambda x_1 \dots x_n.\ x_i$, for some $i \in \{1, \dots, n\}$.
- Instantiate $F \mapsto \lambda x_1 \dots x_n.\ f(F_1(x_1, \dots, x_n), \dots, F_m(x_1, \dots, x_n))$, for some constant $f \in \Sigma_m$ occurring in the original unification problem, and being $F_1, \dots, F_m$ fresh free n-ary variables.

Obviously, this reduction can be performed in polynomial non-deterministic time, and the resulting problem satisfies that variables do not touch. As far as the new problem is an instance of the original one, if the new problem is solvable, so the original one is.

If the original problem is solvable, and σ is a most general unifier, then, for every variable F of the original problem, let $\sigma(F) = \lambda x_1 \dots x_n.\ t$ be written in normal form. Taking t as a tree, descend from the root to the left-most leave, discarding free variables, until you get a bound variable x_i, a 0-ary variable or a constant f (this must be a constant occurring in the problem, by Property 4.2). Then the instantiation $F \mapsto \lambda x_1 \dots x_n.\ x_i$, if we find a bound variable x_i, $F \mapsto \lambda x_1 \dots x_n.\ a$ for some fixed constant a, if we find a 0-ary variable, or $F \mapsto \lambda x_1 \dots x_n.\ f(F_1(x_1, \dots, x_n), \dots, F_m(x_1, \dots, x_n))$, if we find a subterm like $f(t_1, \dots, t_m)$, results in a solvable problem that can be constructed using *project* and *instantiate*.

In fact, the solution of the new problem is $\rho \circ \sigma$ where ρ projects the free variables that we have discarded during the traversal by $\lambda x_1 \cdots x_n.\ x_1$, maps the 0-ary free variables that we have found to the fixed constant a and, in case we have instantiated F by $\lambda x_1 \dots x_n.\ f(F_1(x_1, \dots, x_n), \dots, F_m(x_1, \dots, x_n))$, maps F_i to $\lambda x_1 \dots x_n.\ t_i$ (for $i \in [1..m]$). ■

Theorem 4.15 *Decidability of Context Unification can be NP-reduced to decidability of Context Unification with just one binary function symbol, and constants.*

Proof: Again, by Lemmas 4.5 and 4.13, we know that, when variables do not touch, satisfiability of context unification problems is preserved by currying. Now, we will prove that we can NP-reduce solvability of Context Unification to solvability of the corresponding problems without touching variables.

For every n-ary variable F of the original problem, we conjecture one of the following possibilities:

- Project $F \mapsto \lambda x.\ x$, if it is unary.
- Instantiate

$$F \mapsto \lambda x_1 \ldots x_n.\ f(F_1(x_{\tau(1)}, \ldots, x_{\tau(r_1)}), \ldots, F_m(x_{\tau(r_{m-1}+1)}, \ldots, x_{\tau(n)}))$$

  for some constant $f \in \Sigma_m$ occurring in the original unification problem, some permutation τ, and being $F_1, \ldots, F_m$ fresh free variables of appropriate arity.

As for the second-order case, it can be proved that this nondeterministic reduction preserves satisfiability. However, in this case the assumption that the original signature (the problem) contains, at least, a binary function symbol (say h) and a 0-ary constant (say a) is crucial because our proof will consider ground instantiations of unifiers.

As far as the new problem is an instance of the original one, if the new problem is solvable, so is the original one.

Let the original problem S be solvable, and σ be a most general unifier. Let $\sigma' = \rho \circ \sigma$ be a ground unifier, where ρ maps free n-ary variables ($n \geq 1$) in $\sigma(S)$ to $\lambda x_1 \ldots x_n.\ h(x_1, h(x_2, h(\ldots, h(x_n, a)\ldots)))$ and to a if they are 0-ary, for some binary function symbol h and constant a.

Now, guided by substitution σ', we can build a solvable context unification problem where variables do not touch (say S'), by applying project and instantiate rules to S. For every free variable F in S let

- $\sigma'(F) = \lambda x_1 \ldots x_n.\ f(t_1, \ldots, t_m)$ be written in normal form. Then to obtain the new problem S' we instantiate F by:

$$\lambda x_1 \ldots x_n.\ f(F_1(x_{\tau(1)}, \ldots, x_{\tau(r_1)}), \ldots, F_m(x_{\tau(r_{m-1}+1)}, \ldots, x_{\tau(n)}))$$

  where $\{\tau(r_{i-1}+1), \ldots, \tau(r_i)\}$ is the set of indices of variables $x_1, \ldots, x_n$ occurring in t_i, or let
- $\sigma'(F) = \lambda x.\ x$ be written in normal form, then instantiate F by $\lambda x.\ x$.

Then, the substitution that maps the introduced free variables F_i to the terms $\lambda x_{\tau(r_{i-1}+1)} \ldots x_{\tau(r_i)}.\ t_i$, is a solution of S'. ■

Theorem 4.14 together with Corollary 9 of Levy and Veanes (1998) provides us this corollary.

Corollary 4.16 *Second-Order Unification is undecidable for one binary function symbol and one second-order variable occurring four times.*

4.6 About Currying Higher-Order Matching

Decidability of Higher-Order Matching is still an open question. Proving that Higher-Order Matching can be curried, i.e., that we can simplify the signature,

could contribute to prove its decidability or undecidability. The extension of our technique to third-order and higher orders is proposed as a further work.

The first difficulty we find trying to apply our transformation to third or higher order matching problems is that we must deal with instances of variables that are not connected. For instance, the following matching problem:

$$\begin{array}{l} f(\ F(\lambda x.\ g(x), a),\ F(\lambda x.\ g'(x), a')\ ) \stackrel{?}{=} \\ \stackrel{?}{=} f(\ f(g(h(a)), a),\quad f(g'(h(a')), a')\ ) \end{array}$$

is solved by the substitution:

$$F \mapsto \lambda xy.\ f(x(h(y)), y)$$

where the instance of F is split into two pieces f and h. In such situations we have to guarantee that these pieces do not touch, to avoid that these "cuts" (in the sense of Plandowski (1999b)) could cut a left chain of @'s.

4.7 Summary

Currying terms is an standard technique in functional programming and has been used in practical applications of automated deduction. It is also used in Higher-Order Unification via explicit substitutions or Explicit Unification. However, in these cases not only applications, but also lambda abstractions are made explicit, and unification is made modulo the explicit substitution rules.

In this chapter we have proposed a partial currying transformation for Second-Order Unification, where the "order" of the unification problem is not reduced, like in Explicit Unification, but the signature is simplified. The transformation is not trivial, and we have proved that, to preserve solvability of the problems, we need to ensure that "variables do not touch".

This encoding serves, alternatively to Farmer (1991), to prove that Second-Order Unification is undecidable with just one binary function symbol and it also helps us to sharpen the results of Levy and Veanes (1998) (see Corollary 4.16).

The reduction also works for Context Unification. This allows us to concentrate on a simpler signature containing constant symbols and just one binary function symbol: the explicit application symbol @.

We have also discussed on the possibility that this technique could help us in solving the Higher-Order Matching Problem.

Chapter 5

Context Unification and Traversal Equations

This is the core chapter of the thesis. In this chapter we present a non-trivial sufficient and necessary condition for the decidability of Context Unification. The condition requires unifiers to be "*rank-bound*", a property on terms that does not imply any bound on their size. We will show how Context Unification, under this rank-bound property assumption, can be reduced to Word Unification with Regular Constraints. The reduction requires several encoding techniques and the use of *traversal equations*.

The work presented in this chapter is mostly based on (Levy and Villaret, 2001).

5.1 Introduction

The relationship between Context Unification and Word Unification was originally suggested by Levy (1996). We can easily reduce Word Unification to Context Unification by encoding any word unification problem, like $F\,a\,G \stackrel{?}{=} G\,a\,F$, as the monadic context unification problem $F(a(G(b))) \stackrel{?}{=} G(a(F(b)))$, where a is now a unary constant symbol and b is a new (0-ary) constant. See Subsection 3.3.1 for more details about the relationship between both problems.

This chapter suggests that the opposite reduction may also be possible. In the following subsection we motivate this statement using a naive reduction. Although it does not work, we will see in the rest of the chapter how it could be properly adapted so that it works.

5.1.1 A Naive Reduction

Given a second-order signature, we can encode a term using its *pre-order traversal* sequence. We can use this fact to encode a context unification problem, like

the following one

$$F(G(a,b)) \stackrel{?}{=} G(F(a),b) \tag{5.1}$$

as the following word unification problem

$$F_0\, G_0\, a\, G_1\, b\, G_2\, F_1 \stackrel{?}{=} G_0\, F_0\, a\, F_1\, G_1\, b\, G_2 \tag{5.2}$$

We can easily prove that *if* the context unification problem (5.1) is solvable, *then* its corresponding word unification problem (5.2) is also solvable. In our example, the (word) solution corresponding to the following (context) unifier

$$\begin{array}{l} F \mapsto \lambda x.\ f(f(x,b),b) \\ G \mapsto \lambda x.\ \lambda y.\ f(f(f(x,b),y),b) \end{array} \tag{5.3}$$

is

$$\begin{array}{ll} F_0 \mapsto f\, f & G_0 \mapsto f\, f\, f \\ F_1 \mapsto b\, b & G_1 \mapsto b \\ & G_2 \mapsto b \end{array}$$

Unfortunately, the converse is not true. We can find a solution of the word unification problem which does not correspond to the pre-order traversal of any instantiation of the original context unification problem. For instance, the following unifier:

$$\begin{array}{ll} F_0 \mapsto \epsilon & G_0 \mapsto \epsilon \\ F_1 \mapsto \epsilon & G_1 \mapsto \epsilon \\ & G_2 \mapsto \epsilon \end{array}$$

where ϵ is the empty word, applied to equation 5.2, gives us the word:

$$a\, b$$

that is not a traversal of any term.

Word Unification is decidable (Makanin, 1977), and given a solution of the word unification problem we can check if it corresponds to a solution of the context unification problem. Unfortunately, Word Unification is also infinitary, and we cannot repeat this test for infinitely many word unifiers.

The idea to overcome this difficulty comes from the notion of *rank of a term*. In figure 5.1 there are some examples of terms (represented as trees) with different ranks. Notice that terms with rank bounded by zero are isomorphic to words, and those with rank bounded by one are caterpillars. For signatures of 0-ary and binary symbols, the rank of a term can be defined as follows

$$\begin{array}{l} \mathrm{rank}(a) = 0 \\ \mathrm{rank}(f(t_1,t_2)) = \left\{ \begin{array}{ll} 1 + \mathrm{rank}(t_1) & \text{if } \mathrm{rank}(t_1) = \mathrm{rank}(t_2) \\ \max\{\mathrm{rank}(t_1), \mathrm{rank}(t_2)\} & \text{if } \mathrm{rank}(t_1) \neq \mathrm{rank}(t_2) \end{array} \right. \end{array}$$

Notice that the bound on the rank of a term does not imply any bound on its size, although obviously a bound on the size implies a bound on its rank.

Alternatively, the rank of a binary tree can also be defined as the depth of the greatest complete binary tree that can be embedded in the tree, using the standard embedding of trees.

Nahum Dershowitz made us notice that this notion was already defined for trees under the name of *order* and can be computed by the so-called Horton-Strahler rules (Horton, 1945; Strahler, 1952). The measure is one of the results of the attempts of the geologists to quantify the morphological descriptions of river networks. They intend to reflect, in a quantitative way, the intuitive notions of main and affluent channels in a river network. One of the measures they use is the *order* or *rank*.

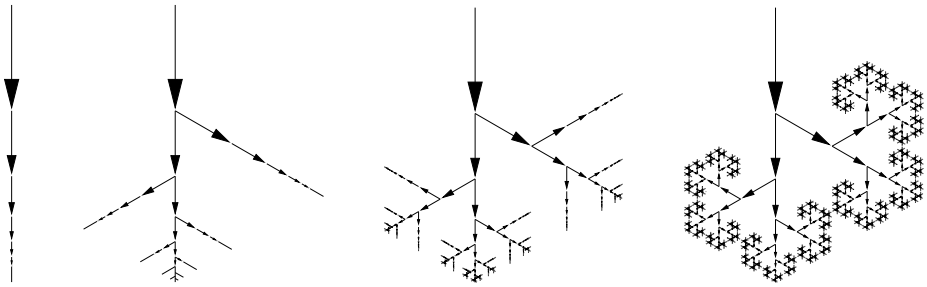


Figure 5.1: Examples of trees with ranks equal to 0, 1, 2 and ∞.

We conjecture that there is a computable function Φ such that, for every solvable context unification problem $t \stackrel{?}{=} u$, there exists a ground unifier σ, such that the rank of $\sigma(t)$ is bounded by the size of the problem as follows: $\text{rank}(\sigma(t)) \leq \Phi(|t \stackrel{?}{=} u|)$.

The other idea is to overcome the difficulties of the previous naive reduction, is to generalise pre-order traversal sequences to a more general notion of *traversal sequence*, by allowing subterms to be traversed in different orders. Then, any rank-bound term has a traversal sequence belonging to a regular language. We also introduce a new notion of *traversal equation*, denoted as $t \equiv u$, which means that t and u are traversal sequences of the same term. We prove that a variant of these constraints can be reduced to word equations with regular constraints which are decidable (Schulz, 1991).

The rest of this chapter proceeds as follows. In Section 5.2 we define positions, we give the notation for permutations and give a formal definition of the *rank*. In Section 5.3 we define the notions of traversal sequence, rank of a traversal sequence, rank of a term, and normal traversal sequence. Traversal equations are introduced in Section 5.4. There, we prove that solvability of rank- and permutation-bound traversal equations is decidable, by reducing the problem to solvability of word equations with regular constraints. In Section 5.5, we state the rank-bound conjecture. Finally, in Section 5.6 we show how, if the conjecture is true, Context Unification could be reduced to rank- and permutation-bound traversal systems.

5.2 Preliminary Definitions

In this section, we introduce some more definitions and notations. We also state the main assumptions of the chapter.

Definition 5.1 *A* position *within a term is defined, using Dewey decimal notation, as a sequence of integers $i_1 \cdots i_n$, and ϵ being the empty sequence. The concatenation of two sequences is denoted by $p_1 \cdot p_2$. The concatenation of an integer and a sequence is also denoted by $i \cdot p$, with $i, j, \ldots$ standing for integers and $p, q, \ldots$ for sequences. The subterm of t at position p is denoted by $t|_p$. If p is a prefix of q then we write $p \leq q$. By $t[u]_p$ we denote the term t where the subterm at position p has been replaced by u. Notice that $t[\ ]$ also denotes a context.*
A position within a problem or an equation is defined by

$$\{t_i \stackrel{?}{=} u_i\}_{i\in\{1..n\}}|_{j\cdot p} = (t_j \stackrel{?}{=} u_j)|_p$$
$$(t \stackrel{?}{=} u)|_{1\cdot p} = t|_p$$
$$(t \stackrel{?}{=} u)|_{2\cdot p} = u|_p$$

The group of permutations of n elements is denoted by Π_n. A permutation ρ of n elements is denoted as a sequence of integers $[\rho(1), \ldots, \rho(n)]$.

Definition 5.2 *The* rank of a term, $\mathrm{rank}(t)$, *is defined by* $\mathrm{rank}(a) = 0$, *for any constant a, and* $\mathrm{rank}(f(t_1, ..., t_n)) = c$ *where c is the minimum integer satisfying: there exists a permutation ρ of indices $1, ..., n$ such that, for any $i \in \{1..n\}$,* $\mathrm{rank}(t_{\rho(i)}) \leq c - n + i$.

This definition is bizarre, but it can be simplified for binary trees.

Definition 5.3 *The* rank of a term, $\mathrm{rank}(t)$, *where all symbols are 0-ary and binary, is defined by:*

$$\begin{array}{lll} \mathrm{rank}(a) & = 0 & \\ \mathrm{rank}(f(t_1, t_2)) & = \left\{ \begin{array}{ll} \mathrm{rank}(t_1) + 1 & \textit{if } \mathrm{rank}(t_1) = \mathrm{rank}(t_2) \\ \max\{\mathrm{rank}(t_1), \mathrm{rank}(t_2)\} & \textit{otherwise} \end{array} \right. & \end{array}$$

for constants a and binary function symbols f.

According to Chapter 4, we can assume that the considered signature is finite, and that it contains, at least, a first-order constant, and only one binary function symbol (say f). This ensures that any solvable context unification problem has a ground unifier. If nothing is said, the signature of a problem is the set of symbols occurring in the problem, plus a first-order constant and the binary constant, if required to fulfil the assumption.

Without loss of generality, we can also assume that the unification problem only contains one equation $t \stackrel{?}{=} u$.

5.3 Terms and Traversal Sequences

The solution to the problems pointed out in the introduction comes from generalising the definition of pre-order traversal sequences. This will allow us to traverse the branches of a tree, i.e. the arguments of a function, in any possible order. In order to reconstruct the term from the traversal sequence, we have to annotate the permutation we have used in this particular traversal sequence. With this purpose, we define a new signature Σ_Π that contains symbols annotated with a permutation that indicates the order in which arguments are traversed. We will firstly provide the Definition of Levy and Villaret (2001) for any signature because it will be used in the next chapter. Then we will give the definition for signatures with just one binary function symbol, that will be used in the rest of this chapter.

Definition 5.4 *Given a signature* Σ*, we define the* (general) extended signature

$$\Sigma^\Pi = \{f^\rho \mid f \in \Sigma \ \wedge \ \rho \in \Pi_{arity(f)}\}$$

where Π_n *is the group of permutations over* n *elements.*
A sequence $s \in (\Sigma^\Pi)^*$ *is said to be a* traversal sequence *of a ground term* t*, noted* $s \in \text{trav}(t)$*, if:*

1. $s = t$ *when* $t = c$ *is a* 0*-ary symbol*
2. $s = f^\rho s_{\rho(1)} \cdots s_{\rho(n)}$ *when* $t = f(t_1, \ldots, t_n)$ *being* s_i *traversal sequences of* t_i *for any* $i \in \{1..n\}$*, and being* $\rho \in \Pi_n$ *a permutation.*

Any traversal sequence of a ground term characterises this term. We use an extended signature with permutations in order to allow us the use of distinct traversals, i.e. the traversals of subterms in distinct possible orders.

In this chapter we are assuming signatures with just one binary function symbol, therefore, our definitions can be simplified and will not use permutations explicitly in the symbols. Instead of containing the two symbols $f^{[1,2]}$ and $f^{[2,1]}$, we will use an extended signature containing f and f', where f denotes that the argument will be traversed from left to right order and f' the other way round.

Definition 5.5 *Given a signature* $\Sigma = \Sigma_0 \cup \{f\}$*, we define the* extended signature

$$\Sigma_\Pi = \Sigma_0 \cup \{f, f'\}$$

We define $arity(f) = arity(f') = 2$*, and the rest of the symbols in* Σ_Π *are constant symbols that have arity* 0*.*
A sequence $s \in (\Sigma_\Pi)^*$ *is said to be a* traversal sequence *of a ground term* $t \in T(\Sigma)$ *if:*

1. $t \in \Sigma_0$*, and* $s = t$*; or*
2. $t = f(t_1, t_2)$*, and either* $s = f\, s_1\, s_2$ *or* $s = f'\, s_2\, s_1$*, where* s_i *is a traversal sequence of* t_i*, for* $i \in \{1, 2\}$*.*

Definition 5.6 *Given a sequence of symbols $a_1 \cdots a_n \in (\Sigma_\Pi)^*$, we define its* width *as*

$$\begin{aligned} &\mathrm{width}(a) = \mathit{arity}(a) - 1 \\ &\mathrm{width}(a_1 \cdots a_n) = \textstyle\sum_{i \in \{1..n\}} \mathrm{width}(a_i) \end{aligned}$$

This definition can be used to characterise *traversal sequences* of ground terms.

Lemma 5.7 *A sequence of symbols $a_1 \cdots a_n \in (\Sigma_\Pi)^*$ is a* traversal sequence, *of some ground term $t \in T(\Sigma)$, if, and only if,*

$$\begin{aligned} &\mathrm{width}(a_1 \cdots a_n) = -1, \textit{ and} \\ &\mathrm{width}(a_1 \cdots a_i) \geq 0, \textit{ for any } i \in \{1..n-1\}. \end{aligned}$$

Proof: For the "if" part we prove, by induction on the length of the sequence, that it determines one and only one term. Such term can be found using the following function $\mathcal{F} : (\Sigma_\Pi)^* \to T(\Sigma)$ defined by:

$$\begin{aligned} &\mathcal{F}(a) = a && \text{if } \mathrm{width}(a) = -1 \\ &\mathcal{F}(f a_2 \ldots a_n) = f(\mathcal{F}(a_2 \cdots a_k), \mathcal{F}(a_{k+1} \cdots a_n)) \\ &\mathcal{F}(f' a_2 \ldots a_n) = f(\mathcal{F}(a_{k+1} \cdots a_n), \mathcal{F}(a_2 \cdots a_k)) \end{aligned}$$

where k is the smallest integer such that $\mathrm{width}(a_2 \cdots a_k) = \mathrm{width}(a_{k+1} \cdots a_n) = -1$. The proof of the "only if" part is trivial. ■

Now we define the rank of a traversal sequence.

Definition 5.8 *Given a sequence of symbols $a_1 \cdots a_n \in (\Sigma_\Pi)^*$, we define its* rank *as*

$$\mathrm{rank}(a_1 \cdots a_n) = \max\{\mathrm{width}(a_1 \cdots a_j) \mid j \in \{0..n\}\}$$

Fact 5.9 *Let $f w_1 w_2$ be a traversal of $f(t_1, t_2)$ such that w_1 is a traversal of t_1 and w_2 is a traversal of t_2. If $\mathrm{rank}(w_1) \geq \mathrm{rank}(w_2)$ then $\mathrm{rank}(f w_1 w_2) = \mathrm{rank}(w_1) + 1$ otherwise $\mathrm{rank}(f w_1 w_2) = \mathrm{rank}(w_2)$ (conversely for $f' w_2 w_1$).*

Proof: The maximal width of the sequence $f w_1$ is the maximal width of the sequence $w_1 + 1$, i.e. $\mathrm{rank}(w_1) + 1$, even more, by Lemma 5.7, $\mathrm{width}(f w_1) = 0$ because w_1 is a traversal. Then, if $\mathrm{rank}(w_1) \geq \mathrm{rank}(w_2)$, adding w_2 to the sequence $f w_1$ will not increase the maximal width, hence we get $\mathrm{rank}(f w_1 w_2) = \mathrm{rank}(w_1) + 1$. Otherwise, if $\mathrm{rank}(w_2) > \mathrm{rank}(w_1)$, then the maximal width of $f w_1 w_2$ is still the one of w_2 or the one of $f w_1$, but being $\mathrm{rank}(w_2) > \mathrm{rank}(w_1)$, we can conclude that $\mathrm{rank}(f w_1 w_2) = \mathrm{rank}(w_2)$. ■

In general, a term has more than one traversal sequence associated. The rank of the term is always smaller or equal to the rank of its traversals, and for at least one of them we have equality. These rank-minimal traversals are relevant for us, and we choose one of them as the *normal traversal sequence*. In figure 5.2, the third traversal sequence $f\,a\,f\,b\,f\,c\,d$ is the normal one.

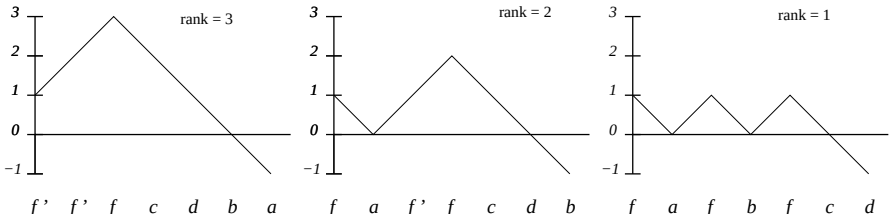


Figure 5.2: Representations of the function $f(i) = \text{width}(a_1 \cdots a_i)$, for some traversal sequences of $f(a, f(b, f(c, d)))$.

Definition 5.10 *Given a term t, its* normal traversal sequence $\text{NF}(t)$ *is defined recursively as follows:*

1. *If $t = a$ then $\text{NF}(t) = a$.*
2. *If $t = f(t_1, t_2)$ then:*
   - *when $\text{rank}(t_1) \leq \text{rank}(t_2)$ we have $\text{NF}(t) = f\ \text{NF}(t_1)\ \text{NF}(t_2)$, and*
   - *when $\text{rank}(t_2) < \text{rank}(t_1)$ we have $\text{NF}(t) = f'\ \text{NF}(t_2)\ \text{NF}(t_1)$.*

Lemma 5.11 *If w is a traversal of t, then $\text{rank}(w) \geq \text{rank}(t)$.*
Moreover, the normal traversal sequence of a term t has minimal rank among their traversals, i.e. $\text{rank}(t) = \text{rank}(\text{NF}(t))$.

Proof: For the first part of the Lemma we proceed by structural induction on the structure of the term. For terms like a, it is trivial. For terms like $t = f(t_1, t_2)$, with $w = fw_1w_2$ such that w_1 is a traversal of t_1 and w_2 a traversal of t_2, we have two possibilities:

- Let $\text{rank}(t_1) = \text{rank}(t_2)$, then $\text{rank}(t) = \text{rank}(t_1) + 1 = \text{rank}(t_2) + 1$. By Fact 5.9, if $\text{rank}(w_1) \geq \text{rank}(w_2)$ then $\text{rank}(w) = \text{rank}(w_1) + 1$ and by induction hypothesis, $\text{rank}(w_1) \geq \text{rank}(t_1)$ hence $\text{rank}(w) \geq \text{rank}(t)$; otherwise if $\text{rank}(w_2) > \text{rank}(w_1)$ then $\text{rank}(w) = \text{rank}(w_2)$, hence by induction hypothesis together with $\text{rank}(t_1) = \text{rank}(t_2)$ and $\text{rank}(w_2) > \text{rank}(w_1)$ we can conclude $\text{rank}(w) \geq \text{rank}(t)$.

- Let $\text{rank}(t_1) > \text{rank}(t_2)$ (the converse is similar), then $\text{rank}(t) = \text{rank}(t_1)$. By Fact 5.9, if $\text{rank}(w_1) \geq \text{rank}(w_2)$ then $\text{rank}(w) = \text{rank}(w_1) + 1$ and by induction hypothesis, $\text{rank}(w_1) \geq \text{rank}(t_1)$ hence $\text{rank}(w) \geq \text{rank}(t)$; otherwise, $\text{rank}(w) = \text{rank}(w_2) > \text{rank}(w_1)$ and by induction hypothesis $\text{rank}(w_1) \geq \text{rank}(t_1)$ hence $\text{rank}(w) \geq \text{rank}(t)$.

Proving that $\text{rank}(t) = \text{rank}(\text{NF}(t))$ is enough for the second part of the Lemma. Again we proceed by structural induction on the structure of the term. For terms like a, it is trivial. For terms like $t = f(t_1, t_2)$, we have three possibilities:

- Let $\mathrm{rank}(t_1) = \mathrm{rank}(t_2)$, then $\mathrm{NF}(t) = f\,\mathrm{NF}(t_1)\,\mathrm{NF}(t_2)$ and $\mathrm{rank}(t) = \mathrm{rank}(t_1)+1$, moreover, by induction hypothesis, $\mathrm{rank}(\mathrm{NF}(t_1)) = \mathrm{rank}(t_1) = \mathrm{rank}(t_2) = \mathrm{rank}(\mathrm{NF}(t_2))$. Then, by Fact 5.9, $\mathrm{rank}(f\,\mathrm{NF}(t_1)\,\mathrm{NF}(t_2)) = \mathrm{rank}(\mathrm{NF}(t_1)) + 1 = \mathrm{rank}(t_1) + 1 = \mathrm{rank}(t)$.
- Let $\mathrm{rank}(t_1) < \mathrm{rank}(t_2)$, then $\mathrm{NF}(t) = f\,\mathrm{NF}(t_1)\,\mathrm{NF}(t_2)$ and $\mathrm{rank}(t) = \mathrm{rank}(t_2)$, moreover, by induction hypothesis, $\mathrm{rank}(\mathrm{NF}(t_1)) = \mathrm{rank}(t_1)$ and $\mathrm{rank}(\mathrm{NF}(t_2)) = \mathrm{rank}(t_2)$. Then, by Fact 5.9, $\mathrm{rank}(f\,\mathrm{NF}(t_1)\,\mathrm{NF}(t_2)) = \mathrm{rank}(\mathrm{NF}(t_2)) = \mathrm{rank}(t_2) = \mathrm{rank}(t)$.
- Let $\mathrm{rank}(t_1) > \mathrm{rank}(t_2)$, then $\mathrm{NF}(t) = f'\,\mathrm{NF}(t_2)\,\mathrm{NF}(t_1)$ and $\mathrm{rank}(t) = \mathrm{rank}(t_1)$, moreover, by induction hypothesis, $\mathrm{rank}(\mathrm{NF}(t_1)) = \mathrm{rank}(t_1)$ and $\mathrm{rank}(\mathrm{NF}(t_2)) = \mathrm{rank}(t_2)$. Then, by Fact 5.9, $\mathrm{rank}(f'\,\mathrm{NF}(t_2)\,\mathrm{NF}(t_1)) = \mathrm{rank}(\mathrm{NF}(t_1)) = \mathrm{rank}(t_1) = \mathrm{rank}(t)$.

■

Rank-upper bounded traversal sequences define a regular language. The construction of associated automata can be found in (Levy and Villaret, 2000) (see Section 6.4).

Definition 5.12 *Given an extended signature Σ_Π and a constant k, the set of* k-bound traversal sequences *is defined as follows:*

$$R^k_\Sigma = \{s \in (\Sigma_\Pi)^* \mid \mathrm{rank}(s) \leq k \ \wedge\ s \text{ is a traversal}\}$$

Lemma 5.13 *Given an extended signature Σ_Π and a constant k, the set of k-bound traversal sequences is a regular language.*

Proof: We can define R^k_Σ inductively as follows:

$$R^0_\Sigma = \Sigma_0$$
$$R^k_\Sigma = R^{k-1}_\Sigma \cup \left((f|f')\ R^{k-1}_\Sigma\right)^*\ \Sigma_0$$

■

5.4 Traversal Equations

In this section we introduce *traversal equations* and *traversal systems*. Solvability of traversal equations and of traversal systems is still an open question, but we prove that a variant of these (the so called *permutation and rank-bound traversal systems*) can be reduced to word equations systems with regular constraints, which are decidable (Schulz, 1991). This reduction is somehow inspired by the reduction from trace equations to word equations used by Diekert *et al.* (1997) to prove decidability of trace equations.

Later in Section 5.6, we will reduce Context Unification to solvability of permutation and rank-bound traversal systems; we need the rank-bound conjecture to prove that this reduction can be done. The conjecture will be presented in the next section, Section 5.5.

Definition 5.14 *A* traversal system *over an extended signature Σ_Π with word variables $\mathcal{W}$ is a conjunction of literals, where every literal has the form $w_1 \stackrel{?}{=} w_2$* (word equation), *$w_1 \equiv w_2$* (traversal equation) *or $w \in R$* (regular constraint), *with $w_i \in (\Sigma_\Pi \cup \mathcal{W})^*$ being words with variables and $R \subseteq (\Sigma_\Pi)^*$ a regular language.*
A solution *of a traversal system is a word substitution $\sigma : \mathcal{W} \to (\Sigma_\Pi)^*$ such that:*

1. *$\sigma(w_1) = \sigma(w_2)$ for any word equation $w_1 \stackrel{?}{=} w_2$,*
2. *$\sigma(w_1)$ and $\sigma(w_2)$ are both traversal sequences of the same term, for any traversal equation $w_1 \equiv w_2$,*
3. *and $\sigma(w)$ belongs to R, for any regular constraint $w \in R$.*

5.4.1 Rank-bound Traversal Systems

Definition 5.15 *A traversal system is said to be* rank-bound *if, for every traversal equation $w_1 \equiv w_2$, there exist two constants k_1 and k_2, and two regular constraints $w_1 \in R_\Sigma^{k_1}$ and $w_2 \in R_\Sigma^{k_2}$ in the system, where $R_\Sigma^{k_i}$ is the (regular) set of k_i-bound traversal sequences.*

We can transform rank-bound traversal systems into equivalent systems of word unification with regular constraints using the following transformation rules.

Definition 5.16 *The following rules define a non-deterministic translation procedure from rank-bound traversal systems into word equations with regular constraints.*

Rule 1: *We guess two symbols γ_1, γ_2 from $\{f, f'\} \subset \Sigma_\Pi$, being ρ and ρ' their corresponding permutations for the order of traversing the arguments. Then we replace the traversal equation $w_1 \equiv w_2$ and the corresponding regular constraints $w_1 \in R_\Sigma^{k_1}$ and $w_2 \in R_\Sigma^{k_2}$ by*

$$
\begin{array}{lll}
 & w_1 \in R_\Sigma^{k_1} & \\
 & w_2 \in R_\Sigma^{k_2} & \\
w_1 \equiv w_2 & w_1 \stackrel{?}{=} X_1\ \gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)}\ X_2 & \\
w_1 \in R_\Sigma^{k_1} \implies & w_2 \stackrel{?}{=} X_1\ \gamma_2\ Y'_{\rho'(1)}\, Y'_{\rho'(2)}\ X_2 & \\
w_2 \in R_\Sigma^{k_2} & Y_i \equiv Y'_i & \\
 & Y_{\rho(i)} \in R_\Sigma^{k_1-2+i} & \text{for any } i \in \{1,2\} \\
 & Y'_{\rho'(i)} \in R_\Sigma^{k_2-2+i} &
\end{array}
$$

where $\{X_i, Y_i, Y'_i\}_{i\in\{1,2\}}$ are fresh word variables.

Rule 2: *We replace the traversal equation* $w_1 \equiv w_2$ *and the corresponding regular constraints* $w_1 \in R_\Sigma^{k_1}$ *and* $w_2 \in R_\Sigma^{k_2}$ *by*

$$\left.\begin{array}{l} w_1 \equiv w_2 \\ w_1 \in R_\Sigma^{k_1} \\ w_2 \in R_\Sigma^{k_2} \end{array}\right. \Longrightarrow \begin{array}{l} w_1 \stackrel{?}{=} w_2 \\ w_1 \in R_\Sigma^{\min\{k_1,k_2\}} \end{array}$$

Notice that if the rank of a traversal sequence $f\, w_1\, w_2$ is bounded by k, then, for any $i \in \{1,2\}$, the rank of w_i is bounded by $k-2+i$. These are the values of the exponents used in the regular restrictions of the right-hand side of Rule 1. Rank-boundness is crucial in order to ensure soundness of Rule 2 because this allows us to enforce words to be traversals of terms. For instance, the traversal equation $X\, a\, a\, Y \equiv Y\, a\, a\, X$ has no solution, whereas the word equation $X\, a\, a\, Y \stackrel{?}{=} Y\, a\, a\, X$ is solvable. Notice that some substitutions, like $X, Y \mapsto a$, give equal sequences, but they are not traversal sequences.

Theorem 5.17 *The rules of Definition 5.16 describe a sound and complete decision procedure for rank-bound traversal systems. In other words, for any rank-bound traversal system* S*,*

1. *if* $S \Longrightarrow^* S'$ *and the substitution* σ' *is a solution of* S'*, then* $\sigma'|_{Var(S)}$ *is also a solution of* S*, and*
2. *if the substitution* σ *is a solution of* S*, then there exists a word unification problem with regular constraints* S'*, a finite transformation sequence* $S \Longrightarrow^* S'$*, and an extension* σ' *of* σ*, such that* σ' *is a solution of* S'*.*

Proof:

1. We proceed by induction on the length of the transformation sequence $S \Rightarrow^* S'$. For 0 transformations is obvious.

Consider the step $S \Rightarrow S'$. We have two possibilities, either *Rule 1* or *Rule 2* has been the rule applied on this step. If it is *Rule 2*, obviously $\sigma'|_{Dom(S)}$ is also a solution of S. Otherwise, let $w_1 \equiv w_2$ be the traversal equation with the two regular constraints $w_1 \in R_\Sigma^{k_1}$ and $w_2 \in R_\Sigma^{k_2}$, in S, where *Rule 1* has been applied and, hence S' is equal to S up to this traversal equation (notice that the regular constraints remain) plus the following literals

$$\begin{array}{l} w_1 \stackrel{?}{=} X_1\ \gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)}\ X_2 \\ w_2 \stackrel{?}{=} X_1\ \gamma_2\ Y'_{\rho'(1)}\, Y'_{\rho'(2)}\ X_2 \\ \left.\begin{array}{l} Y_i \equiv Y'_i \\ Y_{\rho(i)} \in R_\Sigma^{k_1-2+i} \\ Y'_{\rho'(i)} \in R_\Sigma^{k_2-2+i} \end{array}\right\} \text{for any } i \in \{1,2\} \end{array}$$

Notice that by induction hyphotesis $\sigma'|_{Dom(S')}$ solves S'. We just need to prove that $\sigma'|_{Dom(S)}(w_1)$ and $\sigma'|_{Dom(S)}(w_2)$ are both traversals of the same term. We

know that $\sigma'|_{Dom(S')}(w_1)$ is a traversal of some term t and $\sigma'|_{Dom(S')}(w_2)$ is a traversal of some term t' because $\sigma'|_{Dom(S')}$ satisfies $w_1 \in R_\Sigma^{k_1}$ and $w_2 \in R_\Sigma^{k_2}$. We have to prove that $t = t'$. We consider the first word equation $w_1 \stackrel{?}{=} X_1\ \gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)}\ X_2$. We know that $\sigma'|_{Dom(S')}(Y_1)$ and $\sigma'|_{Dom(S')}(Y_2)$ are both traversals of terms (say t_1 and t_2), and γ_1 (without loss of generality let $\gamma_1 = f$) corresponds to the binary function symbol f, then $\sigma'|_{Dom(S')}(\gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)})$ is a traversal of $f(t_1, t_2)$. Let C be the context such that $\sigma'|_{Dom(S')}(X_1\, a\, X_2)$ is a traversal of $C(a)$ and $\sigma'|_{Dom(S')}(X_1\, b\, X_2)$ is a traversal of $C(b)$, then, knowing that $\sigma'|_{Dom(S')}(X_1\ \gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)}\ X_2) = \sigma'|_{Dom(S)}(w_1)$ is a traversal of t, we can conclude that $C(f(t_1, t_2)) = t$. We can do the same reasoning for the second equation and conclude that $t' = C(f(t_1, t_2))$ hence $t = t'$, notice that $\sigma'|_{Dom(S')}$ also satisfy the traversal equations $Y_i \equiv Y_i'$ (for $i \in \{1, 2\}$).

2. Let S be a traversal equations system with solution σ. Consider $w_1 \equiv w_2$, $w_1 \in R_\Sigma^{k_1}$ and $w_2 \in R_\Sigma^{k_2}$ in S. Let t be the term such that $\sigma(w_1) \in \text{trav}(t)$ and $\sigma(w_2) \in \text{trav}(t)$. Now we have two possibilities:

- if $\sigma(w_1) \neq \sigma(w_2)$, then we apply *Rule 1* and we obtain the new traversal system S' by means of removing the traversal equation and adding these literals:

$$\begin{array}{l} w_1 \stackrel{?}{=} X_1\ \gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)}\ X_2 \\ w_2 \stackrel{?}{=} X_1\ \gamma_2\ Y'_{\rho'(1)}\, Y'_{\rho'(2)}\ X_2 \\ \left.\begin{array}{l} Y_i \equiv Y_i' \\ Y_{\rho(i)} \in R_\Sigma^{k_1-2+i} \\ Y'_{\rho'(i)} \in R_\Sigma^{k_2-2+i} \end{array}\right\} \text{for any } i \in \{1, 2\} \end{array}$$

  Let $t = C(f(t_1, t_2))$ such that C is the biggest context that is traversed equally in $\sigma(w_1)$ and in $\sigma(w_2)$; let $\sigma(w_1) = w_{X_1}\gamma_1 w_{Y_1} w_{Y_2} w_{X_2}$ and $\sigma(w_2) = w_{X_1}\gamma_2 w_{Y_1'} w_{Y_2'} w_{X_2}$ where $\gamma_1 w_{Y_1} w_{Y_2}$ is the traversal of $f(t_1, t_2)$ in $\sigma(w_1)$ and $\gamma_2 w_{Y_1'} w_{Y_2'}$ the traversal of $f(t_1, t_2)$ in $\sigma(w_2)$. Moreover, $\gamma_1, \gamma_2 \in \{f, f'\}$ and if $\gamma_1 = f$ then w_{Y_i} is a traversal of t_i, otherwise if $\gamma_1 = f'$ then w_{Y_1} is a traversal of t_2 and w_{Y_2} is a traversal of t_1 (similarly for γ_2 and $w_{Y_1'}, w_{Y_2'}$).

  Now we can extend substitution σ and obtain a solution of S' as follows: Let σ' be $\sigma \circ [X_1 \mapsto w_{X_1}, X_2 \mapsto w_{X_2}, Y_1 \mapsto w_{Y_1}, Y_2 \mapsto w_{Y_2}, Y_1' \mapsto w_{Y_1'}, Y_2' \mapsto w_{Y_2'}]$. It is easy to check that σ' satisfies all literals of S'.

- if $\sigma(w_1) = \sigma(w_2)$ we use *Rule 2* and we obtain a new traversal system S' where the traversal equation has been replaced by the word equation $w_1 \stackrel{?}{=} w_2$ with the regular constraint $w_1 \in R_\Sigma^{\min\{k_1, k_2\}}$ that is trivially solvable with substitution σ hence in this case, $\sigma' = \sigma$.

Notice that given a traversal system and a solution, at each application of a rule, we are closer to obtain a word unification with regular constraints system because *Rule 1* replaces a traversal equation by two word equations and two traversal equations where the size of the solution is strictly smaller than the

original one, and *Rule 2* replaces a traversal equation by a single word equation. ■

Unfortunately, this non-deterministic transformation procedure does not always terminate. Notice that we can have $\gamma_1 = \gamma_2 = f$, hence $\rho(2) = \rho'(2) = 2$, and in such case we obtain a traversal equation $Y_2 \equiv Y_2'$ with the same bounds $Y_2 \in R_\Sigma^{k_1}$ and $Y_2' \in R_\Sigma^{k_2}$ as the original ones.

However, an adaptation of these transformation rules can be used to find solutions σ of equations $w_1 \equiv w_2$, such that $\sigma(w_1)$ and $\sigma(w_2)$ are traversal sequences for the same term, and they are "similar", where "similar" means that they only differ in a bounded number of permutations.

5.4.2 Permutation and Rank-bound Traversal Systems

Definition 5.18 *Given two traversal sequences v and w over Σ_Π, we say that they differ in b permutations if, either*

1. *$v = f\, r_1\, r_2$ and $w = f\, s_1\, s_2$ (or $v = f'\, r_2\, r_1$ and $w = f'\, s_2\, s_1$) , and for any $i \in \{1, 2\}$, r_i and s_i differ in b_i permutations, and $b_1 + b_2 = b$, or*

2. *$v = f\, r_1\, r_2$ and $w = f'\, s_2\, s_1$, for any $i \in \{1, 2\}$, r_i and s_i differ in b_i permutations, and $b_1 + b_2 = b - 1$.*

Definition 5.19 *A* permutation-bound traversal equation*, denoted as $w_1 \equiv_b w_2$, is a tuple of two words with variables w_1 and w_2, and an integer b ($b \geq 0$).*
A substitution σ is said to be a solution of a permutation-bound traversal equation $w_1 \equiv_b w_2$ if $\sigma(w_1)$ and $\sigma(w_2)$ are both traversal sequences of the same term, and they only differ in at most b permutations.
A permutation and rank-bound traversal system *is a rank-bound traversal system where all traversal equations are permutation-bound.*

A slight modification of the rules of Definition 5.16 provides us rules to deal with permutation and rank-bound traversal systems.

Definition 5.20 *The following rules define a non-deterministic translation procedure from permutation and rank-bound traversal systems into word equations with regular constraints.*

Rule 1: *We guess two symbols γ_1, γ_2 from $\{f, f'\} \subset \Sigma_\Pi$, being ρ and ρ' their corresponding permutations for the order of traversing the arguments. Then we replace the permutation-bound traversal equation $w_1 \equiv_b w_2$ and*

the corresponding regular constraints $w_1 \in R_\Sigma^{k_1}$ *and* $w_2 \in R_\Sigma^{k_2}$ *by*

$$\begin{array}{ll} & w_1 \in R_\Sigma^{k_1} \\ & w_2 \in R_\Sigma^{k_2} \\ w_1 \equiv_b w_2 & w_1 \stackrel{?}{=} X_1\ \gamma_1\ Y_{\rho(1)}\, Y_{\rho(2)}\ X_2 \\ w_1 \in R_\Sigma^{k_1} \implies & w_2 \stackrel{?}{=} X_1\ \gamma_2\ Y'_{\rho'(1)}\, Y'_{\rho'(2)}\ X_2 \\ w_2 \in R_\Sigma^{k_2} & Y_i \equiv_{b_i} Y'_i \\ & Y_{\rho(i)} \in R_\Sigma^{k_1-2+i} \\ & Y'_{\rho'(i)} \in R_\Sigma^{k_2-2+i} \end{array} \Bigg\} \text{ for any } i \in \{1,2\}$$

where $\{X_i, Y_i, Y'_i\}_{i\in\{1,2\}}$ *are fresh word variables, and:*

- *if* $\gamma_1 = \gamma_2$ *then* $b_1 + b_2 = b$ *and* $b_1, b_2 > 0$*, otherwise*
- *if* $\gamma_1 \neq \gamma_2$ *then* $b_1 + b_2 = b - 1$ *and* $b_1, b_2 \geq 0$*.*

Rule 2: *We replace the permutation-bound traversal equation* $w_1 \equiv_b w_2$ *and the corresponding regular constraints* $w_1 \in R_\Sigma^{k_1}$ *and* $w_2 \in R_\Sigma^{k_2}$ *by*

$$\begin{array}{ll} w_1 \equiv_b w_2 & \\ w_1 \in R_\Sigma^{k_1} \implies & w_1 \stackrel{?}{=} w_2 \\ w_2 \in R_\Sigma^{k_2} & w_1 \in R_\Sigma^{\min\{k_1,k_2\}} \end{array}$$

Theorem 5.21 *Solvability of permutation and rank-bound traversal systems is decidable.*

Proof: We can reduce any permutation and rank-bound traversal system to an equivalent word unification problem with regular constraints using the rules of Definition 5.20 finitely many times. On the one hand, is easy to prove that this transformation process always terminates using a multiset ordering on bounds of permutation-bound traversal equations. Notice that each time we apply a rule the bounds on the permutations decrease. And on the other hand, notice that we can easily proof soundness and completeness of the rules of Definition 5.20 using the same arguments than in Theorem 5.17. ■

5.5 The Rank-Bound Conjecture

In this section we introduce the rank-bound conjecture. This is the base of the reduction of Context Unification to permutation- and rank-bound traversal systems described in the next section. As we will see, this conjecture is essential in order to prove that the traversal equations that we find in the reduction are both permutation-bound and rank-bound.

Conjecture 5.22 *[Rank-Bound Conjecture] There exists a computable function Φ such that, for any solvable context unification problem $t \stackrel{?}{=} u$ there exists a ground unifier σ satisfying*

$$\mathrm{rank}(\sigma(t)) \leq \Phi(|t \stackrel{?}{=} u|)$$

The validity of the conjecture is still an open question. In fact, we think that the conjecture is true, not only for *just one* ground unifier, but for *any* most general unifier. This stronger version of the conjecture is not true for Second-Order Unification, because we can have most general second-order unifiers with an arbitrarily large rank, as shown by the following example.

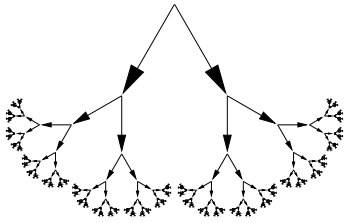
Example 5.23 The second-order unification problem

$$F(f(a,a)) \stackrel{?}{=} f(F(a), F(a))$$

has only one context unifier $\sigma = [F \mapsto \lambda x.\ x]$. However, it has infinitely many second-order unifiers which are not context unifiers, like

$$\sigma = [F \mapsto \lambda x.\ f(f(f(x,x), f(x,x)), f(f(x,x), f(x,x)))]$$

For any $n \geq 0$, there is a second-order unifier where the bound variable x occurs 2^n times in the body of the function, and the rank of $\sigma(F(f(a,a)))$ is equal to $n+1$. This term $\sigma(F(f(a,a)))$ can be represented as follows for $n = \infty$.



■

In the following Lemma we prove that the conjecture is true for First-Order Unification.

Lemma 5.24 *Given a solvable first-order unification problem $t \stackrel{?}{=} u$, its most general unifier σ satisfies*

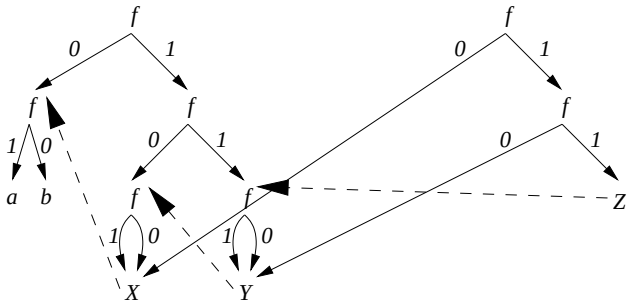
$$\mathrm{rank}(\sigma(t)) \leq |t| + |u|$$

Proof: Suppose we have a unification problem $t \stackrel{?}{=} u$ like

$$f(f(a,b), f(f(X,X), f(Y,Y))) \stackrel{?}{=} f(X, f(Y,Z))$$

We can represent it by a directed acyclic graph where we have two initial nodes (one for each side of the equation), and a unique node per variable. We can solve the unification problem by re-addressing the arrows pointing to a variable,

when this variable is instantiated. Therefore we can represent $\sigma(t)$ by means of a directed acyclic graph D, where $|D| \leq |t| + |s|$, being the size of the graph its number of arrows. This is the representation of the graph corresponding to our example (where, for simplicity, we have added a dashed arrow instead of re-addressing arrows pointing to variables):



Now we can label the graph to indicate the order how it could be traversed, 1-labeled arrows are traversed first and then the 0-labeled. For any labeling of the original graph, the same labels in the graph resulting from instantiation represent a traversal sequence of $\sigma(t)$ and a traversal sequence of $\sigma(u)$.[1] Defining the rank of a node as the addition of the label in the path from the root to this node, the rank of the traversal sequence will be the maximal of the rank of all leaves. In our example, this rank is 5 and it is obtained from the following path

$$f \xrightarrow{1} f \xrightarrow{1} f \xrightarrow{1} f \xrightarrow{1} f \xrightarrow{1} a$$

The rank of a path never exceeds the number of arrows of the graph, i.e. its size, because, to avoid occur-check failure, we cannot repeat nodes in a path. Therefore, when we use an arrow, there is at least another one (the one with the same origin) that cannot be contained in the same path. We can conclude that the traversal sequence of $\sigma(t)$ represented in the path satisfies $\text{rank}(s) \leq |t|+|u|$, thus $\text{rank}(\sigma(t)) \leq |t| + |u|$. ■

Remark: In the proof of Lemma 5.24 and Lemma 5.26 we use labeled trees in order to represent traversal sequences. We can represent any term as a tree, and its normal traversal sequence as a labeled tree, where we decorate every node with the permutation we use to traverse its sons. Alternatively, we can represent this traversal sequence using a tree with labeled arrows. We proceed as follows. Consider the tree associated to $f(t_1, t_2)$, and let n_i be the label of the arrow going from f to the subtree associated to t_i. We choose the values of n_i such that,

$$\begin{aligned} \text{if } \text{rank}(t_1) \leq \text{rank}(t_2) \text{ then} \quad & n_1 = 1 \text{ and } n_2 = 0, \\ \text{otherwise} \quad & n_1 = 0 \text{ and } n_2 = 1 \end{aligned}$$

[1] Notice that, since we are using graphs instead of trees, not all traversal sequences of $\sigma(t)$ can be represented as a labeling of the graph.

It is easy to see that, if in a labeled tree we first follow the arrows with higher labels, we get the normal traversal sequence. Moreover, if we compute the sum of the labels of the arrows we follow from the root to any leave, and we take the maximum of these sums, we get the rank of the tree.

5.6 Reducing Context Unification to Traversal Equations

In this section we prove that Context Unification can be reduced to solvability of traversal systems. Moreover, we also prove that if the rank-bound conjecture is true, then this reduction can be done to permutation and rank-bound traversal systems. Therefore, if the conjecture is true, then Context Unification is decidable.

The reduction is very similar to the naive reduction described in Subsection 5.1.1: first-order variables X are encoded as word variables X' such that, if σ is a solution of the context unification problem, and σ' is the corresponding solution of the equivalent word unification problem, then $\sigma'(X') = \mathrm{NF}(\sigma(X))$.

For every n-ary context variable F, we would need $n+1$ word variables $F'_0,\ldots,F'_n$, such that $\sigma'(F'_0\, a\, F'_1\, a \cdots F'_{n-1}\, a\, F'_n) = \mathrm{NF}(\sigma(F(a,\ldots,a)))$. However, this simple translation does not work. If a term t contains two occurrences of a first-order variable X, then $\mathrm{NF}(\sigma(t))$ will contain two occurrences of $\mathrm{NF}(\sigma(X))$. However, two different occurrences of a context variable can have different arguments, and this means that the context $\sigma(F)$ can be traversed in different ways, depending on the arguments. Notice that, in general, even if $\mathrm{NF}(C(a)) = w_0\, a\, w_1$,[2] we can have $\mathrm{NF}(C(u)) \neq w_0\ \mathrm{NF}(u)\, w_1$. Fortunately, the different ways in which the occurrences of $\sigma(F)$ are traversed in the normal form of $\sigma(C)$ are not very different, i.e. they differ in at most a bounded number of permutations.

Example 5.25 Let $\sigma(F) = \lambda x\,.\, f(f(x,t_1),t_2)$, where $\mathrm{rank}(t_1) < \mathrm{rank}(t_2)$, and $w_i = \mathrm{NF}(t_i)$, for $i = 1,2$. Depending on the argument u, we have

$$\mathrm{NF}(\sigma(F(u))) = \begin{cases} f\, f\ \mathrm{NF}(\sigma(u))\, w_1\, w_2 & \text{if } \mathrm{rank}(\sigma(u)) \leq \mathrm{rank}(t_1) \\ f\, f'\, w_1\ \mathrm{NF}(\sigma(u))\, w_2 & \text{if } \mathrm{rank}(t_1) < \mathrm{rank}(\sigma(u)) \leq \mathrm{rank}(t_2) \\ f'\, w_2\, f'\, w_1\ \mathrm{NF}(\sigma(u)) & \text{if } \mathrm{rank}(t_2) < \mathrm{rank}(\sigma(u)) \end{cases}$$

■

Lemma 5.26 *Let F be a context variable and σ a substitution. For any two terms $F(t_1,\ldots,t_n)$ and $F(u_1,\ldots,u_n)$, there exist sequences $v_0,\ldots,v_n, w_0,\ldots,w_n$ and permutations $\rho,\tau \in \Pi_n$, such that*

$$\mathrm{NF}(\sigma(F(t_1,\ldots,t_n))) = v_0\ \mathrm{NF}(\sigma(t_{\rho(1)}))\, v_1 \cdots v_{n-1}\ \mathrm{NF}(\sigma(t_{\rho(n)}))\, v_n$$
$$\mathrm{NF}(\sigma(F(u_1,\ldots,u_n))) = w_0\ \mathrm{NF}(\sigma(u_{\tau(1)}))\, w_1 \cdots w_{n-1}\ \mathrm{NF}(\sigma(u_{\tau(n)}))\, w_n$$

[2] Recall that C denotes a context.

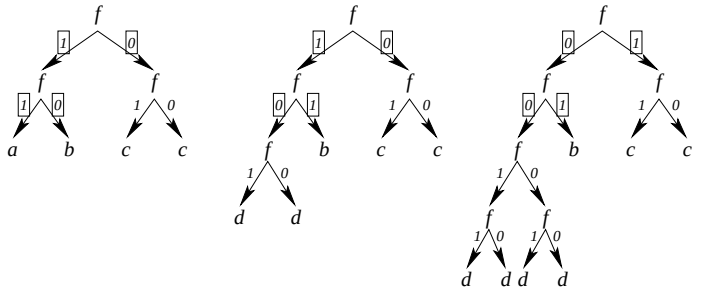
and, for any sequence of constants $\{a_i\}_{i\in\{1..n\}}$,

$$v_0\, a_{\rho(1)}\, v_1 \cdots v_{n-1}\, a_{\rho(n)}\, v_n$$
$$w_0\, a_{\tau(1)}\, w_1 \cdots w_{n-1}\, a_{\tau(n)}\, w_n$$

are both traversal sequences of $\sigma(F(a_1,\ldots,a_n))$*, and they only differ in at most* $n \cdot \mathrm{rank}(\sigma(F(a_1,\ldots,a_n)))$ *permutations.*

Proof: We can prove that, for any context C, and term u, there exist two sequences v_0 and v_1 such that $\mathrm{NF}(C(u)) = v_0\, \mathrm{NF}(u)\, v_1$ and $v_0\, a\, v_1$ is a traversal sequence of $C(a)$. This can be generalised to n-arguments, and proves the first part of the lemma.

For the second part of the lemma, consider the normal traversal sequences of $\sigma(F(a_1,\ldots,a_n))$ and of $\sigma(F(u_1,\ldots,u_n))$, for any u_i's. We can represent these traversal sequences as labeled trees. For instance, for the terms of example 5.25, with $t_1 = b$ and $t_2 = f(c,c)$, and with $u = a$, $u = f(d,d)$ and $u = f(f(d,d),f(d,d))$, we get the following labeled tree:



If we compare the labels of the context $\sigma(F(\_,\ldots,\_))$ in these two terms $\sigma(F(a_1,\ldots,a_n))$, and of $\sigma(F(u_1,\ldots,u_n))$, we see that all the changes involve arrows which have their origin at some point of the path to an argument. Obviously, the traversal of the subterms of $\sigma(F(a_1,\ldots,a_n))$, that do not contain the arguments $a_1,\ldots,a_n$ are labeled as in $\sigma(F(u_1,\ldots,u_n))$. Moreover, as far as $\mathrm{rank}(a_i) \leq \mathrm{rank}(u_i)$, the change can only consist of decreasing the label of the arrow leading to the new argument u_i, and increasing the label of the other arrow with the same origin. If we only consider the path leading to one of the arguments a_i in $\sigma(F(a_1,\ldots,a_n))$, the number of arrows with non-null labels in this path is not greater than $\mathrm{rank}(\sigma(F(a_1,\ldots,a_n)))$, because the sum of the labels of this path is not greater than the rank of the term. Therefore, at most as many arrows as $\mathrm{rank}(\sigma(F(a_1,\ldots,a_n)))$ can decrease their label. As there are $arity(F)$ many paths to arguments, there are at most as many nodes as $arity(F) \cdot \mathrm{rank}(\sigma(F(a_1,\ldots,a_n)))$ with distinct permutations in $\sigma(F(a_1,\ldots,a_n))$ and in $\sigma(F(u_1,\ldots,u_n))$. All these nodes are in the path to an argument, and their arrow pointing to the argument has a non-null label. These are precisely the nodes that can have distinct permutations in $\sigma(F(u_1,\ldots,u_n))$ and in $\sigma(F(u'_1,\ldots,u'_n))$, independently of the values of $\mathrm{rank}(u_i)$ and $\mathrm{rank}(u'_i)$. ■

Notice that we need the rank-bound conjecture in order to bound the value of $\text{rank}(\sigma(F(a_1, \ldots, a_n)))$, i.e. to prove that those two traversal sequences differ in a bounded number of permutations.

In the rest we describe how a context unification problem could be effectively translated into an equivalent system of traversal equations.

Theorem 5.27 *Context Unification can be reduced to solvability of traversal systems.*
If the Rank-Bound Conjecture is true, then Context Unification can be reduced to solvability of permutation and rank-bound traversal systems.

Proof: Let $t \stackrel{?}{=} u$ be the original context unification problem, and Σ be the original signature, with variables $\mathcal{X}$. We assume that Σ is finite, and contains at least $2 \cdot n$ distinct first-order constants $a_1, \ldots, a_n, b_1, \ldots, b_n$, where $n = \max\{arity(F) \mid F \in (Var(t) \cup Var(u))\}$, and a binary symbol f, and that $a_1, \ldots, a_n, b_1, \ldots, b_n$ do not occur in $t \stackrel{?}{=} u$. Therefore, if a problem is solvable, it has a ground unifier.

First step The order of the arguments in F and in $\sigma(F)$ are not necessarily the same. In this first step we guess a permutation $\rho_F \in \Pi_{arity(F)}$ for any context variable and transform $t \stackrel{?}{=} u$ into $\sigma_0(t) \stackrel{?}{=} \sigma_0(u)$ where

$$\sigma_0 = \bigcup_{F \in Var(t) \cup Var(u)} [F \mapsto \lambda x_1 \cdots x_n.\ F'(x_{\rho_F(1)}, \ldots, x_{\rho_F(n)})]$$

Now, we can assume that F' and its instance have their arguments in the same order. Moreover, as far as σ_0 is simply a renaming substitution, $t \stackrel{?}{=} u$ and $\sigma_0(t) \stackrel{?}{=} \sigma_0(u)$ are equivalent problems.

Second step We introduce a word variable $X' \in \mathcal{W}$ *for every first order variable* $X \in \mathcal{X}$, and $arity(F) + 1$ word variables $F_0^p, \ldots, F_{arity(F)}^p \in \mathcal{W}$ *for every occurrence* p *of a context variable* F in the problem (notice that in this case we use different word variables for every occurrence).

We guess a permutation ρ_p for any occurrence of a constant function f or of a context variable F, with arity greater or equal than two, in a position p of the problem.

We define the following translating function $\mathcal{T}$ that given a subterm t of the problem, and its position p, returns its translation in terms of words with variables $w \in (\Sigma_\Pi \cup \mathcal{W})^*$.

For any first-order constant a, or variable X,

$$\mathcal{T}(a, p) = a$$
$$\mathcal{T}(X, p) = X'$$

For every occurrence of the binary function symbol f at position p, let $w_i = \mathcal{T}(t_i, p \cdot i)$, and $\gamma_p \in \{f, f'\}$ be the symbol conjectured for this position, and ρ_p

the corresponding permutation for the arguments traversal order, then

$$\mathcal{T}(f(t_1,t_2),p) = \gamma_p\, w_{\rho_p(1)}\, w_{\rho_p(2)}$$

For every n-ary context variable F, occurring at position p, let $w_i = \mathcal{T}(t_i,\, p\cdot i)$, and ρ_p be the permutation conjectured for this position, then

$$\mathcal{T}(F(t_1,\ldots,t_n),p) = F_0^p\, w_{\rho_p(1)}\, F_1^p\, \cdots F_{n-1}^p\, w_{\rho_p(n)}\, F_n^p$$

Finally, the traversal system will contain the following equations:

1. A word equation for the original problem $t \stackrel{?}{=} u$

$$\mathcal{T}(t,1) \stackrel{?}{=} \mathcal{T}(u,2)$$

2a. For any two occurrences $F(t_1,\ldots,t_n)$ and $F(u_1,\ldots,u_n)$ of a context variable F at positions p and q, we introduce the following traversal equations and regular constraints:[3]

$$\begin{aligned}
&\mathcal{T}(F(a_1,\ldots,a_n),p) \equiv_b \mathcal{T}(F(a_1,\ldots,a_n),q)\\
&\mathcal{T}(F(a_1,\ldots,a_n),p) \in R^{k_1}_{\Sigma_\Pi}\\
&\mathcal{T}(F(a_1,\ldots,a_n),q) \in R^{k_2}_{\Sigma_\Pi}\\
&\mathcal{T}(F(b_1,\ldots,b_n),p) \equiv_b \mathcal{T}(F(b_1,\ldots,b_n),q)\\
&\mathcal{T}(F(b_1,\ldots,b_n),p) \in R^{k_1}_{\Sigma_\Pi}\\
&\mathcal{T}(F(b_1,\ldots,b_n),q) \in R^{k_2}_{\Sigma_\Pi}
\end{aligned}$$

where $b = \mathit{arity}(F)\cdot\Phi(|t \stackrel{?}{=} u|)$, $k_1 = k_2 = \Phi(|t \stackrel{?}{=} u|)$ and Φ is the computable function introduced in the rank-bound conjecture.

2b. In case we want to reduce context unification to (non-bound) traversal systems, we will introduce

$$\begin{aligned}
&\mathcal{T}(F(a_1,\ldots,a_n),p) \equiv \mathcal{T}(F(a_1,\ldots,a_n),q)\\
&\mathcal{T}(F(b_1,\ldots,b_n),p) \equiv \mathcal{T}(F(b_1,\ldots,b_n),q)
\end{aligned}$$

In this second case, we do not need the conjecture to fix b, k_1 and k_2.

The duplication of traversal equations with distinct constants a_i and b_i ensures that these constants occur in the place of the arguments. Otherwise, if we only introduce a traversal equation $F_0\, a\, F_1 \equiv F_0'\, a\, F_1'$, we can get solutions like $\sigma = [F_0 \mapsto f\, a][F_1 \mapsto \epsilon][F_0' \mapsto f][F_1' \mapsto a]$, that do not satisfy $\sigma(F_0\, b\, F_1) \equiv \sigma(F_0'\, b\, F_1')$, and leads to incompatible definitions of $\sigma(F) = \lambda x.\ f(a,x)$ and $\sigma(F) = \lambda x.\ f(x,a)$. ■

[3] We can avoid introducing a context variable occurrence in more than two traversal equation. If we have $p_1,\ldots,p_n$ occurrences of F, we can introduce an equation relating p_1 and p_2, p_2 and p_3,..., p_{n-1} and p_n.

Theorem 5.28 *If Context Unification is decidable, then the Rank-Bound Conjecture holds.*[4]

Proof: To get the computable function Φ such that, for any size n of the problem, gives us the bound on the rank for problems of that size, we just need to build a program that:

1. checks solvability of all (finite set) context unification problems of size n.
2. Then, by "dovetailing" on the size of the substitutions, checks if the generated substitution is a solution of the solvable problems, until it has found a unifier for each solvable problem.

The maximum rank of the set of solutions that it has found, serves as the bound on the rank of the solutions for context unification problems of size n. ■

Corollary 5.29 *Context Unification is decidable if, and only if, the Rank-Bound Conjecture holds.*

Example 5.30 To conclude, let us see how problem $F(G(a,b)) \stackrel{?}{=} G(F(a),b)$ could be translated into a traversal system.

We guess σ_0 equals the identity in the first step. In a second step, we introduce the word variables F_0, F_1, F_0', F_1' for the two occurrences of F, and $G_0, G_1, G_2, G_0',\ G_1', G_2'$ for the two occurrences of G. For both occurrences of G, the only symbol with arity 2 or greater, we have to guess their permutation $\rho_{1\cdot 1} = [1,2]$ and $\rho_2 = [2,1]$.

The translation of the unification problem results then into:

$$F_0\, G_0\, a\, G_1\, b\, G_2\, F_1 \stackrel{?}{=} G_0'\, b\, G_1'\, F_0'\, a\, F_1'\, G_2'$$

$$\begin{array}{ll} F_0\, a_1\, F_1 \equiv_c F_0'\, a_1\, F_1' & F_0\, b_1\, F_1 \equiv_c F_0'\, b_1\, F_1' \\ F_0\, a_1\, F_1 \in R^c_{\Sigma_\Pi} & F_0\, b_1\, F_1 \in R^c_{\Sigma_\Pi} \\ F_0'\, a_1\, F_1' \in R^c_{\Sigma_\Pi} & F_0'\, b_1\, F_1' \in R^c_{\Sigma_\Pi} \end{array}$$

$$\begin{array}{ll} G_0\, a_1\, G_1\, a_2\, G_2 \equiv_{2\cdot c} G_0'\, a_2\, G_1'\, a_1\, G_2' & G_0\, b_1\, G_1\, b_2\, G_2 \equiv_{2\cdot c} G_0'\, b_2\, G_1'\, b_1\, G_2' \\ G_0\, a_1\, G_1\, a_2\, G_2 \in R^c_{\Sigma_\Pi} & G_0\, b_1\, G_1\, b_2\, G_2 \in R^c_{\Sigma_\Pi} \\ G_0'\, a_2\, G_1'\, a_1\, G_2' \in R^c_{\Sigma_\Pi} & G_0'\, b_2\, G_1'\, b_1\, G_2' \in R^c_{\Sigma_\Pi} \end{array}$$

where $c = \Phi(8)$, and Φ is the function introduced by the rank-bound conjecture.

A solution of this permutation bound traversal system is the word substitution:

$$\begin{array}{lllll} \sigma = [& F_0 \mapsto f'\, b, & F_1 \mapsto \epsilon, & & \\ & F_0' \mapsto f, & F_1' \mapsto b & & \\ & G_0 \mapsto f, & G_1 \mapsto \epsilon & G_2 \mapsto \epsilon & \\ & G_0' \mapsto f', & G_1' \mapsto \epsilon & G_2' \mapsto \epsilon\ ] & \end{array}$$

[4] An anonymous referee of (Levy and Villaret, 2001) suggested us this result.

For the initial word equation $F_0\,G_0\,a\,G_1\,b\,G_2\,F_1 \stackrel{?}{=} G'_0\,b\,G'_1\,F'_0\,a\,F'_1\,G'_2$ we have:

$$\begin{array}{cccccccc} \overbrace{}^{F_0} & & & \overbrace{}^{G_0} & a & G_1 & b & G_2F_1 \\ \overbrace{f'}^{}\underbrace{}_{G'_0} & b & G'_1 & \overbrace{f}^{}\underbrace{}_{F'_0} & a & & \underbrace{b}_{F'_1} & G'_2 \end{array}$$

For the traversal equation of F, $F_0\,a_1\,F_1 \equiv_c F'_0\,a_1\,F'_1$ we have that $\sigma(F_0\,a_1\,F_1) = f'\,b\,a_1$ and $\sigma(F'_0\,a_1\,F'_1) = f\,a_1\,b$, and both are traversals of the same term $f(a_1, b)$ (similarly for $F_0\,b_1\,F_1 \equiv_c F'_0\,b_1\,F'_1$) .

Then, for the traversal equations of G, $G_0\,a_1\,G_1\,a_2\,G_2 \equiv_{2\cdot c} G'_0\,a_2\,G'_1\,a_1\,G'_2$ we have that $\sigma(G_0\,a_1\,G_1\,a_2\,G_2) = f\,a_1\,a_2$ and $\sigma(G'_0\,a_2\,G'_1\,a_1\,G'_2) = f'\,a_2\,a_1$, and both are traversals of the same term $f(a_1, a_2)$ (similarly for $G_0\,b_1\,G_1\,b_2\,G_2 \equiv_{2\cdot c} G'_0\,b_2\,G'_1\,b_1\,G'_2$).

The context substitution corresponding to the word substitution σ is the following substitution:

$$\sigma' = [F \mapsto \lambda x.\ f(x, b),\ G \mapsto \lambda xy.\ f(x, y)]$$

Then, the common instance of both sides of the equation is:

$$\sigma(F(G(a, b))) = f(f(a, b), b) = \sigma(G(F(a), b))$$

and notice that $\mathrm{NF}(f(f(a, b), b))$ is $f'\,b\,f\,a\,b$, the common instance of the word equation. ■

5.7 Some Hints in Favor of the Rank-Bound Conjecture

As we have repeatedly said, Conjecture 5.22 has not still been proved, and the decidability of Context Unification remains as an open question. However, we can present some hints reinforcing our opinion about the trueness of the conjecture. All these results have never been published before, and constitutes our main current research interest. As the reader will see most of the proofs of this section are only sketched, and the whole section constitutes an incomplete and unfinished work. We want to prevent the reader, who can consider this section as an appendix of the thesis pointing out further lines of research.

First, we reformulate Conjecture 5.22 in terms of relations on the sets of subterms of a term. This term is the common instance $\sigma(t)$ solving the unification problem $t \stackrel{?}{=} u$. We introduce a new kind of relations, called *sheaf relations*.

Definition 5.31 *A* sheaf *is a pair $\langle u = v, C\rangle$ such that $u = v$ is an equation between terms, and C is a context.*
Given a set of sheaves $S = \{\langle u_i = v_i, C_i\rangle\}_{i=1,\ldots,n}$, we define the sheaf relation *generated by it as the smaller equivalence relation $\approx$ such that, for any $i = 1, \ldots, n$, and any decomposition $C_i = C'_i(C''_i)$, we have $C''_i(u_i) \approx C''_i(v_i)$.*

The use of contexts in sheaf relations defines a kind of *restricted congruence* on the equations. We still define another restriction on the equivalence and congruence relation defined by a set of equations:

Definition 5.32 *Given a term t, and a set of equations $E = \{u_i = v_i\}_{i=1,\ldots,n}$ on subterms of t, we define the* equivalence and restricted to t congruence relation *generated by E as the minimal equivalence relation containing the relation $\approx'$, where $\approx'$ is defined as the minimal relation satisfying:*

1. *$u_i \approx' v_i$ for any equation, and,*
2. *if $u \approx' v$ and $f(s_1, \ldots, u, \ldots, s_m)$ and $f(s_1, \ldots, v, \ldots, s_m)$ are both subterms of t, then $f(s_1, \ldots, u, \ldots, s_m) \approx' f(s_1, \ldots, v, \ldots, s_m)$.*

This new relation is more general than the corresponding sheaf relation, but more restricted than the usual congruence and equivalence closure. This is stated more formally in the following fact.

Lemma 5.33 *Let t be a term, and $S = \{\langle u_i = v_i, C_i\rangle\}_{i=1,\ldots,n}$ a set of sheaves where, for any $i = 1, \ldots, n$, $C_i(u_i)$ and $C_i(v_i)$ are both subterms of t.*
Let $\approx_1$ be the sheaf relation generated by S, let $\approx_2$ be the equivalence and restricted to t congruence relation generated by $E = \{u_i = v_i\}_{i=1,\ldots,n}$, and let $\approx_3$ be the minimal equivalence and congruence relation generated by E (without restrictions).
Then $\approx_1 \subseteq \approx_2 \subseteq \approx_3$.

The converse inclusions do not hold, in general, as the following example proves.

Example 5.34 Let $t = g(f(a), f(b), c, f(d))$, and $S = \{\langle a = b, \bullet\rangle, \langle b = c, \bullet\rangle, \langle c = d, \bullet\rangle\}$. We have

$$\begin{array}{c} f(a) \approx_3 f(b) \approx_3 f(d) \\ f(a) \approx_2 f(b) \not\approx_2 f(d) \\ f(a) \not\approx_1 f(b) \not\approx_1 f(d) \end{array}$$

■

The distinction between $\approx_2$ and $\approx_3$ could seem negligible, but it is extremely important. In fact, dropping out it would lead us to a fake proof of Conjecture 5.22!! (see Remark 5.43).

We are interested on equivalence relations on subterms of t, defining a unique equivalence class of subterms.

Definition 5.35 *Given a term t, and an equivalence relation $\approx$ on subterms of t, we say that $\approx$* collapses *t if $\approx$ relates every pair of subterms of t.*
Given a term t we say that it has at most n independent subterms, *if, for every set $\{u_1, \ldots, u_{n+1}\}$ of subterms of t, there exists at least two of them, u_i and u_j, such that $i \neq j$ and either u_i is subterm of u_j or vice-versa.*

Then, we can reformulate Conjecture 5.22 as follows.

Conjecture 5.36 *There exists a computable function* Φ *such that, for any term* t*, if there exists a sheaf relation* $\approx$ *generated by a set of* n *sheaves, and collapsing* t*, then* $\text{rank}(t) \leq \Phi(n)$.

Conjecture 5.37 *There exists a computable function* Φ *such that, for any term* t*, if there exists an equivalence and restricted to* t *confluence relation* $\approx$ *generated by a set of* n *equations, and collapsing* t*, and* t *has at most* n *independent subterms, then* $\text{rank}(t) \leq \Phi(n)$.

Theorem 5.38 *If Conjecture 5.37 is true, then Conjecture 5.36 is also true. If Conjecture 5.36 is true, then Conjecture 5.22 is also true.*

Proof: The first implication is a direct consequence of Lemma 5.33, and the fact that all subterms of t must be related by some of the sheaves, so there can be as many independent subterms as sheaves.

The second implication requires a longer proof. Here, we only sketch some of the main ideas in this proof. Let σ be a most general unifier of $t \stackrel{?}{=} u$. Let ρ be a substitution instantiating, fresh first-order variables introduced by σ, by a constant a, and fresh n-ary context variables by some size-minimal term $\lambda x_1 \cdots x_n.\, f(x_1, \ldots f(x_{n-1}, x_n) \ldots)$. Now, we will prove that we can collapse the term $\rho(\sigma(t))$ using a number of sheaves bounded on the size of $t \stackrel{?}{=} u$.

A most general unifier σ of a context unification problem $t \stackrel{?}{=} u$ defines a morphism d from positions in t to positions in $\sigma(t)$, and a similar morphism d' for u. The positions of $\sigma(t)$ with inverse for such morphism are usually called *cut* positions in the literature related with word unification.

Let F be a context variable with k occurrences in t and u, we can relate positions inside the occurrences of $\sigma(F)$ using k sheaves. We have to add now some sheaves to relate the cut positions, but they are bounded by the size of the unification problem. Finally, we have to relate the positions of the fresh variables introduced by σ. This can be avoided for the first-order variables if we instantiate them by the same constant (as ρ does). For the n-ary variables, we can do it using as many sheaves as the sum of their arities, that also turns to be bounded on the size of the unification problem. ■

Although Conjecture 5.36 and Conjecture 5.37 have not been proved, we have been able to prove some variants of them (see Theorems 5.41 and 5.47). We think that these proofs merits to be included in this thesis, as a justification of our belief on the trueness of Conjecture 5.22.

5.7.1 First Hint

The first hint we present is the proof of a variant of Conjecture 5.37. Roughly speaking, it states that, if we can orient the rules defining the equivalence and congruence relation, obtaining a rank-decreasing and confluent rewrite system, then the conjecture is true.

Assume for the rest of the section that $R = \{u_i \to v_i\}_{i=1,\ldots,n}$ is a ground term rewriting system such that it is rank decreasing, i.e. $\mathrm{rank}(u_i) \geq \mathrm{rank}(v_i)$, for any $i = 1, \ldots, n$, terminating and confluent. We prove the following Lemma:

Lemma 5.39 *Let u rewrites to v using the above mentioned rewriting system in n steps, and let v be a subterm of u, then there exists a subterm s of u, and a subterm s' of the left-hand side of some rule $u_i \to v_i \in R$ used in $u \to^* v$, such that $\mathrm{rank}(s) = \mathrm{rank}(u) - 1$ and s rewrites to s' in $\leq n$ steps.*

Proof: Let p be the position of some occurrence of v in u, i.e. $u|_p = v$. By the properties of the rank, there exist two subterms $u|_q$ and $u|_{q'}$ of u such that $\mathrm{rank}(u|_q) = \mathrm{rank}(u|_{q'}) = \mathrm{rank}(u) - 1$, and neither q is a prefix of q', nor vice-versa. Therefore, at least, one of them, say q, is not a prefix of $p \cdot p \cdots p$ nor vice-versa, for arbitrarily long sequences $p \cdot p \cdots p$. Let be $s = t|_q$.

Since u rewrites to v in n steps, $u|_q$ rewrites to $v|_q$ in $\leq n$ steps, or $u|_q$ rewrites to some subterm of the left-hand side of some rule u_i. In the second case, we are finish. In the first case, since $u|_p = v$ we have $v|_q = u|_{p \cdot q}$ and we can repeat the same reasoning. The term $u|_{p \cdot q}$ rewrites to $v|_{p \cdot q}$, or to some subterm of the left-hand side of some rule. In this way, we can construct a rewriting sequence of the form:

$$u|_q \to^* v|_q = u|_{p \cdot q} \to^* v|_{p \cdot q} = u|_{p \cdot p \cdot q} \to^* v|_{p \cdot p \cdot q} \to^* \cdots$$

Since the rewriting system is terminating this sequence can not be infinite. All the rewriting steps of this sequence correspond to *distinct* rewriting steps of the rewriting sequence $u \to^* v$. More precisely, for any rewriting step $s_1 \to s_2$ of these sequence, there is a distinct rewriting step $s'_1 \to s'_2$ in $u \to^* v$, such that $s_1 = s'_1|_{p^m \cdot q}$ and $s_2 = s'_2|_{p^m \cdot q}$, for some $m \geq 0$. (Notice that for any $n \neq m$, $p^n \cdot q$ is not a prefix of $p^m \cdot q$, nor vice-versa). Therefore the length of this rewriting sequence is $\leq n$, as stated by the Lemma. ■

The following Lemma states how the rank of a term decreases when we rewriting it.

Lemma 5.40 *Let R be a rank-decreasing rewriting system. If $u \to_R v$ in one step, then either $\mathrm{rank}(u) - 1 \leq \mathrm{rank}(v) \leq \mathrm{rank}(u)$ or there exists a rewriting rule $u_i \to v_i \in R$ such that $\mathrm{rank}(u_i) = \mathrm{rank}(u)$.*

Proof: By the properties of the rank, u contains two independent subterms $u|_p$ and $u|_q$ with $\mathrm{rank}(u|_p) = \mathrm{rank}(u|_q) = \mathrm{rank}(u) - 1$. Then either the redex of the rewriting step contains both subterms, or at least one of them remains unchanged after the rewriting step. In the first case, the redex has rank equal to $\mathrm{rank}(u)$. In the second case, the rank of v is at least $\mathrm{rank}(u) - 1$, since it contains the unchanged subterm of such rank, and at most $\mathrm{rank}(u)$, since the rewriting step is rank-decreasing. ■

Now, we can state the variant of Conjecture 5.37.

Theorem 5.41 *For any term t, if there exists a rank-decreasing, terminating, and confluent rewriting system $R = \{u_i \to v_i\}_{i=1,\ldots,n}$ collapsing t, and t has at most n independent subterms, then* $\mathrm{rank}(t) \leq (n+1)^2$.

Proof: Since the rewriting system is terminating and confluent, and it relates (collapses) all subterms of t, all subterms of t can be rewritten into the *same* normal form. Now, since the rewriting system is also rank-decreasing, the normal form has to be a subterm of rank zero, i.e. a 0-ary constant. For any $i = 1, \ldots, n$, let $r_i = \mathrm{rank}(u_i)$ and $r_{n+1} = \mathrm{rank}(t)$. Assume without loss of generality that $r_1 \leq \cdots \leq r_n \leq r_{n+1}$. Notice that $r_1 \leq 1$ (otherwise, terms of rank 1 could not be rewritten into *the* normal form).

If $r_{i+1} - r_i \leq n+2$, for all $i = 1, \ldots, n$, then the proof is done.

Otherwise, there exists some $i = 1, \ldots, n$, such that $r_{i+1} - r_i > n+2$. Notice that t contains subterms of any rank between 0 and r_{n+1}. Consider a subterm u of t with $\mathrm{rank}(u) = r_{i+1} - 1$. By Lemma 5.40, any rewriting sequence $u = u_1 \to \cdots \to u_{n+1} \to \cdots$ from u to the normal form satisfies $\mathrm{rank}(u_i) - 1 \leq \mathrm{rank}(u_{i+1}) \leq \mathrm{rank}(u_i)$, for $i = 1, \ldots, n$, i.e. during the n first steps, the rank only decreases in at most one unity. Therefore, we have $\mathrm{rank}(u_i) \geq r_{i+1} - i$, for $i = 1, \ldots, n+1$. Now, since there are at most n independent subterms, there exist $1 \leq i < j \leq n+1$ such that u_j is a subterm of u_i. By Lemma 5.39, there exist some subterm v of u_i with $\mathrm{rank}(v) = \mathrm{rank}(u_i) - 1$ that rewrites to some subterm s of the left-hand side of some rule, in $\leq j - i$ steps. We have $\mathrm{rank}(v) = \mathrm{rank}(u_i) - 1 \geq r_{i+1} - i - 1$. Again, by Lemma 5.40, in $\leq j - i$ steps, the rank can decrease in at most $j - i$ unities and we get a term with $\mathrm{rank}(s) \geq r_{i+1} - i - 1 - (j - i) = r_{i+1} - j - 1$. On one hand, since $j \leq n+1$, we have $\mathrm{rank}(s) \geq r_{i+1} - n - 2 > r_i$. But, on the other hand, s is a subterm of the left-hand side of some rewriting rule used in the rewriting sequence $u_1 \to^* u_{n+1}$, therefore $\mathrm{rank}(s) \leq r_i$. This contradicts the assumption $r_{i+1} - r_i > n+2$. ■

Remark 5.42 *The rank of a term respects the subterm relation, i.e. if u is a subterm of t, then* $\mathrm{rank}(u) \leq \mathrm{rank}(t)$. *This means that, given a set of ground equations $E = \{u_i = v_i\}_{i=1,\ldots,n}$, it is always possible to orient them to obtain a terminating ant rank-decreasing rewriting system $R = \{u_i \to v_i\}_{i=1,\ldots,n}$, using, for instance a Knuth-Bendix ordering.*

Remark 5.43 *Given a terminating and rank-decreasing rewriting system $R = \{u_i \to v_i\}_{i=1,\ldots,n}$, it is always possible to obtain an equivalent rewriting system R' such that R' is, apart from rank-decreasing and terminating, confluent. We can apply, for instance, a Knuth-Bendix completion process where, every time we find a non-confluent critical pair $t[v_2] \leftarrow t[u_2] = u_1 \to v_1$, we replace the rule $u_1 \to v_1$ by either $t[v_2] \to v_1$ or $v_1 \to t[v_2]$, depending on the rank-decreasing ordering. This completion process always terminates and does not increase the number of rewriting rules.*

From these remarks, one could be tempted to infer that, once we have a set of ground equations $E = \{u_i = v_i\}_{i=1,\ldots,n}$, we can obtain a rewriting system accomplishing the conditions of Theorem 5.41 and prove Conjecture 5.37. Un-

fortunately, these remarks only apply to the usual congruence and equivalence relations, but not to the restricted *to t congruences considered in Conjecture 5.37.*

5.7.2 Second Hint

The quadratic bound stated by Theorem 5.41 can be reduced to $\mathcal{O}(n \log n)$. First, we define the *rational rank* of a term as follows:

$$\begin{aligned} \mathrm{rrank}(a) &= 0 \\ \mathrm{rrank}(f(t,u)) &= \max \begin{cases} \mathrm{rrank}(t) \\ \mathrm{rrank}(u) \\ \dfrac{\mathrm{rrank}(t) + \mathrm{rrank}(u)}{2} + 1 \end{cases} \end{aligned}$$

The rational rank and the usual rank do not differ in more than one unity:

$$\mathrm{rank}(t) \leq \mathrm{rrank}(t) \leq \mathrm{rank}(t) + 1$$

Using the rational rank it is possible to narrow the bounds stated in Lemma 5.40, and obtain the bound annunciated.

In the sequel, we prove a variant of Conjecture 5.36 for a measure of terms, called *average depth*, and inspired in the rational rank.

Definition 5.44 *The* average depth *of a term is defined recursively as follows:*

$$\begin{aligned} \mathrm{adepth}(a) &= 0 \\ \mathrm{adepth}(f(t,u)) &= \frac{\mathrm{adepth}(t) + \mathrm{adepth}(u)}{2} + 1 \end{aligned}$$

Like in the case of rank, any term contains subterms of any average depth between zero and the total average depth of the term.

Lemma 5.45 *For any term t, and any integer $n \leq \mathrm{adepth}(t)$, there exists a subterm u of t such that*

$$n \leq \mathrm{adepth}(u) < n + 1$$

Proof: By structural induction on t. If $t = f(t_1, t_2)$, for some $i = 1, 2$, we have $\mathrm{adepth}(t_i) \geq \mathrm{adepth}(t) - 1$. Now, if $n \leq \mathrm{adepth}(t_i)$, apply the induction hypothesis to t_i, and take the u given in such case. Otherwise, from $\mathrm{adepth}(t) - 1 \leq \mathrm{adepth}(t_i) < n \leq \mathrm{adepth}(t)$, we can conclude $n = \lfloor \mathrm{adepth}(t) \rfloor$, and take $u = t$. ■

The following is a technical lemma:

Lemma 5.46 *Let $G = (V, E)$ be a connected graph, where $V \subset \mathbb{R}$. The number of intervals of the form $[n, n + 1)$, for some integer number n, beaten by G is*

bounded on twice the number of edges with length longer than 1 plus the sum of the lengths of the rest of edges.
Formally, if the length of an edge is the difference between its extremes,

$$|\{\lfloor n \rfloor \mid n \in V\}| \;\leq\; 2\,|\{e \in E \mid \text{length}(e) > 1\}| + \sum_{\substack{e \in E \\ \text{length}(e) \leq 1}} \text{length}(e)$$

Theorem 5.47 *For any term t, if there exists a sheaf relation $\approx$ generated by a set of n sheaves collapsing t, then* $\text{adepth}(t) \leq n^2$.

Proof: Let $S = \{\langle u_i = v_i, C_i\rangle\}_{i=1,\ldots,n}$ be the set of sheaves. Let $G = (V, D)$ be the graph defined by the set of vertexes

$$V = \{\text{adepth}(C_i''(u_i)), \text{adepth}(C_i''(v_i)) \mid \langle u_i = v_i, C_i\rangle \in S \wedge C_i = C_i'(C_i'')\}$$

and the set of edges

$$E = \{\langle \text{adepth}(C_i''(u_i)), \text{adepth}(C_i''(v_i))\rangle \mid \langle u_i = v_i, C_i\rangle \in S \wedge C_i = C_i'(C_i'')\}$$

If $\approx$ is collapsing, then G is connected.

Let d be the maximal average depth of some subterm of t. The values of V are bounded on d.

Let $l_i = |\,\text{adepth}(u_i) - \text{adepth}(v_i)|$. The sheaf $\langle u_i = v_i, C_i\rangle$ generates an edge of length l_i, another of length $l_i/2$,... until another of length $l_i/2^{k_i}$, being k_i the depth of the argument in C_i. Now, since $l_i \leq d$, this sheaf generates at most $\log(d) + 1$ edges longer than one. The sum of the lengths of the rest of edges generated by this sheaf is smaller than 2. Using Lemma 5.46, we can conclude that the number of intervals beaten by the average depths of the sheaves is bounded by $2\,n\,(\log(d) + 2)$.

On the other hand, according to Lemma 5.45, if t contains a subterm with average depth d, then it contains subterms with average depth beating any interval $[n, n+1[$ with $n \leq d$. We have to beat $\lfloor d \rfloor$ of such intervals. Therefore

$$2\,n\,(\log(d) + 2) \geq d - 1$$

This inequality implicitly defines a bound of d in terms of n. In particular, we can conclude $d \leq n^2$ for large enough values of n. ∎

Remark 5.48 *Unfortunately, a bound in the average depth of a term does not imply a bound on its rank. For instance, the infinite rational term generated by the recursion $t = f(f(t, b), f(b, t))$ has* $\text{adepth}(t) = 4$ *and* $\text{rank}(t) = \infty$.
We can generalise the definition of average depth to

$$\text{adepth}(f(t, u)) = \frac{\text{adepth}(t) + \text{adepth}(u)}{k} + 1$$

for any value $k > 1$. We obtain then a result similar to Theorem 5.47 that suggest us that, if Conjecture 5.22 turns to be false, then counter-examples must be some form of "sparse" terms, i.e. terms with small generalized average depth.

5.8 Summary

In this chapter we have proved that, if the rank-bound conjecture is true, then Context Unification is decidable. This rank-bound conjecture requires the existence of a computable function Φ that allows us to ensure that if the given Context Unification problem P is solvable then it has a solution whose rank does not exceed $\Phi(|P|)$. This measure is non-trivial because a bound on the rank does not imply any bound on the size.

The existence of such a computable function would allow us to reduce solvability of context equations to solvability of word equations with regular constraints, the solvability of which is decidable.

Our opinion is that, knowing how hard it has been to prove Word Unification decidability and knowing also how closely related Word Unification is to Context Unification, this reduction helps to avoid some of the intrinsic complexity in the proof of Context Unification decidability.

This reduction requires several encoding steps and has led us to define *traversal equations* and *rank and permutation-bound traversal equations*. We have also proved that solvability of the latter is decidable.

Chapter 6

From Linear Second-Order Unification to Context Unification with Tree-Regular Constraints

Linear Second-Order Unification and Context Unification are closely related problems. However, the equivalence between both problems has not been proved. Context Unification can be defined as a restriction of Linear Second-Order Unification. In this chapter we prove that Linear Second-Order Unification can be reduced to Context Unification with tree-regular constraints. We also show that *rank-bound* tree-regular constraints can be reduced to word-regular constraints. These two results, together with the results of the previous chapter, suggest us to comment on the possibility that Linear Second-Order Unification is decidable, if Context Unification is.

The work of this chapter is based on (Levy and Villaret, 2000).

6.1 Introduction

Context Unification can be defined as a restriction on Linear Second-Order Unification where third- or higher-order constants are not allowed, there are no internal λ-bindings, and the external ones are only used to denote the parameters of a second-order variable. The common belief was that third- or higher-order constants do not play an important role with respect to the decidability of both problems, neither the use of λ-bindings. However, the equivalence of both problems has never been proved.

The naive attempt to reduce Linear Second-Order Unification to Context Unification by replacing bound variables by new constant symbols does not work. This is because we have to ensure that substitutions avoid *variable capture*. For

instance, the following linear second-order unification problem:

$$\lambda x.\ f(x) \stackrel{?}{=}_{lsou} \lambda x.\ f(Y)$$

is not solvable[1]. However, applying the naive reduction to this problem we get the following context unification problem:

$$f(c_x) \stackrel{?}{=}_{cu} f(Y)$$

that is obviously solvable, being the substitution $[Y \mapsto c_x]$ a unifier.

We can try to apply a more sophisticated reduction. Take the original linear second-order unification problem and replace the bound variables by two distinct constants in two equations (instead of just one). This would avoid that the instance of a free variable could contain constants corresponding to a bound variable translation.

However, this method only works for the most external λ-bindings. Applying the reduction to the following solvable linear second-order unification problem with internal λ-bindings:

$$f(g(\lambda x.\ x), a) \stackrel{?}{=}_{lsou} f(Y, Z)$$

we get the following unsolvable context unification problem:

$$\begin{array}{l} f(g(c_x), a) \stackrel{?}{=}_{cu} f(Y, Z) \\ f(g(c'_x), a) \stackrel{?}{=}_{cu} f(Y, Z) \end{array}$$

whereas the original problem has this unifier $[Y \mapsto g(\lambda x.\ x), Z \mapsto a]$.

Bindings can transform free variables into bound variables at different depths. Somehow we have to ensure that if an instance of a variable (unknown) contains a bound variable, then it also contains its corresponding λ-binding. For instance, given the linear second-order unification problem $F(X) \stackrel{?}{=}_{lsou} g(\lambda y.\ y, a)$ and the following substitutions:

$$\begin{array}{lllll} \sigma_1 = [& X \mapsto a, & F \mapsto \lambda z.\ g(\lambda y.\ y, z) &] \\ \sigma_2 = [& X \mapsto y, & F \mapsto \lambda z.\ g(\lambda y.\ z, a) &] \\ \sigma_3 = [& X \mapsto g(\lambda y.\ y, a), & F \mapsto \lambda z.\ z &] \end{array}$$

only σ_1 and σ_3 are unifiers. As we will show, such a restriction can be ensured by means of tree automata (Comon *et al.*, 1997), but it does not seem easy to be simply encoded in terms of context equations.

On the other hand, we have also shown in Subsection 3.3.1 that Context Unification and Word Unification are also closely related problems. Word Unification with regular restrictions is decidable (Schulz, 1991). Tree-regular languages are to terms the same as regular languages are to words. Therefore, if Context Unification turns out to be decidable, then it seems reasonable to

[1] The substitution $\sigma = [Y \mapsto x]$ gives us: $\sigma(\lambda x.\ f(x)) = \lambda x.\ f(x) \neq_\lambda \lambda y.\ f(y) = \sigma(\lambda x.\ f(Y))$ but both terms are not λ-equivalent, because an α-conversion is needed in order to avoid the capture of variable x.

think that Context Unification could be enriched with tree-regular restrictions without losing decidability. To support this hypothesis, we would like to prove that membership equations on tree-regular languages can be reduced to membership equations on (word) regular languages, by encoding terms as traversal sequences. Unfortunately, we can only prove this reduction for a certain subset of tree-regular languages (what we call *rank-bound tree-regular languages*).

We have conjectured in the previous chapter that whenever a context unification problem is solvable, it has a rank-bound unifier. Unfortunately, it is not clear what happens with Context Unification with tree-regular constraints problems, and hence it is not known what could happen with Linear Second-Order Unification decidability.

This chapter proceeds as follows. We reduce Linear Second-Order Unification to Context Unification with tree-regular constraints in Section 6.3. In Section 6.4 we reduce rank-bound tree-regular restrictions to word-regular restrictions. In Section 6.5 we discuss the effects that this reduction could have in terms of decidability for Linear Second-Order Unification. Finally, in Section 6.6 we discuss whether this results could be extended to Linear Third- or Higher-Order Unification.

6.2 Preliminaries

Firstly we are going to define what are *tree automata* and *tree-regular languages*. The main reference of this topic is (Comon *et al.*, 1997), and a pioneer work in relating tree-automata and unification or matching is the work of Comon and Jurski (1997).

Definition 6.1 *A* non-deterministic finite tree automaton *(*tree automaton *for short) $\mathcal{A}$ is a tuple (Σ, Q, Q_f, Δ) where:*

- *Σ is a finite signature,*
- *Q is a finite set of* states*,*
- *$Q_f \subseteq Q$ is the set of* final states*,*
- *Δ is a* transition relation*: it is a finite set of rules of the form:*

$$f(q_1, \ldots, q_n) \to q$$

where $f \in \Sigma$ is n-ary and $q_1, \ldots, q_n \in Q$.

Definition 6.2 *A tree automaton $\mathcal{A}$* accepts *(respectively accepts in state q) a term $t \in \mathcal{T}(\Sigma, \emptyset)$ and a state $q_f \in Q_f$ such that $t \to_\Delta^* q_f$ (respectively $t \to_\Delta^* q$). The* language $\mathcal{L}(\mathcal{A})$ *recognised by a tree automaton $\mathcal{A}$ is the set of terms which are accepted by $\mathcal{A}$.*
A language L is said to be tree-regular *if there exists a tree automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = L$.*

Example 6.3 Consider the automaton $\mathcal{A}$ defined by:

$$\begin{array}{lcll} (& \Sigma & = & \{cons(,), s(), 0, nil\} & , \\ & Q & = & \{q_N, q_L\} & , \\ & Q_f & = & \{q_L\} & , \\ & \Delta & = & \{nil \to q_L, cons(q_N, q_L) \to q_L, & \\ & & & 0 \to q_N, s(q_N) \to q_N\} &) \end{array}$$

This automaton recognises the set of Lisp-like lists of naturals. For instance:

$$cons(0, cons(s(0), cons(s(s(0)), nil)))$$

■

Theorem 6.4 *[Comon* et al. *(1997)] Given a term t and a tree automaton $\mathcal{A}$, the membership question $t \in \mathcal{L}(\mathcal{A})$ is decidable in polynomial time.*

Theorem 6.5 *[Comon* et al. *(1997)] The class of recognisable tree languages is closed under union, under intersection and under complementation.*

As we have done in the introduction of the chapter, in the rest of the chapter, willing to help on readability, we will denote linear second-order equations with the symbol $\stackrel{?}{=}_{lsou}$ and context equations with $\stackrel{?}{=}_{cu}$.

Definition 6.6 *A context unification problem with tree-regular constraints is a context unification problem with a tree-regular constraint $v \in \mathcal{A}$.*[2] *A solution is a ground substitution σ solving the context unification problem, and satisfying $\sigma(v) \in \mathcal{L}(\mathcal{A})$.*

Fact 6.7 *The solvability of the set of tree-regular constraints $\{t_1 \in \mathcal{L}(\mathcal{A}_1), \cdots, t_n \in \mathcal{L}(\mathcal{A}_n)\}$ is equivalent to the solvability of $f(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{A})$ where $\mathcal{A}$ has a copy of each $\mathcal{A}_i$ plus a transition $f(q_{f_1}, \ldots, q_{f_n}) \to q_f$ for any possible combination of final states of $\mathcal{Q}_{f_1}, \ldots, \mathcal{Q}_{f_n}$. Then, $\mathcal{Q}_f$ of $\mathcal{A}$ is simply $\{q_f\}$.*

Fact 6.8 *A linear second-order unification problem is solvable if and only if, properly extending the signature, there exists a ground unifier.*

Proof: We only need the existence of a 0-ary constant and a constant of arity greater or equal to 2. In Linear Second-Order Unification this does not affect the unifiability of problems whereas in Context Unification it does as we have already seen in Subsection 3.2.3. ■

For simplicity we assume that the signature of the problem is finite and allows us to ensure the existence of a ground solution whenever a solution exists. Recall that this fact can always be guaranteed if we extend the signature ensuring that it contains at least a 0-ary constant for any base type and a binary function.

Sometimes we will denote λ-abstractions as $\lambda\vec{x}.\ t$, where $\vec{x}$ denotes a list of bound variables.

[2]Notice that a set of tree-regular constraints is equivalent to just one constraint, (see Fact 6.7).

6.3 Reducing Linear Second-Order Unification to Context Unification with Tree-Regular Constraints

In this section, we prove that Linear Second-Order Unification can be reduced to Context Unification with tree-regular constraints. We reduce the linear second-order unification problem to a context unification problem by removing λ-bindings and constants with order higher than two, and adding tree-regular constraints to ensure that unifiers correctly treat bound variables occurring in the problem. Recall that in second-order languages, λ-bindings of normal terms are always just below higher-order constants or bound variables, or are the most external symbol. They are never just below free variables.

Definition 6.9 *Let the signature be Σ and the set of variables be $\mathcal{X}$, the translation will be performed firstly removing external λ-bindings as a sort of* preprocess*, and then removing internal λ-bindings and higher-order constants in a* three-step process*:*

Preprocess, *we can eliminate external λ-bindings by extending the signature Σ with an appropriate new unary constant o (if it does not contain any) and translating the equation $\lambda\vec{x}.\ s \stackrel{?}{=}_{lsou} \lambda\vec{y}.\ t$ into $o(\lambda\vec{x}.\ s) \stackrel{?}{=}_{lsou} o(\lambda\vec{y}.\ t)$. This new problem does not have external λ-bindings and is equivalent to the original one.*

First, *we conjecture an α-conversion of bound variables in order to allow unification when they are later translated into constants in the following step. Notice that the second step of this translation procedure depends on the "names" of these bound variables.*

Second, *let $B \subset \mathcal{X}$ be a finite set of variables and let $A \subset \mathcal{L}istsof\,(B)$ be a finite set of lists of variables from B. We define a translation function $trans^{A,B}$ that replaces any occurrence of a variable from B by a new first-order constant, and any occurrence of a λ-abstraction, whose list of bound variables is in A, also by a new constant. This set B will be the set of bound variables of the unification problem resulting from step 1, and A will be the set of lists of variables used in the λ-abstractions.*
The signature Σ' of the resulting context unification problem also depends on the set B of bound variables conjectured in the previous step, and on the set A of lists of variables of the λ-abstractions. It is defined as follows: Σ' contains the same constants as Σ, but every constant h or bound variable z, with order higher than two, is replaced in Σ' by a new second-order constant h' or c_z respectively. The arity of h' (similarly for c_z) is equal to the arity of h plus its number of non-first-order arguments. Any non-first-order n-ary argument of h with type $\tau_1 \to \cdots \to \tau_n \to \tau_0$ is replaced by two first-order arguments (one for the λ-abstraction and one for the body). For instance, if $h : \iota \to (\iota \to \iota) \to \iota$ then

$h' : \iota \to \iota \to \iota \to \iota$. *The signature* Σ' *also contains a new constant symbol* $b_{[x_1,...,x_n]}$, *for every list* $[x_1, ..., x_n] \in A$, *and a new constant symbol* c_x, *for every variable* $x \in B$. *The set of variables of the resulting problem is* $\mathcal{X}' = \mathcal{X} \backslash B$.
Let $t \in T(\Sigma, \mathcal{X})$ *be a term,* $B \subset \mathcal{X}$ *a finite set of variables, and* $A \subset \mathcal{L}istsof\,(B)$ *a finite set of lists of variables from* B. *The term* $trans^{A,B}(t) \in T(\Sigma', \mathcal{X}')$ *is defined by:*

1. $trans^{A,B}(c) = c$
2. $trans^{A,B}(f(t_1, ..., t_n)) = f(trans^{A,B}(t_1), ..., trans^{A,B}(t_n))$
3. $trans^{A,B}(X) = \left\{ \begin{array}{ll} X & \text{if } X \notin B \\ c_X & \text{if } X \in B \end{array} \right)$
4. $trans^{A,B}(F(t_1, ..., t_n)) = \left\{ \begin{array}{ll} F(trans^{A,B}(t_1), ..., trans^{A,B}(t_n)) & \text{if } F \notin B \\ c_F(trans^{A,B}(t_1), ..., trans^{A,B}(t_n)) & \text{if } F \in B \end{array} \right.$
5. $trans^{A,B}(\ h(t_1, ..., t_n, \lambda\vec{x_1}.\ u_1, ..., \lambda\vec{x_m}.\ u_m)\ ) =$
   $= h'(trans^{A,B}(t_1), ..., trans^{A,B}(t_n), b_{\vec{x_1}}, trans^{A,B}(u_1), ..., b_{\vec{x_m}}, trans^{A,B}(u_m))$
6. $trans^{A,B}(\ z(t_1, ..., t_n, \lambda\vec{x_1}.\ u_1, ..., \lambda\vec{x_m}.\ u_m)\ ) =$
   $= c_z(trans^{A,B}(t_1), ..., trans^{A,B}(t_n), b_{\vec{x_1}}, trans^{A,B}(u_1), ..., b_{\vec{x_m}}, trans^{A,B}(u_m))$
7. $trans^{A,B}(\lambda\vec{x}.\ t) = \lambda\vec{x}.\ trans^{A',B\backslash\vec{x}}(t)$

In the fifth and sixth case, for constants h *and variables* z *with order higher than two, we have assumed, for simplicity, that non-first-order parameters are in the last positions. The constant* h' *is the second-order constant associated to* h, c_z *is the constant associated to the variable* z, *and* $b_{\vec{x_i}}$ *is the constant associated to the list of variables* $\vec{x_i} \in A$ *of the* λ*-binding* $\lambda\vec{x_i}$. *If, for some* $i \in \{1..m\}$, $\vec{x_i} \notin A$, *then the translation is undefined. In the latter case,* A' *is the set of lists* A *where any list containing variables from* $\vec{x}$ *has been removed. Notice that most external* λ*-bindings are not removed by this translation, hence it behaves as we may expect when applied to unifiers.*

Third, *we introduce a set of tree-regular restrictions over the instantiations of variables to prevent them from containing constants associated to bound variables from* B *without its corresponding* λ*-bindings. The tree automaton* $\mathcal{A}^{A,B} = \langle \Sigma', \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *that characterises the set of linear terms that do not contain these bound variables from* B *in* free positions *is defined as follows.*

- *The signature contains the set of constants* Σ'. *Notice that it is finite because* Σ, B *and* A *are finite. Remember that* Σ' *allows us to ensure that, if a certain context unification problem* S *is solvable then,* S *has a ground solution.*
- *The set of states is* $\mathcal{Q} = \{q_X | X \subseteq B\} \cup \{p_X | X \in A\}$; *where* B *is the (finite) set of bound variables of the problem and* A *is the (finite) set of lists of bound variables of the* λ*-bindings of the problem.*

- *There is a single final state* $\mathcal{Q}_f = \{q_\emptyset\}$
- *The set of transitions* Δ *is defined as follows:*
  - *For any first-order constant* $a \in \Sigma'$ *not associated to a variable from* B*:*
    $$a \to q_\emptyset$$
  - *For any first-order constant* c_x *associated to a bound variable* $x \in B$ *and any first-order constant* $b_{\vec{y}}$ *associated to a list of bound variables* $\vec{y} \in A$*:*
    $$c_x \to q_{\{x\}}$$
    $$b_{\vec{y}} \to p_{\vec{y}}$$
  - *For any second-order constant* $f \in \Sigma'$ *not associated to a variable from* B*, and states* $q_{A_1}, \ldots, q_{A_n}$*:*
    $$f(q_{A_1}, ..., q_{A_n}) \to q_{A_1 \cup ... \cup A_n}$$
    provided for all $i \neq j \in \{1..n\}, A_i \cap A_j = \emptyset$*.*
    And, for any constant c_x *associated to a bound variable* $x \in B$*, and states* $q_{A_1}, \ldots, q_{A_n}$*:*
    $$c_x(q_{A_1}, ..., q_{A_m}) \to q_{A_1 \cup ... \cup A_n \cup \{x\}}$$
    provided for all $i \neq j \in \{1..n\}, A_i \cap A_j = \emptyset$*, and* $x \notin A_i$*.*[3]
  - *For any second-order constant* $h' \in \Sigma'$ *associated to a higher-order constant* $h \in \Sigma$*, and states* $q_{A_1}, \ldots, q_{A_n}, p_{B_1}, q_{C_1}, ..., p_{B_m}, q_{C_m}$*:*
    $$h'(q_{A_1}, ...q_{A_n}, p_{B_1}, q_{C_1}, ..., p_{B_m}, q_{C_m}) \to q_D$$
    where for $i \in \{1..m\}$*, let* $A_{n+i} = (C_i \setminus B_i)$ *and*
    $$D = \bigcup_{i \in \{1..n+m\}} A_i$$
    provided for all $i \in \{1..m\}$*, all variables occurring in* B_i *also occur in* C_i*, and for all* $j \neq k \in \{1..(n+m)\}, A_j \cap A_k = \emptyset$*.*
    And for any second-order constant $c_z \in \Sigma'$ *associated to a higher-order bound variable* $z \in B$*, and states* $q_{A_1}, \ldots, q_{A_n}, p_{B_1}, q_{C_1}, ..., p_{B_m}, q_{C_m}$*:*
    $$c_z(q_{A_1}, ...q_{A_n}, p_{B_1}, q_{C_1}, ..., p_{B_m}, q_{C_m}) \to q_E$$
    And where for $i \in \{1..m\}$*, let* $A_{n+i} = (C_i \setminus B_i)$ *and*
    $$E = \{z\} \cup \bigcup_{i \in \{1..n+m\}} A_i$$

[3] These provisos, as well as the ones of the next transition, ensure linearity. In the paper of Levy and Villaret (2000), these provisos where omitted.

provided for all $i \in \{1..m\}$, all variables occurring in B_i also occur in C_i, and for all $j \neq k \in \{1..(n+m)\}$, $A_j \cap A_k = \emptyset$, and $z \notin A_j$.

Notice that the B_k's are treated as sets in the transitions but they denote lists: λxy is not the same λ-abstraction as λyx, so they have distinct associated constants but here they are treated as the same set.

Then, we introduce a set of tree-regular restrictions over the translated problem.

- *For any first-order variable X, the restriction $X \in \mathcal{L}(\mathcal{A}^{A,B})$.*
- *For any second-order variable F, the restriction $F(a, ..., a) \in \mathcal{L}(\mathcal{A}^{A,B})$, where $a \in \Sigma'$ is a first-order constant not associated to any bound variable, nor to a λ-binding.*

Example 6.10 Given the problem $f(X, X) \stackrel{?}{=}_{lsou} f(g(\lambda x.\ F(x)), F(g(\lambda y.\ y)))$, with unifier $\sigma = [F \mapsto \lambda x.\ x,\ X \mapsto g(\lambda x.\ x)]$, we can conjecture the following α-equivalent problem $f(X, X) \stackrel{?}{=}_{lsou} f(g(\lambda x.\ F(x)), F(g(\lambda x.\ x)))$ (this is the only solvable one), and translate it into the following context unification problem with tree-regular constraints:

$$
\begin{array}{l}
f(X, X) \stackrel{?}{=}_{cu} f(g'(b_{[x]}, F(c_x)), F(g'(b_{[x]}, c_x))) \\
\sigma(X) \in \mathcal{L}(\mathcal{A}^{\{[x]\},\{x\}}) \\
\sigma(F(a)) \in \mathcal{L}(\mathcal{A}^{\{[x]\},\{x\}})
\end{array}
$$

where the transitions of the tree automaton $\mathcal{A}^{\{[x]\},\{x\}}$ are the following:

$$
\begin{array}{lll}
a \to q_\emptyset & b_{[x]} \to p_{\{x\}} & c_x \to q_{\{x\}} \\
f(q_\emptyset, q_\emptyset) \to q_\emptyset & f(q_\emptyset, q_{\{x\}}) \to q_{\{x\}} & f(q_{\{x\}}, q_\emptyset) \to q_{\{x\}} \\
g'(p_{[x]}, q_{\{x\}}) \to q_\emptyset &&
\end{array}
$$

Notice that the transitions $f(q_{\{x\}}, q_{\{x\}}) \to q_{\{x\}}$ and $g'(p_{[x]}, q_\emptyset) \to q_\emptyset$ are not present in the automaton, otherwise the linearity on the instances of variables would not be guaranteed. Notice also, that the translation of substitution σ is also a solution of the translated problem of context unification with tree-regular constraints: $[F \mapsto \lambda x.\ x,\ X \mapsto g'(b_{[x]}, c_x)]$. ■

In the following lemmas we assume that all the bound variables of the problem are in B and all lists of the λ-abstractions bound variables are in A.

Lemma 6.11 *For any second-order substitution σ satisfying that $\sigma(X)$ does not contain variables of B in free positions, and satisfying also that the domain of σ does not contain variables of B either, let $\sigma' = trans^{A,B}(\sigma)$ be the context substitution defined by $\sigma'(X) = trans^{A,B}(\sigma(X))$. Then, for any term t we have*

$$trans^{A,B}(\sigma(t)) = \sigma'(trans^{A,B}(t))$$

Proof: By structural induction on t. For most cases, it is trivial. Let us see the two main cases.

If $t = h(t_1, ..., t_n, \lambda\vec{x_1}.u_1, ..., \lambda\vec{x_m}.u_m))$ where h is a higher-order constant, then assume variables of $\vec{x_i}$ are in B (otherwise it is trivial), and $\sigma(X)$ does not have variables from B in free positions. We have $\sigma(t) = h(\sigma(t_1), ..., \lambda\vec{x_1}.\sigma(u_1), ...)$ and

$$\begin{aligned} trans^{A,B}(\sigma(t)) &= trans^{A,B}(h(\sigma(t_1), ..., \lambda\vec{x_1}.\sigma(u_1), ...)) \\ &= h'(trans^{A,B}(\sigma(t_1)), ..., b_{\vec{x_1}}, trans^{A,B}(\sigma(u_1)), ...) \\ &= h'(\sigma'(trans^{A,B}(t_1)), ..., b_{\vec{x_1}}, \sigma'(trans^{A,B}(u_1)), ...) \\ &= \sigma'(trans^{A,B}(t)) \end{aligned}$$

If $t = F(t_1, ..., t_n)$ where $\sigma(F) = \lambda x_1...x_n.\ u$ then,

$$\begin{aligned} trans^{A,B}(\sigma(t)) &= trans^{A,B}((\lambda x_1...x_n.\ u)(\sigma(t_1), ..., \sigma(t_n))) \\ &= (trans^{A,B}(\lambda x_1...x_n.\ u))(trans^{A,B}(\sigma(t_1)), ..., trans^{A,B}(\sigma(t_n))) \\ &= (\lambda x_1...x_n.\ trans^{A,B\setminus\{x_1...x_n\}}(u))(trans^{A,B}(\sigma(t_1)), ..., trans^{A,B}(\sigma(t_n))) \\ &= \sigma'(F)(\sigma'(trans^{A,B}(t_1)), ..., \sigma'(trans^{A,B}(t_n)))) \\ &= \sigma'(F(trans^{A,B}(t_1), ..., trans^{A,B}(t_n))) \\ &= \sigma'(trans^{A,B}(t)) \end{aligned}$$

■

Lemma 6.12 *For any closed first-order term t,*

$$trans^{A,B}(t) \in \mathcal{L}(\mathcal{A}^{A,B})$$

if and only if:

1. *the set of bound variables of t is a subset of B*
2. *the set of lists of the bound variables of the λ-abstractions of t, is a subset of A*
3. *t is linear*

Proof:

($\Rightarrow$) The term $trans^{A,B}(t)$ only contains the variables that do not occur in B, and the λ-abstractions whose lists are not in A. Then, by the construction of $\mathcal{A}^{A,B}$, if $trans^{A,B}(t) \in \mathcal{L}(\mathcal{A}^{A,B})$, then $trans^{A,B}(t)$ does not contain any variable nor any λ-abstraction, therefore conditions 1 and 2 must hold. For condition 3, we only need to check that the provisos in the tree automaton transitions enforce linearity in $trans^{A,B}(t)$ for the original term t. This can be done by structural induction on t.

($\Leftarrow$) Notice that the function $trans^{A,B}$ translates any variable from t belonging to B into a constant c_x (except for bound variables bound by the most external bindings, but recall that t is first-order), and all λ-abstractions $\lambda\vec{y}$, $\vec{y} \in A$ into a constant $b_{\vec{y}}$. We also know that t is linear, therefore, $trans^{A,B} \notin \mathcal{L}(\mathcal{A}^{A,B})$ only if the tree automaton detects free occurrences of variables $x \in B$. The tree automaton has to reject terms containing constants c_x, except if t also contains a binding for x before or over it, i.e. if $trans^{A,B}(t)$ also contains $b_{[\vec{y}]}$ as its immediate left brother or as an immediate left brother of one of its predecessors, for some list such that $x \in [\vec{y}]$. This is exactly what the automaton does. We only have to prove, by structural induction on u, that if $u \rightarrow^* q_X$, then X is the subset of variables of B occurring in u and such that, for any $x \in X$, there is no constant $b_{[\vec{y}]}$ occurring in u with $x \in [\vec{y}]$ in the positions described before. But this will not happen because t is a linear, closed, first-order term. ■

Theorem 6.13 *A linear second-order unification problem $s \stackrel{?}{=}_{lsou} t$ is unifiable if and only if there exists an α-equivalent unification problem $s' \stackrel{?}{=}_{lsou} t'$ such that*

$$trans^{A,B}(s') \stackrel{?}{=}_{cu} trans^{A,B}(t')$$

and the corresponding tree-regular constraints are solvable. Here, B is the set of bound variables of s' and t', and A is the set of lists of bound variables corresponding to λ-abstractions of s' and t'.

Proof:

($\Rightarrow$) Let σ be a most general unifier of $s \stackrel{?}{=}_{lsou} t$, it's easy to prove that most general unifiers do not introduce constants, bound variables or λ-abstractions not occurring in $s \stackrel{?}{=}_{lsou} t$ (see for instance the procedure of Levy (1996)). Being s (and t) and terms in $Range(\sigma)$ in normal form, $\sigma(t)$ only requires β-recutions (not η-expansions) to be normalised, hence, no new λ-abstractions or bound variables not occurring in $s \stackrel{?}{=} t$ will be introduced[4]. This is true also for extensions of σ that are closed without using λ-abstractions or bound variables (like it is done in Theorem 4.15), let σ' be such unifier. Then, we can conclude that there exists an α-conversion that makes s and t α-equivalents to s' and t' such that $\sigma'(s')$ is syntactically equal to $\sigma'(t')$. The α-conversion needed to transform s into s' and t into t' is the one conjectured in step 1 of the translation algorithm.

Let us see now that $\sigma_c = trans^{A,B}(\sigma')$ solves $trans^{A,B}(s') \stackrel{?}{=}_{cu} trans^{A,B}(t')$. We have $\sigma'(s') = \sigma'(t')$ and, by Lemma 6.11 (we can apply this Lemma because obviously σ' cannot contain variables of B in free positions), $trans^{A,B}(\sigma'(s')) = \sigma_c(trans^{A,B}(s'))$, therefore $\sigma_c(trans^{A,B}(s')) = \sigma_c(trans^{A,B}(t'))$.

Now we have to see that $\sigma_c(X)$ and $\sigma_c(F(a,\ldots,a))$ satisfy the tree-regular restriction, for any first-order variable X and second-order one $F \in Dom(\sigma_c)$. As $\sigma'(X)$ does not contain any bound variable of the problem, i.e. any variable from B, in free positions, and it is linear (otherwise σ' would not be a valid

[4] recall that η-expansions introduce λ-abstactions and bound variables

unifier), by Lemma 6.12 $trans^{A,B}(\sigma'(X)) = \sigma_c(X)$ satisfies the tree restriction. Similarly for $F(a,\ldots,a)$ when F is a context variable.

($\Leftarrow$) This implication is based on two facts. First, the $trans^{A,B}$ is injective. This allows us to use the inverse of $trans^{A,B}$ to compute the solution of the linear second-order unification problem. Second, Lemma 6.12 and the tree restrictions ensure that we do not capture any free variable, and that substitutions are linear terms. Thus, the inverse translation of the solution for the translated (context) problem is a correct linear second-order unifier of the original (linear second-order) problem. ■

Corollary 6.14 *Linear Second-Order Unification is reducible to Context Unification with tree-regular constraints.*

Proof: The conjecture of an α-conversion is computable because, although we have infinitely many possible names for bound variables, the only relevant choice is whether we give the same name to two bound variables or two λ-bindings (and in a problem there are finitely many). ■

6.4 Translating Tree-Regular Constraints to Regular Constraints over Traversal Sequences

We have seen in the previous chapter that, if the rank-bound conjecture is true, then Context Unification is decidable. The proof is based on a reduction of Context Unification to Word Unification with regular constraints. In this section we will show that a certain kind of tree-regular languages can be reduced to regular languages on the traversals of trees, mainly the rank-bound languages. This result suggests the possibility that, if the rank-bound conjecture is true, then the decidability proof of Context Unification could be extended to Context Unification with tree-regular constraints and therefore, Linear Second-Order Unification would also be decidable. Nevertheless, as we will discuss in the next section, the adaptation of the proof seems to require a stronger conjecture to hold.

In what follows we show how membership equations on tree-regular languages of rank-bound terms can be reduced to membership equations on (word) regular languages. We start by defining rank-bound tree automata.

Definition 6.15 *For any tree automaton $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$, and any state $q_i \in \mathcal{Q}$, we define*[5]

$$\mathrm{rank}(q_i) = \max\{\mathrm{rank}(t) \mid t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{q_i\}, \Delta\rangle)\}$$

For any tree automaton $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$, we define

$$\mathrm{rank}(\mathcal{A}) = \max\{\mathrm{rank}(q_i) \mid q_i \in \mathcal{Q}_f\}$$

[5]Notice that $\langle \Sigma, \mathcal{Q}, \{q_i\}, \Delta\rangle$ is similar to $\mathcal{A}$ but with a unique final state q_i.

A tree automaton $\mathcal{A}$ is said to be rank-bound if $\mathrm{rank}(\mathcal{A}) < \infty$.

Notice that the rank of the states of a tree automaton satisfies the following property:

- for any state q having only transitions like $c \to q$, where $c \in \Sigma$ is a 0-ary constant, $\mathrm{rank}(q) = 0$, and
- for any accessible state q_0 having transitions like $f(q_1, q_2) \to q_0$, where $f \in \Sigma$ is a binary function symbol, we have:

$$rank(q_0) \geq \begin{cases} max\{rank(q_1), rank(q_2)\} & \text{if } rank(q_1) \neq rank(q_2) \\ rank(q_1) + 1 & \text{if } rank(q_1) = rank(q_2) \end{cases}$$

Next we will translate tree-regular restrictions to (word) regular restrictions over traversal sequences of terms. We can assume that the signature Σ contains just constants and a binary symbol (say f). The result can easily be extended to any signature. The signature used in the (word) regular automata is the *extended signature* of Σ (see Definition 5.4 and Definition 5.5).

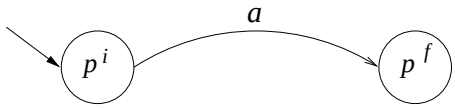
Definition 6.16 *For any rank-bound tree automaton $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ and any state $q \in \mathcal{Q}$, we define a regular language R_q satisfying that*

- *for any term $t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{q\}, \Delta \rangle)$, $R_q \cap \mathrm{trav}(t) \neq \emptyset$,*
- *and*

$$R_q \subseteq \bigcup_{t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{q\}, \Delta \rangle)} \mathrm{trav}(t)$$

We will construct the automaton that recognises R_q using the following rules. Assume that $R_{q'}$ is already computed for any state $q' \in \mathcal{Q}$ with $\mathrm{rank}(q') < \mathrm{rank}(q)$. Let $n = \mathrm{rank}(q)$. The automaton R_q has a pair of states p^i and p^f for any state p of the tree automaton satisfying $\mathrm{rank}(p) = n$, and some additional states that will be specified later. The initial state of R_q is q^i, and there is a single final state which is q^f. The set of transitions of R_q is defined as follows.

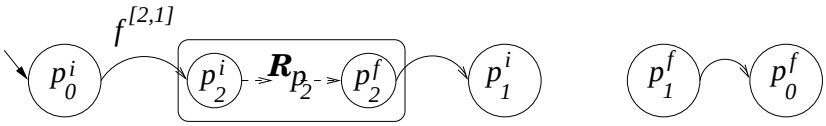
- Base case, *for any state $p \in \mathcal{Q}$ satisfying $\mathrm{rank}(p) = n$, and any transition $a \to p \in \Delta$, where a is 0-ary, we add a transition:*



  from p^i to p^f labeled with a.

- Inductive case 1, *for any state p_0 with $\mathrm{rank}(p_0) = n$ and any transition $f(p_1, p_2) \to p_0$ satisfying $\mathrm{rank}(p_2) < \mathrm{rank}(p_1) \leq \mathrm{rank}(p_0)$* [6]

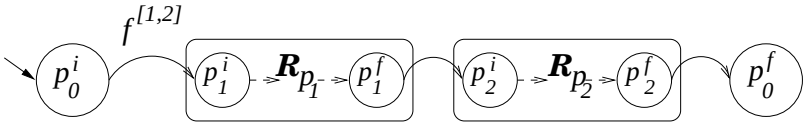
[6] Notice that if $\mathrm{rank}(p_1) = \mathrm{rank}(p_2) = \mathrm{rank}(p_0)$ then $\mathrm{rank}(p_0) = \infty$ and the tree automaton would be non-rank-bound. Thus the existence of a bound n for the rank of the tree automaton is crucial in our translation.



we can assume that R_{p_2} is already computed. We add a copy of the automaton R_{p_2}, i.e. a copy of all its states and transitions (these are the unspecified additional states). We also add a transition from p_0^i to the initial state of the copy of R_{p_2} labeled with $f^{[2,1]}$, an ϵ-transition from the final state of R_{p_2} to p_1^i, and another from p_1^f to p_0^f.

For any transition $f(p_1, p_2) \rightarrow p_0$ satisfying $\text{rank}(p_1) < \text{rank}(p_2) \leq \text{rank}(p_0)$ *we do something similar using the label $f^{[1,2]}$*

- Inductive case 2, *for any state p_0 with* $\text{rank}(p_0) = n$ *and any transition $f(p_1, p_2) \rightarrow p_0$ satisfying* $\text{rank}(p_1) < \text{rank}(p_0)$ *and* $\text{rank}(p_2) < \text{rank}(p_0)$



we can assume that R_{p_1} and R_{p_2} have been already computed. We add a copy of each one of these automata, a transition from p_0^i to the initial state of R_{p_1} labeled with $f^{[1,2]}$, an ϵ-transition from the final state of R_{p_1} to the initial state of R_{p_2}, and another from the final state of R_{p_2} to p_0^f.

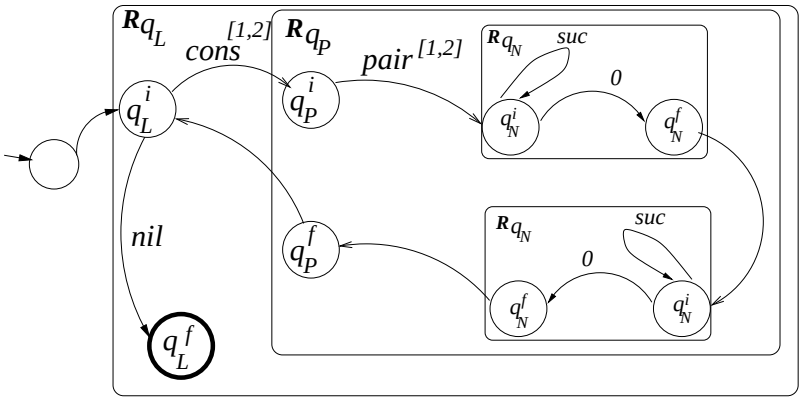
The final automaton associated to $\mathcal{A}$ has the extended signature of Σ and consists of an initial state q_0, a copy of R_q for any final state $q \in \mathcal{Q}_f$, a ϵ-transitions from q_0 to each one of the initial states of the R_q's. The set of final states of $\mathcal{A}$ is the set of final states of the R_q's such that $q \in Q_f$.

Notice that with these cases, transitions like $f(q, q) \rightarrow q$ are not considered, because this means that $\text{rank}(q) = \infty$, and q can not lead to a final state.

Example 6.17 The tree automaton defined by the following transitions:

$0 \rightarrow q_N$, $\quad pair(q_N, q_N) \rightarrow q_P$, $\quad nil \rightarrow q_L$,
$s(q_N) \rightarrow q_N$, $\quad cons(q_P, q_L) \rightarrow q_L$

is translated into the following regular automaton:



The term $cons(pair(suc(suc(0)), suc(0)), cons(pair(suc(0), 0), nil))$ recognised by the tree automaton has a traversal sequence:

$$cons^{[1,2]}\ pair^{[1,2]}\ suc\ suc\ 0\ suc\ 0\ cons^{[1,2]}\ pair^{[1,2]}\ suc\ 0\ 0\ nil$$

recognised by the regular automaton. ■

Theorem 6.18 *For any tree-regular language $\mathcal{L}(\mathcal{A})$ of a rank-bound tree automaton $\mathcal{A}$, let $\mathcal{L}(B)$ be the (word) regular language recognised by the (word) automaton B resulting from applying the previous translation. The following properties hold:*

1. *If $t \in \mathcal{L}(\mathcal{A})$ then there exists a sequence $l \in \mathcal{L}(B)$ such that $l \in \text{trav}(t)$.*
2. *If $l \in \mathcal{L}(B)$ then there exists a term $t \in \mathcal{L}(\mathcal{A})$ such that $l \in \text{trav}(t)$.*

Proof:

1) We first prove that for any term t, and any $q \in \mathcal{Q}$, if $t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{q\}, \Delta \rangle)$, then there exists a sequence $l \in \text{trav}(t)$ such that $l \in R_q$. This is proved by structural induction on t.

- The term $t = c$ is a constant. Then, if $t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{q\}, \Delta \rangle)$, then it has been accepted due to a transition $c \to q$ in the tree automaton. The automaton R_q recognises the sequence c using a transition from q^i to q^f.
- The term is $t = f(t_1, t_2)$ and is recognised using a transition $f(p_1, p_2) \to q$ of the tree automaton. By the induction hypothesis we can ensure that the automaton R_{p_1} and R_{p_2} recognise a traversal sequence l_1 of t_1 and l_2 of t_2 respectively. The translation that we use for the transition $f(p_1, p_2) \to q$, in the two possible cases, recognises a traversal of t. This is $f^{[1,2]} l_1 l_2$ or $f^{[2,1]} l_2 l_1$, depending on the relation between the rank of p_1 and of p_2.

By the construction of B, if $t \in \mathcal{L}(\mathcal{A})$ then $t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{q\}, \Delta \rangle)$ for some $q \in \mathcal{Q}_f$, and R_q recognises a traversal of t, therefore so does B.

2) Firstly, we prove the following property of the automaton R_p.

For any states $p_1, p_2 \in \mathcal{Q}$, if $\text{rank}(p_1) = \text{rank}(p_2) = \text{rank}(p)$, then any word w read when going from p_1^i to p_2^f in the automaton R_p satisfies $w \in \text{trav}(t)$ for some term $t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p_1\}, \Delta \rangle)$.

To prove this result, we proceed by induction on the length of w and the rank of p.

1. If $w = a\ w'$ where a is a constant, then after reading a we are in p_1^f, and from p_1^f to p_2^f we can only follow ϵ-transitions (notice that, by the definition of the translating rules, in an automaton R_p, all transitions between final states p_j^f and p_k^f such that $\text{rank}(p_j) = \text{rank}(p_k) = \text{rank}(p)$ are ϵ-transitions), so w' is the empty word. Along the whole path we have only read a, where $a \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p_1\}, \Delta \rangle)$.

2. If $w = f\ w'$ where f is unary, then after reading f we are in some p_3^i. There are two cases:

   (a) If $\text{rank}(p_3) < \text{rank}(p_1)$ then we are in the initial state of a copy of the subautomaton R_{p_3}. By the induction hypothesis, the automaton R_{p_3} only recognises traversal sequences of terms $t_3 \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p_3\}, \Delta \rangle)$. After reading this traversal sequence and an ϵ-transition, we are in p_1^f. From there to p_2^f we can only follow ϵ-transitions. Along the whole path we have recognised a traversal sequence of $f(t_3)$.

   (b) If $\text{rank}(p_3) = \text{rank}(p_1)$ then applying the induction hypothesis, from p_3^i to p_2^f we read a traversal sequence of some $t_3 \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p_3\}, \Delta \rangle)$. Along the whole path we have also recognised a traversal sequence of $f(t_3) \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p_1\}, \Delta \rangle)$.

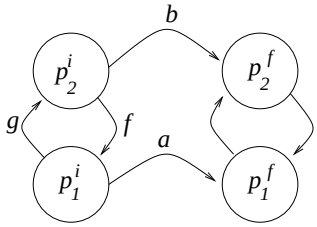
3. If $w = f\ w'$ where f is binary, then after reading f we are in the initial state of a copy of a subautomaton R_{p_3} where $\text{rank}(p_3) < \text{rank}(p_1)$. After reading a traversal sequence of some term $t_3 \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p_3\}, \Delta \rangle)$ and an ϵ-transition, we leave the subautomaton and go to some state p_4^i. Now we have to consider the same subcases $\text{rank}(p_4) < \text{rank}(p_1)$ and $\text{rank}(p_4) = \text{rank}(p_1)$, and proceed as when f is unary.

For $p_1 = p_2 = p$, the previous property states that any word recognised by R_p corresponds to some traversal sequence of some term $t \in \mathcal{L}(\langle \Sigma, \mathcal{Q}, \{p\}, \Delta \rangle)$. From here, we can conclude that any word recognised by B corresponds to some traversal sequence of some term recognised by A.

Notice that when we have crossed definitions between states, the induction hypothesis on the rank of p is not enough and we need to use an induction hypothesis on the length of the sequence. Here there is a typical example of crossed definition:

$$\begin{array}{ll} f(p_1) \to p_2, & b \to p_2, \\ g(p_2) \to p_1, & a \to p_1 \end{array}$$

We get $\text{rank}(p_1) = \text{rank}(p_2)$ and the following automaton:



In such situations, to prove that any path from p_1^i to p_1^f recognises a traversal of some term t_1, we have to use as induction hypothesis that any *shorter* path from p_2^i to p_2^f recognises a traversal of some term t_2.

■

The set of terms satisfying $\text{rank}(t) \leq n$ defines a tree-regular language. Moreover, this language can be recognised by a rank-bound tree automaton $\mathcal{A}_n$. For instance, if $\Sigma = \{a, b, h(), f(,)\}$, then

$$\mathcal{A}_n = \begin{cases} \Sigma = \{a, b, h(), f(,)\},\ \mathcal{Q} = \{q_0, ..., q_n, q_{n+1}\},\ \mathcal{Q}_f = \{q_0, ..., q_n\}, \\ \Delta = \begin{cases} a \to q_0, \quad b \to q_0, \\ h(q_i) \to q_i & \text{for any } i \in [0..n+1] \\ f(q_i, q_j) \to q_{max\{i,j\}} & \text{for any } i, j \in [0..n+1] \text{ with } i \neq j \\ f(q_i, q_i) \to q_{i+1} & \text{for any } i \in [0..n] \end{cases} \end{cases}$$

Rank-boundness can also be defined for languages, obtaining a good correspondence with automata.

Definition 6.19 *A tree-regular language L is said to be n* rank-bound *if for any term $t \in L$ we have* $\text{rank}(t) \leq n$. *A tree-regular language L is said to be* rank-bound *if there exists a natural number n such that L is n rank-bound.*

Theorem 6.20 *Any rank-bound tree-regular language is recognised by a rank-bound tree automaton. The language recognised by a rank-bound tree automaton is a rank-bound tree-regular language.*

Definition 6.21 *A* rank-bound tree-regular constraint *is a tree-regular constraint that recognises rank-bound languages.*

Now, from Theorem 5.27 and Theorem 6.18 the following result holds:

Corollary 6.22 *Context Unification with rank-bound tree-regular constraints over all variables occurring in the problem is decidable.*

6.5 About Decidability

Although we have proved that Linear Second-Order Unification can be reduced to Context Unification with Tree-Regular Constraints, and that rank-bound regular constraints can be reduced to regular contraints on the traversals, it is not

clear what happens with decidability of Linear Second-Order Unification in case Context Unification turns out to be decidable.

What is true is that if Context Unification with tree-regular constraints were decidable, then Linear Second-Order Unification would also be. But Corollary 6.22 only proves decidability of Context Unification with *rank-bound* tree-regular constraints.

In other words, if the conjecture turns out to be true, we just get decidability of Context Unification, but this would not say anything about decidability of Context Unification with tree-regular constraints. One could think that, given a context unification problem S with tree-regular constraints R, one could build a rank-bound tree regular constraint Rb, as the result of intersecting R and the tree-regular constraints that state the bound on the rank of the solutions of S. Unfortunately, there is no guarantee that any solution of $S + R$ is rank-bound. Therefore, satisfiability would not be preserved in $S + Rb$.

To "use" the rank, we would need a *stronger* conjecture like:

Conjecture 6.23 *[Extended Rank-Bound Conjecture] There exists a computable function Φ' such that, for any solvable context unification problem $t \stackrel{?}{=} u$ with tree-regular constraints R, there exists a ground unifier σ satisfying*

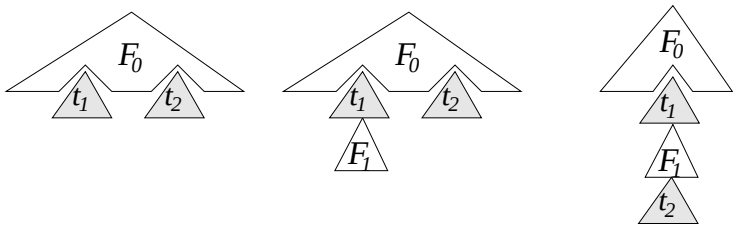
$$\text{rank}(\sigma(t)) \leq \Phi'(|t \stackrel{?}{=} u|, |R|)$$

We leave the study of this Conjecture and its implications for further work.

6.6 Extending the Results to Higher-Order Unification

In Section 6.3 we have shown how Linear Second-Order Unification can be reduced to Context Unification with tree-regular constraints. In this section we discuss whether this result could be extended to Linear Higher-Order Unification.

Higher-Order Unification can be defined as the problem of finding a substitution σ making the normal form of two instances of terms $\sigma(s)$ and $\sigma(t)$ equal. When we try to find such a substitution we have to take into account how these terms will be β-reduced after being instantiated. The problem is simple in linear second-order. We know that any instance of $F(t_1, ..., t_n)$, after β-reduction, will contain $\sigma(t_i)$ as subterms, and representing $\sigma(F(t_1, ..., t_n))$ as a tree, all nodes corresponding to $\sigma(F)$ will be connected, forming a context. In third-order the situation is more complicate. If we apply the substitution $F \mapsto \lambda yz.\ g_1(y(g_2(z)))$ to $F(\lambda x.\ f(x), a)$ we get $g_1(f(g_2(a)))$. The nodes corresponding to F are no longer connected: $\sigma(F)$ is broken into pieces, and some of the arguments can also disappear. Each one of such pieces forms a kind of context. For instance, if $F : (\iota \to \iota) \to \iota \to \iota$, any instance of $F(t_1, t_2)$ has one of the following forms:



Each one of these situations is respectively captured by:

$$F \mapsto \lambda xy.\ F_0(\lambda z.\ x(z), y)$$
$$F \mapsto \lambda xy.\ F_0(\lambda \vec{z}.\ x(F_1(\vec{z})), y)$$
$$F \mapsto \lambda xy.\ F_0(\lambda \vec{z}.\ x(F_1(y, \vec{z})))$$

In this example, F_0 is still a third-order typed variable. Moreover, the first instantiation is $F \mapsto F_0$ in normal form, so it subsumes the other two. The second one is equal to the first one, if $\vec{z}$ contains a single variable and we instantiate $F_1 \mapsto \lambda z.\ z$. In fact, this classification only makes sense if we translate F_0 into a context variable using the method described in Section 6.3 for higher-order constants. We would get:

$$F \mapsto \lambda xy.\ F'_0(d_z, x(c_z), y)$$
$$F \mapsto \lambda xy.\ F'_0(X_{\vec{z}}, x(F_1(\vec{c_z})), y)$$
$$F \mapsto \lambda xy.\ F'_0(X_{\vec{z}}, x(F_1(y, \vec{c_z})))$$

The variable $X_{\vec{z}}$ encodes the binding $\lambda\vec{z}$. If we were able to know a priori how long and with which types, can these λ-bindings be, then the translation would not seem much more complicated than in the second-order case.

We believe that the results on Linear Higher-Order Matching of de Groote (2000) and of Dougherty and Wierzbicki (2002) shed some light on this question. But we leave this study for further work.

6.7 Summary

As we have already explained in Chapter 3, Linear Second-Order and Context Unification are closely related problems and they have sometimes been identified. In this chapter we have shown that such an identification is not possible in a naif sense. We have shown that the permission of using λ-bindings and bound variables in Linear Second-Order Unification seems to really increase the expressive power of this problem with respect to Context Unification.

Nevertheless, we have still been able to relate both problems by means of reducing Linear Second-Order Unification to Context Unification with tree-regular constraints. This reduction relies on the trick of translating λ-bindings and bound variables to new constant symbols that codify the names of the bound variables. The tree-regular constraints serve to guarantee that the reduction is sound by means of avoiding variable capture and loss of linearity in the instantiation of context variables.

Then, trying to be able to apply the results of Chapter 5 to Context Unification with tree-regular constraints, we have shown that satisfiability of rank-bound tree-regular constraints can be reduced to satisfiability of regular constraints over traversal sequences. We have argued why this reduction is not enough and we have extended the rank-bound conjecture to deal also with tree-regular constaints.

Finally, we have explained the difficulties that arise when trying to extend the reduction of Linear Second-Order Unification to Context Unification with tree-regular constraints, to Linear Higher-Order Unification.

Chapter 7

Describing Lambda-Terms in Context Unification with Tree-Regular Constraints

In this chapter we relate the Constraint Language for Lambda-Structures with the Problem of Context Unification with tree-regular constraints. On the one hand, the Constraint Language for Lambda-Structures has been shown to be more suitable for applications in computational linguistics than Context Unification but, on the other hand, the unification framework is more computationally well-known and understood. Our result helps to better understand the computational nature of the Constraint Language for Lambda-Structures, and establishes a more precise "bridge" between both frameworks that can be used to apply the theoretic results from one side to the other.

This chapter summarises the results of (Niehren and Villaret, 2002, 2003).

7.1 Introduction

In the Introduction Chapter we comment on the relevance of Higher-Order Unification in the field of computational linguistics (Dalrymple *et al.*, 1991; Gardent and Kohlhase, 1996; Miller and Nadathur, 1986), and more precisely of Linear Second-Order (Pinkal, 1995) and Context Unification (Niehren *et al.*, 1997b) in modelling semantics of ambiguous sentences applications. There exists another formalism for this last purpose, the so-called *Constraint Language for Lambda Structures* (Egg *et al.*, 2001), which is a first-order language that is more suitable for representing semantic underspecification than Unification (see Section 7.3). It has been proved that some fragments of this language are closely related to Context Unification (Egg *et al.*, 1998, 2001; Erk *et al.*, 2002). On the one hand it is known that the *Dominance Constraints* sublanguage (Koller *et al.*, 1998), which on its own deals with the *dominance* and *labeling* relation between nodes

of trees, is subsumed by the stratified fragment of Context Unification (Niehren, 2002), and it is also decidable (see Subsection 3.3.2). On the other hand, it has been proved that the *Parallelism Constraints* sublanguage (Erk and Niehren, 2000), that in turn subsumes Dominance Constraints and deals with the *parallelism* relation between segments of trees, has the same expressive power as Context Unification (Niehren and Koller, 2001), hence its decidability is still unknown. Recently, a fragment of Parallelism Constraints has been introduced and shown to be decidable by Erk and Niehren (2003), the so called *well-nested* fragment. In spite of these equivalences, the algorithms used to solve these constraints have a nicer behaviour than the ones used to solve Context Unification (Koller, 1998; Erk and Niehren, 2000; Althaus *et al.*, 2003; Erk *et al.*, 2002).

The Constraint Language for Lambda Structures also provides *lambda binding* and *anaphoric binding* constraints, and has been extended with *beta reduction* and *group parallelism* constraints. These constraints are used in applications but their expressiveness has never been studied and the way they are related with the Unification framework is still unknown.

In this chapter we partially ask these questions by relating Parallelism with lambda binding constraints, with Context Unification. The key point of our result is the *non-intervenance* property which is crucial to ensure soundness of the solutions when eliminating lambda binding constraints and then translating the problem to Context Unification. This property can be expressed by means of a logic that we introduce and that is equal in expressiveness to tree automata: the Monadic Second-Order Logic over Dominance Constraints (in fact, as we will show, the first-order fragment is enough to make the translation). Then, using several encoding tricks, we show that Parallelism with lambda binding constraints can be expressed in Context Unification with tree-regular constraints.

One may think that if Parallelism Constraints are equivalent to Context Unification and we just add lambda binding constraints, we could directly translate Parallelism with lambda binding constraints to Linear Second-Order Unification, but as we will show in Section 7.3, the notion of binding required for the constraint language is not the same as the notion of binding in λ-calculus and hence it is not the same as in Linear Second-Order Unification.

In Section 7.2 we define the sublanguages of the Constraint Languages for Lambda Structures that we consider in this Chapter. Then, in Section 7.3, we show why the direct translation of Parallelism with lambda binding constraints to Linear Second-Order Unification does not work. In Section 7.4 we introduce the non-intervenance property. In Section 7.5 we prove that lambda binding constraints can be eliminated using First-Order Dominance Formulae, hence we can translate Parallelism with lambda binding constraints into Parallelism with First-Order Dominance formulae. Section 7.6 is devoted to the introduction, for the first time, of the Monadic Second-Order Dominance Logic and showing its satisfiability equivalence with respect to tree-regular constraints. In Section 7.7 we extend this last equivalence result when considering Parallelism Constraints. Finally, in Section 7.8 we complete the reduction to Context Unification with tree-regular constraints. In Section 7.9 we comment on the limitations of the

translation.

7.2 The Parallelism and Lambda Binding Constraints Language

We now describe the semantics and the relations considered by the languages that we are introducing; firstly, *tree structures* with the parallelism relation and then an extension of them, the *lambda structures* with the parallelism relation also. Then we introduce the syntax of the constraint languages that we consider.

We make the usual assumption of a given finite first-order *signature* Σ of function symbols ranged over by $f, g, h, \ldots$ for functions and $a, b, c, \ldots$ for constants where each function symbol has an arity $arity(f) \geq 0$. We also assume that there exists at least one constant $a \in \Sigma$ of arity 0 and at least one binary function symbol (say $f \in \Sigma$).

7.2.1 Tree Structures and Parallelism

The relation between terms and trees and its representation has been taken for granted during all this work. We now make this relation precise in the approach of the Constraint Language for Lambda Structures.

Definition 7.1 *A* finite tree τ *(*tree *for short) over the signature* Σ *is a ground term over* Σ*:*

$$\tau ::= f(\tau_1, \ldots, \tau_n)$$

where $f \in \Sigma$, $n = arity(f) \geq 0$ *and* $\tau_1, \ldots, \tau_n$ *are trees. We identify a* node *of a tree with the word of positive integers* π *that addresses this node from the root, as it is done with subterms and positions in terms (see Definition 5.1):*

$$\mathsf{nodes}_{f(\tau_1,\ldots,\tau_n)} = \{\epsilon\} \cup \{i\pi \mid 1 \leq i \leq n,\ \pi \in \mathsf{nodes}_{\tau_i}\}$$

The empty word ϵ *is called the* root *of the tree, i.e.* $\mathsf{root}(\tau) = \epsilon$*, while a word* $i\pi$ *addresses the node* π *of the* i*-th subtree of* τ*.*
We freely identify a tree τ *with the function* $\tau : \mathsf{nodes}_\tau \to \Sigma$ *that maps nodes to their label, then, for a tree* $\tau = f(\tau_1, \ldots, \tau_n)$ *we have:*

$$\tau(\pi) = f(\tau_1, \ldots, \tau_n)(\pi) = \begin{cases} f & \text{if } \pi = \epsilon \\ \tau_i(\pi') & \text{if } \pi = i\pi' \text{ and } i \in \{1..n\} \end{cases}$$

If τ *is a tree with* $\pi \in \mathsf{nodes}_\tau$ *then we write* $\tau.\pi$ *for the subtree of* τ *rooted by* π*, and* $\tau[\pi/\tau']$ *for the tree obtained by replacing the subtree of* τ *at position* π *by* τ'*.*

Definition 7.2 *The* tree structure τ *of a tree* τ *(we will often make no distinction between trees and tree structures) over* Σ *is a first-order structure with domain* nodes_τ*. It provides a children-labeling relation* $: f$ *for each* $f \in \Sigma$*:*

$$: f = \{(\pi, \pi 1, \ldots, \pi n) | \tau(\pi) = f \ \wedge \ arity(f) = n\}$$

We denote this relation as $\pi{:}f(\pi 1, \ldots, \pi n)$ *where the children of* π *are* $\pi 1, \ldots, \pi n$ *in that order from left to right. The* dominance relation $\pi \triangleleft^* \pi'$ *holds for* τ *if* π *is an ancestor of* π'*, in other words, if* π *is above* π' *in* τ*, or if* π *is a prefix of* π'*.* Strict dominance $\pi \triangleleft^+ \pi'$ *holds for* τ *if* $\pi \triangleleft^* \pi'$ *but* $\pi{=}\pi'$ *does not. The* disjointness relation $\pi \perp \pi'$ *holds for* τ *if neither* $\pi \triangleleft^* \pi'$ *nor* $\pi' \triangleleft^* \pi$ *in* τ*.*

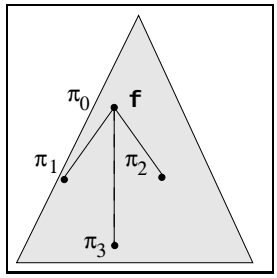


Figure 7.1: Representation of part of a tree that satisfies the relations of children-labeling: $\pi_0 : f(\pi_1, \pi_2)$, dominance: $\pi_0 \triangleleft^* \pi_3$ and disjointness: $\pi_1 \perp \pi_2$.

Definition 7.3 *A* segment $\sigma = \pi/\pi_1, \ldots, \pi_n$ *for* $n \geq 0$*, of a tree* τ *(see Figure 7.2), is a tuple of nodes* $\pi, \pi_1, \ldots, \pi_n$ *of* τ *such that* π *dominates all* π_i *and all* π_i *with different indexes are pairwise disjoint. We call* π *the* root *of* σ *and* $\pi_1, \ldots, \pi_n$ *its* holes*. The* nodes of a segment σ *of a tree* τ*, noted as* $\mathsf{nodes}_\tau(\sigma)$*, lie between the root and the holes of* σ *in* τ*:*

$$\mathsf{nodes}_\tau(\pi/\pi_1, \ldots, \pi_n) = \{\pi' \in \mathsf{nodes}_\tau \mid \pi \triangleleft^* \pi' \text{ and not } \pi_i \triangleleft^+ \pi' \text{ for } i \in \{1..n\}\}$$

The labels of holes "do not belong" to segments. The inner nodes of a segment *are those that are not holes:*

$$\mathsf{nodes}^-_\tau(\sigma) = \mathsf{nodes}_\tau(\sigma) - \{\pi_1, \ldots, \pi_n\} \qquad \text{if } \sigma = \pi/\pi_1, \ldots, \pi_n$$

The segment $\pi/$ *is the segment with 0 holes (hence, a tree).*

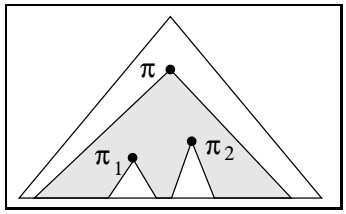


Figure 7.2: The segment $\pi/\pi_1, \pi_2$.

Segment nodes generalise tree nodes in that $\mathsf{nodes}_{\tau.\pi} = \mathsf{nodes}_\tau(\pi/)$ for all trees τ and $\pi \in \mathsf{nodes}_\tau$.

Definition 7.4 *A* correspondence function *between segments σ_1 and σ_2 with the same number of holes, of a tree τ is a function $c : \mathsf{nodes}_\tau(\sigma_1) \to \mathsf{nodes}_\tau(\sigma_2)$ that is one-to-one and onto and satisfies the following homomorphism conditions:*

1. *The root of σ_1 is mapped to the root of σ_2 and the sequence of holes of σ_1 is mapped to the sequence of holes of σ_2 in the same order.*
2. *The labels of inner nodes $\pi \in \mathsf{nodes}^-_\tau(\sigma_1)$ are preserved: $\tau(\pi) = \tau(c(\pi))$.*
3. *The children of inner nodes $\pi \in \mathsf{nodes}^-_\tau(\sigma_1)$ are mapped to corresponding children in σ_2: for all $1 \leq i \leq arity(\tau(\pi))$ it holds that $c(\pi i) = c(\pi)i$.*

Definition 7.5 *We call two segments σ_1 and σ_2 of a tree structure τ* (tree) parallel *and write $\sigma_1 \sim \sigma_2$ if and only if there exists a correspondence function between them.*

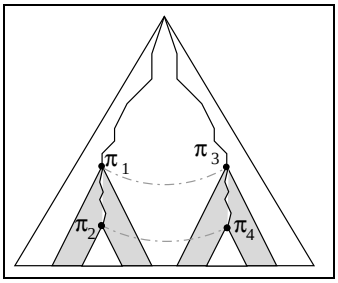


Figure 7.3: Parallelism relation between π_1/π_2 and π_3/π_4.

Tree parallelism can be characterised equivalently by saying that parallel segments are occurrences of the same context (from the first-order perspective but with n holes).

7.2.2 Lambda Structures and Parallel Lambda Binding

Now we are going to introduce lambda structures which are tree structure extensions corresponding to λ-terms and including *anaphora* bindings. Here we just define and use the λ-binding feature, which is the one we are considering. One can find the precise definition of lambda structures in (Egg *et al.*, 2001).

Lambda structures represent λ-terms uniquely modulo renaming of bound variables. They are tree structures extended by a lambda binding function, which can be represented graphically as lambda binding edges that go from nodes corresponding to bound variables to nodes corresponding to λ-abstractions. The signature Σ of lambda structures contains, at least, the symbols var (arity 0, for bound variables), lam (arity 1, for λ-abstractions), and @ (arity 2, for application).

The tree uses these symbols to reflect the structure of the λ-term in a curried manner. There is a binding function λ which maps var-labeled to lam-labeled

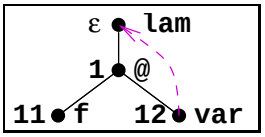


Figure 7.4: The lambda structure of $\lambda x.\,(f\ x)$.

nodes (var-nodes and lam-nodes, for short). For example, Figure 7.4 shows the lambda structure of the term $\lambda x.\,(f\ x)$ which satisfies $\lambda(12) = \epsilon$.

Definition 7.6 A *lambda structure* (τ, λ) is a pair of a tree τ and a total binding function $\lambda : \tau^{-1}(\mathsf{var}) \to \tau^{-1}(\mathsf{lam})$ such that $\lambda(\pi) \lhd^* \pi$ for all var-nodes π in τ.

We consider lambda structures as logical structures with the relations of tree structures plus lambda binding $\lambda(\pi) = \pi'$, and its inverse relation. Inverse lambda binding $\lambda^{-1}(\pi_0) = \{\pi_1, \ldots, \pi_n\}$ states that π_0 binds $\pi_1, \ldots, \pi_n$ and no other nodes.

Definition 7.7 *Two segments σ_1, σ_2 of a lambda structure (τ, λ) are* (binding) parallel $\sigma_1 \sim \sigma_2$ *if they are tree parallel so that the correspondence function c between σ_1 and σ_2 satisfies the following axioms of parallel binding (see Figure 7.5):*

Internal binder. *Internal lambda binders in parallel segments correspond: for all $\pi \in \mathsf{nodes}^-_\tau(\sigma_1)$ if $\lambda(\pi) \in \mathsf{nodes}^-_\tau(\sigma_1)$ then $\lambda(c(\pi)) = c(\lambda(\pi))$.*

External binder. *External lambda binders of corresponding* var*-nodes are equal: for all $\pi \in \mathsf{nodes}^-_\tau(\sigma_1)$ if $\lambda(\pi) \notin \mathsf{nodes}^-_\tau(\sigma_1)$ then $\lambda(c(\pi)) = \lambda(\pi)$.*

No hanging binder. *A* var*-node below a segment cannot be bound by a* lam*-node within: $\lambda^{-1}(\pi) \subseteq \mathsf{nodes}^-_\tau(\sigma_i)$ for all $i \in 1, 2$ and $\pi \in \mathsf{nodes}^-_\tau(\sigma_i)$.*

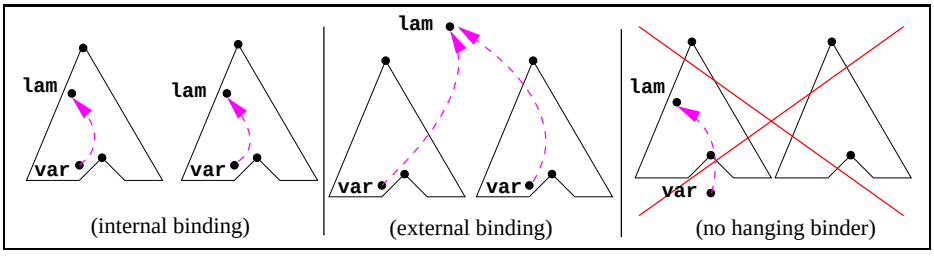


Figure 7.5: Representation of the axioms of parallel lambda binding.

Note that this definition overloads the notion of parallelism $\sigma_1 \sim \sigma_2$. For tree structures this means tree parallelism and for lambda structures binding

parallelism (if not stated differently). Which of the two notions of parallelism is spoken about should always become clear from the respective context.

The following lemma will be required in Section 7.4 to prove Lemma 7.14.

Lemma 7.8 *Parallelism in lambda structures is symmetric: if $\sigma_1 \sim \sigma_2$ holds in a lambda structure then $\sigma_2 \sim \sigma_1$ holds as well.*

Proof: Suppose that σ_1 and σ_2 are parallel segments of a lambda structure (τ, λ) and that c is the correspondence function between them. By assumption, c satisfies the axioms of parallel binding. We have to show that the inverse correspondence function c^{-1} also satisfies these axioms.

Internal binder. Let $\pi, \lambda(\pi) \in \mathsf{nodes}^-_\tau(\sigma_2)$ and $\pi' = c^{-1}(\pi)$ be a node in $\mathsf{nodes}^-_\tau(\sigma_1)$. Since $\lambda(\pi')$ dominates π' there remain only two possibilities:

1. Case $\lambda(\pi') \in \mathsf{nodes}^-_\tau(\sigma_1)$. The **internal binder** axiom for c yields $c(\lambda(\pi')) = \lambda(c(\pi')) = \lambda(\pi)$. We can apply the inverse function c^{-1} on both sides and obtain $\lambda(c^{-1}(\pi)) = c^{-1}(\lambda(\pi))$ as required.
2. Case $\lambda(\pi') \notin \mathsf{nodes}^-_\tau(\sigma_1)$. The **external binder** axiom for c implies $\lambda(\pi') = \lambda(c(\pi')) = \lambda(\pi)$. If π' does not belong to the inner nodes of σ_2 then $\lambda(\pi')$ is a **hanging binder** which is not possible. In the same way, we can prove by induction that $(c^{-1})^n(\pi)$ must belong to the inner nodes of σ_2 for all $n \geq 1$. But this is also impossible as trees are finite.

External binder. Suppose that $\pi \in \mathsf{nodes}^-_\tau(\sigma_2)$ while $\lambda(\pi) \notin \mathsf{nodes}^-_\tau(\sigma_2)$. Let $\pi' = c^{-1}(\pi) \in \mathsf{nodes}^-_\tau(\sigma_1)$. Again, there are two possibilities:

1. Case $\lambda(\pi') \in \mathsf{nodes}^-_\tau(\sigma_1)$. The **internal binder** axiom for c yields $c(\lambda(\pi')) = \lambda(c(\pi')) = \lambda(\pi)$ which is impossible since $\lambda(\pi)$ does not belong to the image $\mathsf{nodes}^-_\tau(\sigma_2)$ of c.
2. Case $\lambda(\pi') \notin \mathsf{nodes}^-_\tau(\sigma_1)$. The **external binder** for c implies $\lambda(\pi') = \lambda(c(\pi')) = \lambda(\pi)$ as required.

No hanging binder. This axiom coincides for c and c^{-1}.

■

7.2.3 Constraint Languages

Given the model-theoretic notions of tree structures and lambda structures we can now define logical languages for their description in the usual Tarski'an manner.

We assume an infinite set $\{X, Y, Z, \ldots\}$ of *node variables* $\mathcal{V}_{\text{node}}$, and define languages of tree descriptions. Figure 7.6 summarises these definitions.

Definition 7.9 *A* dominance constraint *is a conjunction of dominance $X \triangleleft^* Y$ and children-labeling literals $X{:}f(X_1, \ldots, X_n)$ that describe the respective relations in some tree structure. We write $X{=}Y$ to abbreviate $X \triangleleft^* Y \ \wedge \ Y \triangleleft^* X$.*

Definition 7.10 *A* first-order dominance formula ν *is built from dominance constraints and the usual first-order connectives: universal quantification, negation, and conjunction.*

These can also express existential quantification $\exists X.\nu$ and disjunction $\nu_1 \vee \nu_2$ which we freely use. Furthermore, we write $X{\neq}Y$ instead of $\neg X{=}Y$ and $X{\lhd^+}Y$ for $X{\lhd^*}Y \ \wedge \ X{\neq}Y$.

Definition 7.11 *A* parallelism constraint ϕ *is a conjunction of children-labeling, dominance, and parallelism literals* $S_1{\sim}S_2$*, where* S_1 *and* S_2 *are segments with the same number of holes. We use* segment terms S *of the form* $X/X_1,\ldots,X_m$ *to describe segments with* m *holes, given that the values of* X *and* $X_1,\ldots,X_m$ *satisfy the conditions imposed on the root and holes of segments (Definition 7.3).*

Note that a parallelism literal $S_1{\sim}S_2$ requires that the values of S_1 and S_2 are indeed segments. Note also that dominance constraints are subsumed by parallelism constraints, the holes of the segments must be dominated by the root.

Definition 7.12 *A* lambda binding constraint μ *is a conjunction of lambda binding and inverse lambda binding literals:* $\lambda(X){=}Y$ *means that the value of* X *is a* var*-node that is lambda bound by the value of* Y*, while* $\lambda^{-1}(X){\subseteq}\{X_1,\ldots,X_m\}$ *says that all* var*-nodes bound by the* lam*-node denoted by* X *are values of one of* $X_1,\ldots,X_m$*.*

Lambda binding constraints:

$$\mu \quad ::= \quad \lambda(X){=}Y \ \mid \ \lambda^{-1}(X){\subseteq}\{X_1,\ldots,X_m\} \ \mid \ \mu_1 \ \wedge \ \mu_2$$

First-order dominance formulae:

$$\nu \quad ::= \quad X{:}f(X_1,\ldots,X_n) \ \mid \ X{\lhd^*}Y \ \mid \ \forall X.\nu \ \mid \ \neg\nu \ \mid \ \nu_1 \ \wedge \ \nu_2$$

Parallelism constraints:

$$\phi \quad ::= \quad X{:}f(X_1,\ldots,X_n) \ \mid \ X{\lhd^*}Y \ \mid \ S_1{\sim}S_2 \ \mid \ \phi_1 \ \wedge \ \phi_2$$

Figure 7.6: Logical languages for tree and lambda structures.

To keep this section self-contained let us quickly recall some model theoretic notions. We write $Var(\psi)$ for the set of free variables of a formula ψ of one of the above kinds. A *variable assignment* to the nodes of a tree τ is a total function $\alpha : V \rightarrow \mathsf{nodes}_\tau$ where V is a finite subset of node variables. A *solution* of a formula ψ thus consists of a tree structure τ or a lambda structure (τ,λ) and a variable assignment $\alpha : V \rightarrow \mathsf{nodes}_\tau$ such that $Var(\psi) \subseteq V$. Segment terms evaluate to tuples of nodes $\alpha(X/X_1,\ldots,X_n) = \alpha(X)/\alpha(X_1),\ldots,\alpha(X_n)$ which may or may not be segments. Apart from this, we require as usual that a

formula evaluates to true in all solutions. We write $\tau, \alpha \models \psi$ if τ, α is a solution of ψ, and similarly $(\tau, \lambda), \alpha \models \psi$. A formula is *satisfiable* if it has a solution. We write $\alpha_{|V'}$ for the restriction of a variable assignment α to the variables in V', (like it is done for substitutions restrictions).

Remark: It is interesting to notice here the different point of view when talking about trees (terms) between the Constraint Language for Lambda Structures and the (Context) Unification framework. In the first, variables refer to *nodes* of trees while in the second, variables refer to *subtrees* or to *contexts*. One could also think that in the first language, there is just one tree and the literals partially describe the relation between the nodes of that tree, and that one has to find the tree that satisfies those relations, while in unification, there are two trees (in an equation) and one has to make both trees be equal by "filling" the pieces of the puzzle variables with appropriate subtrees and contexts. This difference is properly studied in Section 7.8.

Before starting with the translation, we want to show why the Constraint Language for Lambda Structures is better than Unification for the treatment of semantics underspecification problems, mainly in the requirements of binding.

7.3 Example

As we have already said, the prime application of the Constraint Language for Lambda Structures is to describe natural language semantics using underspecified representations (Egg *et al.*, 2001; Erk *et al.*, 2002) and, as the following example shows, the role of parallel lambda binding is crucial.

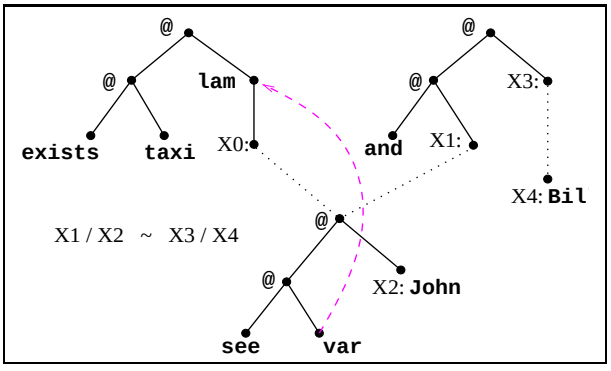


Figure 7.7: The graph of the constraint language for lambda structures formula for the semantics of the sentence: *John saw a taxi and so did Bill.*

Consider for instance the sentence: *John saw a taxi and so did Bill.* This elliptic sentence has two possible meanings that can be represented by the following λ-terms of Boolean type in higher-order logics:

1. There exists a taxi t seen by John and Bill:

$$@(@(exists, taxi), \lambda t.@(@(and, @(@(see, t), john)), @(@(see, t), bill)))$$

2. There exists a taxi t_1 seen by John and a taxi t_2 seen by Bill.

$$\begin{aligned} @(@(and, \quad & @(@(exists, taxi), \lambda t_1.@(@(see, t_1), john))), \\ & @(@(exists, taxi), \lambda t_2.@(@(see, t_2), bill))) \end{aligned}$$

The ellipses require the meanings of source and target sentence to be the same, except for the contribution of the contrasting elements *john* and *bill*. Note that the elided parallel segments (shown in grey), are different in the two previous readings.

This example also illustrates the interaction of parallelism and lambda binding: in the first reading, both occurrences of t are bound by the same lambda binder outside the parallel segments, while in the second case, the occurrence of t_1 and t_2 are bound by distinct but corresponding lambda binders inside the parallel segments.

The lambda terms of both readings satisfy the constraint described by the graph in Figure 7.7. The graph representation uses solid lines for children-labeling literals, dotted black lines for dominance and dashed pink edges from var to lam-labeled nodes for binding literals. Parallelism literals are made explicit by means of writing them in the picture. These graph representation serves also for describing lambda structures with their relations.

The binding edge in the picture represents the binding of the first occurrence of t in the first reading and of the unique occurrence of t_1 in the second reading. To deal with λ-bindings, no variable names are needed in the constraint language, this avoids variable renaming and capturing once and for all.

Another advantage of the constraint in Figure 7.7, hence of the Constraint Language for Lambda Structures, is that it can be derived by compositional semantics construction from a parse tree (Egg *et al.*, 2001). The graphical part is a dominance constraint that describes the meanings of the source sentence and the conjunction. But it leaves underspecified where the fragment with the lambda binder @(@(exists, taxi), lam( )) should be placed, either above the conjunction or below its first argument that starts at node X_1.

The parallelism constraint $X_1/X_2 \sim X_3/X_4$ expresses the parallelism requirement of the ellipses. The parallel segments defined by X_1/X_2 and X_3/X_4 must have equal tree structure and parallel binding relations.

But this parallel lambda binding cannot be expressed in Linear Second-Order Unification or Context Unification in any naive sense. In fact, this constitutes a serious problem to Linear Second-Order Unification or other kinds of Higher-Order Unification approaches dealing with ellipsis resolution (Pinkal, 1995; Shieber *et al.*, 1996). One might hope for instance, to describe the first reading above through the following lambda term:

$$@(@(exists, taxi), \lambda t.@(@(and, C(john)), C(bill))) \tag{7.1}$$

where $C(john) = @(@(see, john), t)$. The problem is that every unifier for the linear second-order equation:

$$C(john) \stackrel{?}{=} @(@(see, john), t)$$

must map the second-order variable C to the linear lambda term $\lambda j.@(@(see, j), t)$ with free variable t. This unifier is not valid in Linear Second-Order Unification since t would get captured by λt when applying the substitution to the term 7.1, while Definition 7.7 allows this kind of external binder.

In fact, this example illustrates that the nature of the lambda binding relation in the presence of parallelism in the Constraint Language for Lambda Structures is not the same as the one of λ-calculus in Second-Order Unification (which, as we have shown, models parallelism by means of multiple occurrences of the same variable). Hence, this fact suggests that to be able to deal with this kind of lambda bindings with some Second-Order Unification fragment, we need something more.

7.4 The Non-intervenance Property

We now present the "corner-stone" of our translation to get rid of lambda binding constraints.

The idea behind our translation is to eliminate lambda bindings from the parallelism and lambda-binding constraints formulae, by naming variable binders and bound variables. This means that we want to obtain similar parallelism constraints that use *named* labels lam_u and var_u, instead of *anonymous* labels lam and var and of lambda-binding constraints.

In order to avoid undesired variable capture, we would like to associate different names to different lambda binders. But unfortunately we cannot always do so in the presence of parallelism because corresponding lam-nodes have to carry the same label lam_u and corresponding var-nodes the same label var_u.

Given that we cannot freely assign fresh names, we are faced with the danger of capturing which we have to avoid. The simplest idea would be to forbid trees where some node with label lam_u intervenes (occurs) between any two other nodes with labels lam_u and var_u. This restriction can be easily expressed by a closed first-order dominance formula or could also be directly checked by a tree automaton in some tree-regular constraint.

Unfortunately, the above restriction is too restrictive and thus not correct because forbids valid lambda structures, as illustrated by the following tree structure with named lam and var-nodes, where the parallelism relation $\epsilon/11 \sim 11/1111$ holds:

$$\mathsf{lam}_u(@(\mathsf{lam}_u(@(a, \mathsf{var}_u)), \mathsf{var}_u))$$

and where the lam-node 11 that corresponds to lam-node ϵ, occurs between ϵ and the var-node 1112 (bound by 11).

It can always happen that a corresponding lam_u takes place above a binding lam_u-node, so that the binding lam_u intervenes between the corresponding

lam_u-node and one of the var_u-nodes bound by it. Thus we need a refined *non-intervenance property*, which holds in a tree structure stating that: *no corresponding* lam_u *may intervene between a* lam_u*-node and one of the* var_u*-nodes bound by it.*

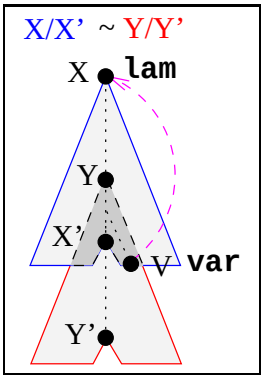


Figure 7.8: Intervenance.

Example 7.13 The following parallelism constraint that is drawn in Figure 7.8 is unsatisfiable:

$$X \lhd^+ Y \lhd^+ X' \ \wedge \ X/X' \sim Y/Y' \ \wedge \ Y \lhd^+ V \ \wedge \ \lambda(V) = X$$

This is proved by Lemma 7.14. The problem is that lam-node Y must correspond to X but intervene between X and the var-node V that X binds. ■

Lemma 7.14 *Let* (τ, λ) *be a lambda structure with parallel segments* σ *and* σ' *that correspond via the correspondence function* c*. For all* π *with* $\lambda(\pi) \in \mathsf{nodes}_\tau^-(\sigma)$ *it is not the case that* $\lambda(\pi) \lhd^+ c(\lambda(\pi)) \lhd^+ \pi$*.*

Proof: We suppose that $\lambda(\pi) \in \mathsf{nodes}_\tau^-(\sigma)$ and $\lambda(\pi) \lhd^+ c(\lambda(\pi)) \lhd^+ \pi$ and derive a contradiction. The segments σ and σ' must overlap such that the root of σ dominates that of σ' properly.

$$\mathsf{root}(\sigma) \lhd^+ \mathsf{root}(\sigma')$$

Notice that π must belong to the inner nodes of segment σ, $\pi \in \mathsf{nodes}_\tau^-(\sigma)$, since otherwise $\lambda(\pi)$ would be a **hanging binder**.

Now suppose π does not belong to the inner nodes of the lower segment $\pi \notin \mathsf{nodes}_\tau^-(\sigma')$. First of all, notice that π must be dominated by a hole of σ' since otherwise $\pi \perp \mathsf{root}(\sigma')$ and the lemma would follow trivially.

We also know that $c(\lambda(\pi)) \lhd^+ \pi$ and that it belongs to the inner nodes of segments σ and σ', therefore we can apply the **internal binder** axiom and we get that $c(\lambda(\pi)) = \lambda(c(\pi))$, to avoid the **hanging binder** axiom violation, $c(\pi)$

must belong to the inner nodes of segments σ and σ', which corresponds to the next case.

Consider now the case in which π also belongs to the inner nodes of the lower segment $\pi \in \mathsf{nodes}^-_\tau(\sigma')$. We prove the following property inductively and thus derive a contradiction: For all $\pi \in \mathsf{nodes}^-_\tau(\sigma) \cap \mathsf{nodes}^-_\tau(\sigma')$ it is impossible that:

$$\lambda(\pi) \lhd^+ c(\lambda(\pi)) \lhd^+ \pi.$$

The proof is by well-founded induction on the length of the word π.

1. Case $\mathsf{root}(\sigma') \lhd^* \lambda(\pi) \lhd^+ c(\lambda(\pi))$. Let $\pi' = c^{-1}(\pi)$ be an inner node of σ. The length of the word π' is properly smaller than the length of π. Since π' belongs to the inner nodes of σ, the axiom for **internal binder** can be applied to the correspondence function c yielding $c(\lambda(\pi')) = \lambda(c(\pi'))$ and thus $c(\lambda(\pi')) = \lambda(\pi)$. The node $\lambda(\pi')$ must properly dominate both $c(\lambda(\pi'))$ and π'. The address (length) of $c(\lambda(\pi'))$ is smaller than that of π', so that:
$$\lambda(\pi') \lhd^+ c(\lambda(\pi')) \lhd^+ \pi'$$
This is impossible as stated by induction hypothesis applied to π'.

2. Case $\lambda(\pi) \lhd^+ \mathsf{root}(\sigma') \lhd^* c(\lambda(\pi))$. Let $\pi' = c^{-1}(\pi)$ be an inner node of σ. Since π is externally bound outside σ', the axiom for **external binder** applies to the inverse correspondence function c^{-1} by Lemma 7.8 and yields $\lambda(\pi') = \lambda(\pi)$. By now, π' is internally bound in σ. The axiom for **internal binder** applied to the correspondence function c yields: $c(\lambda(\pi')) = \lambda(c(\pi'))$ which is $c(\lambda(\pi)) = \lambda(\pi)$. This clearly contradicts $\lambda(\pi) \lhd^+ c(\lambda(\pi))$.

■

This Lemma will be crucial in next Section to be able of eliminating lambda binding constraints soundly. The problem could be that when we translate a lambda binding constraint formula that is unsatisfiable because of the conditions of Lemma 7.14, into a parallelism constraint formula where lambda binding constraints have been removed by means of labeling lam and var nodes, we could obtain a satisfiable parallelism constraint formula. We will avoid this by means of first-order dominance formulae that will forbid this situation.

7.5 Elimination of Lambda Binding Constraints

We now give a translation that eliminates lambda binding literals while preserving satisfiability. The procedure is highly non-deterministic and introduces first-order dominance formulae to express and ensure consistent naming of bound variables.

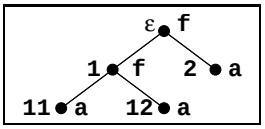
We impose the non-intervenance property of Lemma 7.14 when expressing the lambda binding predicate $\mathsf{bind}_u(X, Y)$ in Figure 7.9. This is defined by using the predicate $\mathsf{intervene}_{\mathsf{lam}_u}(Y, X)$, which expresses, via first-order dominance formulae, that some lam_u-node intervenes between X and Y.

$$\mathsf{intervene}_{\mathsf{lam}_u}(Y,X) \quad = \quad \exists Z \exists Z'.\ Y \lhd^+ Z \lhd^+ X \ \wedge\ Z{:}\mathsf{lam}_u(Z')$$

$$\mathsf{bind}_u(X,Y) \ = \ \exists Z\ (Y{:}\mathsf{lam}_u(Z) \wedge Z \lhd^* X \wedge X{:}\mathsf{var}_u) \wedge \neg \mathsf{intervene}_{\mathsf{lam}_u}(Y,X)$$

Figure 7.9: Non-intervenance and lambda binding.

Guessing Correspondence Classes. The lam and var-nodes that correspond via a correspondence function, clearly have to carry the same node labels. But we have to be a little more careful since we may have several correspondence functions for several pairs of parallel segments. We say that two nodes are in the same correspondence class for a given set of correspondence functions $\{c_1, \ldots, c_n\}$ if they belong to the symmetric, reflexive and transitive closure of the common graph of these functions. Consider for instance the following tree structure τ:



and the correspondence functions c_1 and c_2 defined by $c_1(11) = 12$ and $c_2(12) = 2$. Then, $\mathcal{C}_{\tau,\{c_1,c_2\}} = \{(11,11), (11,12), (12,11), (12,12), (12,2), (2,11), (2,12), (2,2)\}$ is the symmetric, reflexive and transitive closure of $\{c_1, c_2\}$ in τ.

Given a parallelism and lambda binding constraint[1] $\phi \ \wedge \ \mu$ we consider the set of correspondence functions for pairs of segments that are required to be parallel in ϕ. But how can we know a priori whether two variables of $\phi \ \wedge \ \mu$ will denote nodes in the same correspondence class? We want to guess an equivalence relation e between variables of $\phi \wedge \mu$ depending on a solution τ, α for $\phi \wedge \mu$, such that for any two variables X and Y of $\phi \ \wedge \ \mu$, $(X,Y) \in e \iff (\alpha(X), \alpha(Y)) \in \mathcal{C}_{\tau,\{c_1,\ldots,c_n\}}$, where $\{c_1, \ldots, c_n\}$ are the correspondence functions for pairs of segments that are required to be parallel in ϕ. We cannot do this a priori, but we simply guess the equivalence relation between them as there are only finitely many possibilities for the finitely many variables of the given formula.

We want to guess one of the possible partitions into correspondence classes for variables of ϕ. Instead, we simply guess an equivalence relation on the variables of ϕ, and as our proofs will show, we don't have to express that equivalent variables denote values in the same correspondence class.

Definition 7.15 *Let*

$$\mathsf{equ}_\phi = \{e \mid e \subseteq Var(\phi) \times Var(\phi) \textit{ equivalence relation}\}$$

be the set of possible equivalence relations on the variables of ϕ. We write $e(X)$ for the equivalence class of some variable $X \in Var(\phi)$ with respect to e, but

[1]Recall that we use ϕ to denote parallelism constraints and μ to denote lambda binding constraints.

consider equivalence classes of distinct equivalence relations to be distinct. Let

$$\mathsf{names}_e = \{e(X) \mid X \in Var(\phi)\}$$

be the set of names of e which contains all equivalence classes of e. Note that names_e is finite for all $e \in \mathsf{equ}_\phi$, and that names_e and $\mathsf{names}_{e'}$ are disjoint for distinct equivalence classes e and e'.
We now fix a parallelism and lambda binding constraint formula $\Phi = \phi \wedge \mu$ and guess an equivalence relation $e \in \mathsf{equ}_\phi$ that determines the translation $[\ ]_e$ presented in Figure 7.10.

$$\begin{array}{rcl}
[\lambda(X){=}Y]_e & = & \mathsf{bind}_{e(Y)}(X,Y) \\
[\lambda^{-1}(Y){\subseteq}\{X_1,\ldots,X_n\}]_e & = & \forall X.\mathsf{bind}_{e(Y)}(X,Y) \rightarrow \vee_{i=1}^{n} X{=}X_i \\
[Y{:}\mathsf{lam}(Z)]_e & = & Y{:}\mathsf{lam}_{e(Y)}(Z) \\
[X{:}\mathsf{var}]_e & = & \exists Y. \bigvee_{\{Z \mid Z:\mathsf{lam}(Z')\in\phi\}} \mathsf{bind}_{e(Z)}(X,Y) \\
[Y{:}f(Y_1\ldots,Y_n)]_e & = & Y{:}f(Y_1\ldots,Y_n) \qquad \text{if } f \notin \{\mathsf{lam},\mathsf{var}\} \\
[X{\lhd^*}Y]_e & = & X{\lhd^*}Y \\
[S_1 \sim S_2]_e & = & S_1 \sim S_2 \ \wedge\ \mathsf{external{-}binder}_e(S_1,S_2) \ \wedge \\
 & & \mathsf{no{-}hang{-}binder}_e(S_1) \ \wedge \\
 & & \mathsf{no{-}hang{-}binder}_e(S_2) \\
[\Phi_1 \ \wedge\ \Phi_2]_e & = & [\Phi_1]_e \ \wedge\ [\Phi_2]_e
\end{array}$$

Figure 7.10: Translation Literals. Naming variable binder for correspondence classes e. Auxiliary predicates in Figure 7.11.

This translation maps Φ to a parallelism constraint plus first-order dominance formulae $\phi' \wedge \nu$ over the following signature Σ_ϕ which extends Σ with finitely many symbols:

$$\Sigma_\phi = \Sigma \uplus \{\mathsf{lam}_u, \mathsf{var}_u \mid u \in \mathsf{names}_e,\ e \in \mathsf{equ}(\phi)\}$$

The literal $\lambda(X) = Y$ is translated to $\mathsf{bind}_{e(Y)}(X,Y)$ as explained before (see Figure 7.9). This ensures that all corresponding nodes in e are translated with the same name $e(Y)$. The axioms about **external binding** *and* **no hanging binder** *are stated by first-order dominance formulae in the translation of parallelism literals (see Figure 7.11).*
Note that the axiom of **internal binding** *will always be satisfied without extra requirements.*
We have to ensure that all var_u-nodes in the solutions of translated constraints will be bound by some lam_u-node. Let $\mathsf{no{-}free{-}var}_e$ be as defined in Figure 7.11. Recall that the lambda binding function is total. We then define the complete translation $[\Phi]$ by:

$$[\Phi] = \bigvee_{e\in\mathsf{equ}_\phi} [\Phi]_e \ \wedge\ \mathsf{no{-}free{-}var}_e$$

$$
\begin{array}{rcl}
\mathsf{inside}(X,\ Y/Y_1,\ldots,Y_n) & = & Y{\lhd^*}X\ \wedge\ (\bigvee_{i\in\{1..n\}} X{\lhd^+}Y_i) \\
\mathsf{root}(X,\ Y/Y_1,\ldots,Y_n) & = & X{=}Y \\
\mathsf{no{-}hang{-}binder}_e(S) & = & \bigwedge_{u\in\mathsf{names}_e} \mathsf{no{-}hang{-}binder}_u(S) \\
\mathsf{no{-}hang{-}binder}_u(S) & = & \\
\multicolumn{3}{c}{\neg(\exists Y \exists Z.\ \mathsf{bind}_u(Y,Z)\ \wedge\ \neg\mathsf{inside}(Y,S)\ \wedge\ \mathsf{inside}(Z,S))} \\
\mathsf{external{-}binder}_e(S_1,S_2) & = & \bigwedge_{u\in\mathsf{names}_e} \mathsf{external{-}binder}_u(S_1,S_2) \\
\mathsf{external{-}binder}_u(S_1,S_2) & = & \\
\multicolumn{3}{c}{\forall Z_1 \forall Z_2 \forall Y.\ (\mathsf{bind}_u(Z_1,Z_2) \wedge \mathsf{inside}(Z_1,S_1) \wedge \neg\mathsf{inside}(Z_2,S_1) \wedge \mathsf{root}(Y,S_2))} \\
\multicolumn{3}{c}{\rightarrow (Z_2{\lhd^*}Y\ \wedge\ \neg\mathsf{intervene}_{\mathsf{lam}_u}(Z_2,Y))} \\
\multicolumn{3}{l}{\mathsf{no{-}free{-}var}_e = \bigwedge_{u\in\mathsf{names}_e} \forall X.\ X{:}\mathsf{var}_u \rightarrow (\ \exists Y \exists Z.\ Y{:}\mathsf{lam}_u(Z)\ \wedge\ Y{\lhd^*}X\ )}
\end{array}
$$

Figure 7.11: Auxiliary predicates.

The following two lemmas, Lemma 7.16 and Lemma 7.17, entail that our translation preserves satisfiability, Proposition 7.18.

Lemma 7.16 *Let Φ be a conjunction of a parallelism and lambda binding constraint and $e \in \mathsf{equ}(\Phi)$ an equivalence relation on $Var(\Phi)$. If $[\Phi]_e \wedge \mathsf{no{-}free{-}var}_e$ is satisfiable then Φ is satisfiable.*

Proof: Let τ be a tree structure and $\alpha : V \rightarrow \mathsf{nodes}_\tau$ an assignment with

$$\tau, \alpha \models [\Phi]_e\ \wedge\ \mathsf{no{-}free{-}var}_e$$

We now define a lambda structure $(p(\tau), \lambda)$ of signature Σ by projecting labels away. The nodes of $p(\tau)$ are the nodes of τ. Let projection $\mathsf{proj} : \Sigma_\phi \rightarrow \Sigma$ be the identity function except that $\mathsf{proj}(\mathsf{lam}_u) = \mathsf{lam}$ and $\mathsf{proj}(\mathsf{var}_u) = \mathsf{var}$ for any $u \in \mathsf{names}_e$. The labels of $p(\tau)$ satisfy for all $\pi \in \mathsf{nodes}_\tau$:

$$p(\tau)(\pi) = \mathsf{proj}(\tau(\pi))$$

We define the lambda binding function $\lambda : p(\tau)^{-1}(\mathsf{var}) \rightarrow p(\tau)^{-1}(\mathsf{lam})$ as follows: Let π be a node such that $p(\tau)(\pi) = \mathsf{var}$. There exists a unique name u such that $\tau(\pi) = \mathsf{var}_u$. We define $\lambda(\pi)$ to be the lowest ancestor of π that is labeled by lam_u. This is the unique node in $p(\tau)$ that satisfies $\mathsf{bind}_u(\pi, \lambda(\pi))$. It exists since we required $\tau, \alpha \models \mathsf{no{-}free{-}var}_e$.

It remains to prove that $(p(\tau), \lambda), \alpha$ is indeed a solution of Φ, i.e. whether $(p(\tau), \lambda), \alpha$ satisfies all literals of Φ.

- $X{\lhd^*}Y$ in Φ: The dominance relation of τ coincides with that of $p(\tau)$. Since $\tau, \alpha \models X{\lhd^*}Y$ it follows that $(p(\tau), \lambda), \alpha \models X{\lhd^*}Y$.
- $X{:}f(X_1,\ldots,X_n)$ in Φ where $f \notin \{\mathsf{lam}, \mathsf{var}\}$. The children-labeling relation of τ coincides with that of $p(\tau)$, so there is no difference again.

- $X{:}\mathsf{var}$ in Φ: Notice that $\mathsf{bind}_{e(Y)}(X,Y)$ enforces X to be a $\mathsf{var}_{\mathsf{e(Y)}}$-labeled node in $[\Phi]_e$, which implies $(p(\tau),\lambda),\alpha \models X{:}\mathsf{var}$ by the definition of p.
- $X{:}\mathsf{lam}(Z)$ in Φ: Now, the literal $X{:}\mathsf{lam}_{e(X)}(Z)$ belongs to $[\Phi]_e$. Thus, $\tau,\alpha \models X{:}\mathsf{lam}_{e(X)}(Z)$ which implies $(p(\tau),\lambda),\alpha \models X{:}\mathsf{lam}(Z)$ by the definition of p.
- $\lambda(X){=}Y$ in Φ: Let $\tau,\alpha \models [\lambda(X){=}Y]_e$. By definition of the translation $[\lambda(X){=}Y]_e$ this means that $\tau,\alpha \models \mathsf{bind}_{e(Y)}(X,Y)$. In particular, it follows that $\alpha(Y)$ is the lowest $\mathsf{lam}_{e(Y)}$-labeled ancestor of the $\mathsf{var}_{e(Y)}$-labeled node $\alpha(X)$. The definition of the lambda-binding relation of $p(\tau)$ yields $(p(\tau),\lambda),\alpha \models \lambda(X){=}Y$ as required.
- $\lambda^{-1}(Y) \subseteq \{X_1,\ldots,X_n\}$ in Φ: the proof for this literal follows straightforward using similar arguments as for the previous one.

Consider at last, $S_1{\sim}S_2$ in Φ: This is the most complicated case. If τ,α satisfies this literal then clearly, $(p(\tau),\lambda),\alpha$ satisfies the correspondence conditions for all children-labeling relations. We have to verify that $(p(\tau),\lambda)$ also satisfies the conditions of parallel binding. Let $c : \mathsf{nodes}^-_\tau(\alpha(S_1)) \to \mathsf{nodes}^-_\tau(\alpha(S_2))$ be the correspondence function between $\alpha(S_1)$ and $\alpha(S_2)$ which exists since $\tau,\alpha \models [\Phi]_e$.

Internal binder. Let $\lambda(\pi_1){=}\pi_2$ for some $\pi_1,\pi_2 \in \mathsf{nodes}^-_\tau(\alpha(S_1))$. By definition of λ, there exists a name u such that $\tau(\pi_1) = \mathsf{var}_u$ and π_2 is the lowest node above π_1 with $\tau(\pi_2) = \mathsf{lam}_u$. Since the labels of the nodes on the path between π_1 and π_2 are equal to the labels of the nodes of the corresponding path from $c(\pi_1)$ to $c(\pi_2)$ it follows that $\tau(c(\pi_1)) = \mathsf{var}_u$, $\tau(c(\pi_2)) = \mathsf{lam}_u$ and that no node in between is labeled with lam_u. Thus, $\lambda(c(\pi_1)) = c(\pi_2)$.

External binder. Suppose that $\lambda(\pi_1){=}\pi_2$ for two nodes $\pi_1 \in \mathsf{nodes}^-_\tau(\alpha(S_1))$ and $\pi_2 \notin \mathsf{nodes}^-_\tau(\alpha(S_1))$. There exists a name u such that $\tau(\pi_1) = \mathsf{var}_u$ and π_2 is the lowest ancestor of π_1 with $\tau(\pi_2) = \mathsf{lam}_u$. By correspondence, it follows that $\tau(c(\pi_1)) = \mathsf{var}_u$ and that no lam_u-node lies on the path from the root of segment $\alpha(S_2)$ to $c(\pi_1)$. The predicate $\mathsf{external{-}binder}_u(S_1,S_2)$ requires that π_2 dominates that root of $\alpha(S_2)$ and that no lam_u-node intervenes on the path from π_2 to that root. Thus, π_2 is the lowest ancestor of $c(\pi_1)$ that satisfies $\tau(\pi_2) = \mathsf{lam}_u$, i.e. $\lambda(c(\pi_1)) = \pi_2$.

No hanging binder. Let S be either of the segment terms S_1 or S_2. Suppose that $\lambda(\pi_1){=}\pi_2$ for some nodes $\pi_1 \notin \mathsf{nodes}^-_\tau(S)$ and $\pi_2 \in \mathsf{nodes}^-_\tau(S)$. There exists a name $u \in \mathsf{names}_e$ such that $\tau(\pi_1) = \mathsf{var}_u$ and π_2 is the lowest ancestor of π_1 with $\tau(\pi_2) = \mathsf{lam}_u$. This contradicts that τ,α solves $\mathsf{no{-}hang{-}binder}_u(S)$ as required by $[S_1{\sim}S_2]_e$.

■

Lemma 7.17 *If Φ has a solution whose correspondence classes induce the equivalence relation e then $[\Phi]_e \wedge \mathsf{no{-}free{-}var}_e$ is satisfiable.*

Proof: Let Φ be a conjunction of a parallelism and lambda binding constraint over signature Σ and $(\tau, \lambda), \alpha$ a solution of it. Let $\{c_1, \ldots, c_n\}$ be the correspondence functions for the parallel segments $\alpha(S) \sim \alpha(S')$ where $S \sim S'$ belongs to ϕ. Let $c \subseteq \mathsf{nodes}_\tau \times \mathsf{nodes}_\tau$ be the reflexive, symmetric, and transitive closure of $\{c_1, \ldots, c_n\}$, and $e \in \mathsf{equ}(\Phi)$ be the relation $\{(X, Y) \mid (\alpha(X), \alpha(Y)) \in c\}$.

We define $\mathsf{tree}_e(\tau, \lambda)$ as a tree over the extended signature Σ_ϕ whose nodes are those of τ and whose labeling function satisfies for all $\pi \in \mathsf{nodes}_\tau$ that:

$$\mathsf{tree}_e(\tau,\lambda)(\pi) = \begin{cases} \mathsf{lam}_{e(X)} & \text{if } (\pi, \alpha(X)) \in c, \tau(\pi) = \mathsf{lam},\ X \in Var(\Phi) \\ \mathsf{var}_{e(X)} & \text{if } (\lambda(\pi), \alpha(X)) \in c, \tau(\pi) = \mathsf{var}, X \in Var(\Phi) \\ \tau(\pi) & \text{otherwise} \end{cases}$$

We now prove that $\mathsf{tree}_e(\tau, \lambda), \alpha$ solves $[\Phi]_e$, i.e. all of its conjuncts. This can be easily verified for dominance, children-labeling, and parallelism literals in $[\Phi]_e$. Notice in particular that corresponding lam-nodes in τ are assigned the same labels in $\mathsf{tree}_e(\tau, \lambda)$. Next, we consider the first-order formulae introduced in the translation of lambda binding and parallelism literals.

1. Case $\mathsf{bind}_{e(Y)}(X, Y)$ in $[\Phi]_e$. This requires either $\lambda(X){=}Y$ or $\lambda^{-1}(Y) \subseteq \{X_1, \ldots, X_n\}$ or $X{:}\mathsf{var}$ in Φ. Let us consider the first case. The corresponding cases of $\lambda^{-1}(Y) \subseteq \{X_1, \ldots, X_n\}$ in Φ, and of $X{:}\mathsf{var}$ in Φ are quite similar. It then clearly holds that $\mathsf{tree}_e(\tau, \lambda)(\alpha(X)) = \mathsf{var}_{e(Y)}$ and $\mathsf{tree}_e(\tau, \lambda)(\alpha(Y)) = \mathsf{lam}_{e(Y)}$. Furthermore $\alpha(Y) \lhd^+ \alpha(X)$. It remains to show for $\mathsf{tree}_e(\tau, \lambda)$ that no $\mathsf{lam}_{e(Y)}$-node intervenes between $\alpha(X)$ and $\alpha(Y)$. We do this by contradiction. Suppose there exists π such that $\alpha(Y) \lhd^+ \pi \lhd^+ \alpha(X)$ and $\mathsf{tree}_e(\tau, \lambda)(\pi) = \mathsf{lam}_{e(Y)}$. By definition of $\mathsf{tree}_e(\tau, \lambda)$ there exists Z such that $(\pi, \alpha(Z)) \in c$ and $e(Y) = e(Z)$. Hence $(\alpha(Y), \alpha(Z)) \in c$ and thus $(\pi, \alpha(Y)) \in c$. But this is impossible by the non-intervenance property shown in Lemma 7.14: no lam-node such as π that corresponds to $\alpha(Y)$ intervene between $\alpha(Y)$ and the var-node $\alpha(X)$ bound by it.

2. Case $\mathsf{external{-}binder}_u(S_1, S_2)$ in $[\Phi]_e$ where $S_1 \sim S_2$ in Φ and $u \in \mathsf{names}_e$. By contradiction. Suppose that there exist $\pi_1 \in \mathsf{nodes}_\tau(\alpha(S_1))$, $\pi_2 \notin \mathsf{nodes}_\tau(\alpha(S_1))$ such that $\mathsf{tree}_e(\tau, \lambda)(\pi_1) = \mathsf{var}_u$ and π_2 is the lowest ancestor of π_1 with $\mathsf{tree}_e(\tau, \lambda)(\pi_2) = \mathsf{lam}_u$. Furthermore, assume either not $\pi_2 \lhd^* \mathsf{root}(\alpha(S_2))$ or $\mathsf{intervene}_{\mathsf{lam}_u}(\pi_2, \mathsf{root}(\alpha(S_2)))$. The first choice is impossible since the binding axioms would be violated otherwise. (The correspondent of an externally bound node must be bound externally). Let π_1' be the correspondent of π_1 with respect to the parallel segment $\alpha(S_1) \sim \alpha(S_2)$. By Lemma 7.14 we know that no lam-node corresponding to π_2 can intervene between π_2 and π' and thus between π_2 and $\mathsf{root}(S_2)$. This also contradicts the second choice: $\mathsf{intervene}_{\mathsf{lam}_u}(\pi_2, \mathsf{root}(\alpha(S_2)))$.

3. Case $\mathsf{no{-}hang{-}binder}_e(S)$ in $[\Phi]_e$ where S is either S_1 or S_2 and $S_1 \sim S_2$ in Φ. Let us proceed by contradiction. Assume that it is not satisfied by $\mathsf{tree}_e(\tau, \lambda), \alpha$, then there must exist a name $u \in \mathsf{names}_\phi$ and two nodes

π_1, π_2 such that $\mathsf{tree}_e(\tau,\lambda)(\pi_1) = \mathsf{lam}_u$ and $\mathsf{tree}_e(\tau,\lambda)(\pi_2) = \mathsf{var}_u$. Even more, $\pi_1 \in \mathsf{nodes}_\tau(\alpha(S))$, $\pi_2 \notin \mathsf{nodes}_\tau(\alpha(S))$ and there does not exist a third node π_3 between π_1 and π_2. Then, by Lemma 7.14, π_1 cannot be a corresponding node of the lambda binding node of π_2, therefore, by definition of $\mathsf{tree}_e(\tau,\lambda)$, $\lambda(\pi_1) = \pi_2 \in \lambda$, but this is impossible because $(\tau,\lambda),\alpha$ must satisfy the **no hanging binder** condition.

4. Finally, we prove that $\mathsf{tree}_e(\tau,\lambda),\alpha$ satisfies $\mathsf{no{-}free{-}var}_e$. The only var_u labeled nodes in $\mathsf{tree}_e(\tau,\lambda)$ are the ones that were var labeled in τ. As (τ,λ) is a lambda structure, there are not free var labeled nodes, and since the construction of $\mathsf{tree}_e(\tau,\lambda)$ only labels nodes by var_u if they are in $\lambda^{-1}(\pi)$ for some node π such that $\mathsf{tree}_e(\tau,\lambda)(\pi) = \mathsf{lam}_u$, there cannot be any var_u labeled node not being under a lam_u labeled node as required by $\mathsf{no{-}free{-}var}_e$.

■

Proposition 7.18 *A parallelism and lambda binding constraint $\phi \wedge \mu$ is satisfiable if and only if its translation $[\phi \wedge \mu]$ is.*

Theorem 7.19 *Satisfiability of Parallelism and Lambda Binding Constraints can be reduced in non-deterministic polynomial time to satisfiability of Parallelism Constraints with First-Order Dominance Formulae.*

Proof: The result follows from Proposition 7.18 and observing that all guesses that we need are polynomially bounded on the size of the problem. ■

7.6 The Monadic Second-Order Dominance Logic and Tree-Regular Constraints

We have removed the lambda binding constraints by adding first-order dominance formulae. Now we introduce a superlanguage of these last formulae, the so called *Monadic Second-Order Dominance Logic*, and we show that tree-regular constraints and second-order dominance formulae are satisfaction equivalent. Then, in Section 7.7, we prove that this satisfaction equivalence is preserved in presence of parallelism constraints, and in Section 7.8 we show that this also holds for Context Unification with tree-regular constraints.

7.6.1 Tree-Regular Constraints

We next introduce tree-regular constraints from the node perspective of trees[2] and show how to express them in logics. A *tree-regular constraint* ξ has the form:

$$\xi \quad ::= \quad \mathsf{tree}(X) \in \mathcal{L}(\mathcal{A}) \mid \xi_1 \wedge \xi_2$$

[2]Recall that now, X is a node variable, and not a first-order variable denoting a tree.

Interpreted over a tree τ, the term $\mathsf{tree}(X)$ denotes the subtree of τ rooted by X, while $\mathcal{L}(\mathcal{A})$ stands for the tree language accepted by the tree automaton $\mathcal{A}$ over the assumed signature Σ.

But which properties of trees can be expressed by tree-regular constraints? Can we express, for instance, the first-order dominance formula which requires that no f labeled node intervenes between nodes X and Y? We will see that we can in Example 7.23.

7.6.2 Monadic Second-Order Dominance Logic

Now we define the *Monadic Second-Order Dominance Logic* to be the monadic second-order logic over dominance constraints, hence, of ground terms. Note that monadic second-order logics were already investigated for many other graph structures (see Courcelle (2000)).

Apart from the assumed node variables $\mathcal{V}_{\text{node}}$, we also consider an infinite set $\mathcal{V}_{\text{set}}$ of *monadic second-order variables* $\{A, B, \ldots\}$ that denote sets of nodes. The formulae ψ of Monadic Second-Order Dominance Logic have the form:

$$\psi \quad ::= \quad X \triangleleft^* Y \mid X{:}f(X_1, \ldots, X_n) \mid X \in A \mid \psi \ \wedge \ \psi' \mid \neg\psi \mid \exists X.\ \psi \mid \exists A.\ \psi$$

Beyond conjunctions of dominance and children-labeling literals, there are membership constraints, existential quantification over nodes and sets, negation, and thus universal quantification.

The Monadic Second-Order Dominance Logic is interpreted over ground terms. Every ground term τ now defines a two sorted domain: $\mathsf{domain}_\tau = \mathsf{nodes}_\tau \uplus 2^{\mathsf{nodes}_\tau}$. Variable assignments to a tree τ are functions $\alpha : V \to \mathsf{domain}_\tau$ defined on a finite set $V \subseteq \mathcal{V}_{\text{node}} \uplus \mathcal{V}_{\text{set}}$ which map node variables to nodes and set variables to sets of nodes, in other words, for all $X, A \in V$ we have that $\alpha(X) \in \mathsf{nodes}_\tau$ and $\alpha(A) \in 2^{\mathsf{nodes}_\tau}$.

The language of Monadic Second-Order Dominance Logic is closely related to the Weak Monadic Second-Order Logic of the complete binary tree (Thatcher and Wright, 1967; Doner, 1970). This was first noticed by Backofen *et al.* (1995). The models of Monadic Second-Order Dominance Logic are ground terms while the only model of the Weak Monadic Second-Order Logic is the infinite binary tree. The latter is simpler in that all its nodes have first and second successors (children). This allows us to found the Weak Monadic Second-Order Logic on the two successor functions while Monadic Second-Order Dominance Logic must rely on the children-labeling relation.

Still, one can encode all ground terms in the infinite binary tree and thereby encode Monadic Second-Order Dominance Logic into Weak Monadic Second-Order Logic. This was used in Koller *et al.* (1998) to encode the first-order theory of dominance constraints into Weak Monadic Second-Order Logic. The current section generalises and complements this earlier result.

Proposition 7.20 *Every tree-regular constraint ξ is equivalent to some formula ψ in the monadic second-order dominance logic over the same signature.*

Proof: Let $\mathcal{A}$ be a tree automaton and X a node variable. We show how to express $\mathsf{tree}(X) \in \mathcal{L}(\mathcal{A})$ through an equivalent formula ψ of Monadic Second-Order Dominance Logic. Let Q be the set of states of $\mathcal{A}$ and Q_{fin} the set of its final states. We consider all states $q \in Q$ as second-order variables, whose set value contains all those nodes Y such that $\mathsf{tree}(Y)$ has a run into state q in $\mathcal{A}$. We then require that the value of $\mathsf{tree}(X)$ has a run into a final state, i.e. that $\mathsf{tree}(X) \in q$ for some final state $q \in Q_{\mathsf{fin}}$.

$$\psi = \exists Q.(\bigvee_{q \in Q_{\mathsf{fin}}} X \in q \ \wedge \bigwedge_{q \in Q} \forall Y.\ (Y \in q \leftrightarrow \mathsf{step}_{\mathcal{A}}(Y, q)))$$

where $\mathsf{step}_{\mathcal{A}}(Y, q)$ means that there is a single automaton step proving that the value of $\mathsf{tree}(Y)$ has a run into q.

$$\mathsf{step}_{\mathcal{A}}(Y, q) \quad = \quad \bigvee_{f(q_1,\ldots,q_n) \to q \in \mathcal{A}} \quad \begin{array}{l} \exists Y_1 \ldots \exists Y_n.\ (Y{:}f(Y_1, \ldots, Y_n) \wedge \\ Y_1 \in q_1 \ \wedge \ \ldots \ \wedge \ Y_n \in q_n) \end{array}$$

Note that all states of $\mathcal{A}$ may belong to the set of free set variables of formula $\mathsf{step}_{\mathcal{A}}(Y, q)$ so that the values of all sets $q \in Q$ are defined by mutual recursion. ■

The converse of the above proposition is wrong. For instance, one cannot express $X \lhd^* Y$ equivalently by means of tree-regular constraints since satisfiable tree-regular constraints can always be satisfied such that all variables denote disjoint nodes. Nevertheless, a weakened converse modulo satisfaction equivalence still holds and will allow us to prove Theorem 7.27, which states that *every tree-regular constraint ξ is satisfaction equivalent to some formula ψ of the monadic second-order dominance logic over the same signature, and vice versa.*

This theorem establishes a bidirectional relationship between dominance logics and tree automata. The one direction is already proved (Proposition 7.20). The proof of the other direction relies on standard encoding techniques known from Weak Monadic Second-Order Logic. For every formula of Monadic Second-Order Dominance Logic, we have to construct a tree automaton that recognises all its solutions converted into some tree format (Corollary 7.26 below). This format is obtained by encoding information about the values of node variables into extended node labels of some extended signature.

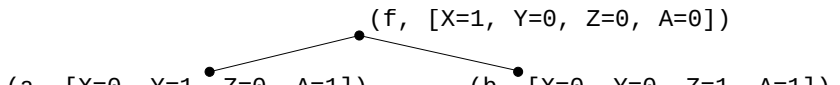
7.6.3 Extending Node Labels

The trick is to encode a solution pair τ, α into a single tree which looks like the tree τ except that it contains all information about the variable assignment α in extended node labels. Given a formula ψ of the Monadic Second-Order Dominance Logic, one can then recognise all encoded solutions of ψ by a tree automaton.

We first illustrate the encoding of pairs τ, α with an example.

Example 7.21 Let τ be the tree $f(a, b)$ with nodes $\mathsf{nodes}_\tau = \{\epsilon, 1, 2\}$ and α be the variable assignment given by $\alpha(X) = \epsilon$, $\alpha(Y) = 1$, $\alpha(Z) = 2$, and

$\alpha(A) = \{1, 2\}$. We then encode τ, α by means of the following tree with extended node labels:



In the general case, we encode pairs $\tau, \alpha : V \to \Sigma$ into trees over the signature of extended labels Σ_V:

$$\Sigma_V = \{(f, ch) \mid f \in \Sigma, ch : V \to \{0, 1\}\}$$

The second components of extended labels (f, ch) are finite characteristic functions with domain V. The arity of a label (f, ch) in Σ_V is equal to the arity of f. We identify the labels f and (f, ch) where ch is the constant 0-valued function. Through this identification, we turn Σ into a subset of Σ_V. As in the preceding example, we use the record notation $[Z_1{=}B_1, \ldots, Z_n{=}B_n]$ to represent the finite characteristic function $ch : \{Z_1, \ldots, Z_n\} \to \{0, 1\}$ with $ch(Z_1) = B_1$, ..., $ch(Z_n) = B_n$.

Definition 7.22 *We encode a pair $\tau, \alpha : V \to \mathsf{nodes}_\tau$ through the α-extension $\mathsf{ext}_\alpha(\tau)$. The trees $\mathsf{ext}_\alpha(\tau)$ and τ have the same set of nodes. A node π of $\mathsf{ext}_\alpha(\tau)$ is given the label (f, ch) if and only if the same node of τ is given the label f and for all $X, A \in V$:*

$$\pi = \alpha(X) \text{ iff } ch(X) = 1 \qquad \text{and} \qquad \pi \in \alpha(A) \text{ iff } ch(A) = 1$$

Example 7.23 We now illustrate the encoding of the non-intervenance formula which is crucial for our elimination of lambda binding constraints (see Section 7.5). We present a tree automaton which accepts trees where no f-labeled node intervenes properly between X and Y:

$$\neg\, \exists Z.\ (X \lhd^+ Z \ \wedge\ Z \lhd^+ Y \ \wedge\ \exists Z_1 \exists Z_2.\ Z{:}f(Z_1, Z_2)\ )$$

Since automata are closed under complementation, it is sufficient to construct an automaton for positive intervenance. The signature is Σ_V where $V = \{X, Y\}$. The only acceptance state of this automaton is $q_{\mathsf{above}(X)}$. For all finitely many labels $g \in \Sigma$ we have the following rules:

$$\begin{array}{lcll}
(g, [X{=}0, Y{=}0])(q_{\mathsf{below}(Y)}, \ldots, q_{\mathsf{below}(Y)}) & \to & q_{\mathsf{below}(Y)} & \\
(g, [X{=}0, Y{=}1])(q_{\mathsf{below}(Y)}, \ldots, q_{\mathsf{below}(Y)}) & \to & q_{\mathsf{above}(Y)} & \\
(g, [X{=}0, Y{=}0])(\ldots, q_{\mathsf{above}(Y)}, \ldots) & \to & q_{\mathsf{above}(Y)} & \text{if } f \neq g \\
(f, [X{=}0, Y{=}0])(\ldots, q_{\mathsf{above}(Y)}, \ldots) & \to & q_{\mathsf{above}_f} & \\
(g, [X{=}0, Y{=}0])(\ldots, q_{\mathsf{above}_f}, \ldots) & \to & q_{\mathsf{above}_f} & \\
(g, [X{=}1, Y{=}0])(\ldots, q_{\mathsf{above}_f}, \ldots) & \to & q_{\mathsf{above}(X)} & \\
(g, [X{=}0, Y{=}0])(\ldots, q_{\mathsf{above}(X)}, \ldots) & \to & q_{\mathsf{above}(X)} &
\end{array}$$

The state $q_{\mathsf{below}(Y)}$ recognises all trees where $Y{=}0$ for all nodes. The state $q_{\mathsf{above}(Y)}$ recognises trees containing a node where $Y{=}1$. The state q_{above_f} recognises trees which contain a proper f-labeled ancestor of some node with $Y{=}1$. Finally, $q_{\mathsf{above}(X)}$ accepts all trees where $X{=}1$ occurs properly above an f-ancestor of some node where $Y{=}1$.

We also need to check that $X{=}1$ and $Y{=}1$ are seen at most once in a node label. This can be done by intersection with another tree automaton. ■

7.6.4 Constructing Tree Automata

We now construct tree automata for general formulae of Monadic Second-Order Dominance Logic. The following lemma will be useful.

Lemma 7.24 *If* $\mathsf{ext}_{\alpha_1}(\tau_1) = \mathsf{ext}_{\alpha_2}(\tau_2)$ *then* $\alpha_1 = \alpha_2$ *and* $\tau_1 = \tau_2$.

Proof: We first notice that being $\mathsf{ext}_{\alpha_1}(\tau_1) = \mathsf{ext}_{\alpha_2}(\tau_2)$, we can state that $\mathsf{nodes}_{\mathsf{ext}_{\alpha_1}(\tau_1)} = \mathsf{nodes}_{\mathsf{ext}_{\alpha_2}(\tau_2)}$, and that for all node $\pi \in \mathsf{nodes}_{\mathsf{ext}_{\alpha_1}(\tau_1)}$, we have that $\mathsf{ext}_{\alpha_1}(\tau_1)(\pi) = \mathsf{ext}_{\alpha_2}(\tau_2)(\pi)$. Then, it is easy to see (by structural induction on the trees) that $\tau_1 = \tau_2$ and $\alpha_1 = \alpha_2$ hold.

■

Proposition 7.25 *For all second-order dominance formulae* ψ *and finite sets* V *of variables, there exists a tree automaton* $\mathcal{A}$ *over the signature* Σ_V *which accepts those trees over* Σ_V *that encode tree-assignment-pairs* $\tau, \alpha_{|V}$ *such that* $\tau, \alpha \models \psi$*:*

$$\mathcal{L}(\mathcal{A}) = \{\mathsf{ext}_{\alpha_{|V}}(\tau) \mid \tau, \alpha \models \psi\}$$

Proof: We can assume without loss of generality that $Var(\psi) \subseteq V$. Otherwise, we can apply the proposition to $\psi' = \exists\, Var(\psi) - V.\ \psi$ which satisfies $Var(\psi') = V$ since $Var(\psi) - (Var(\psi) - V) \subseteq V$. The automaton $\mathcal{A}$ for ψ' recognises the required language $\mathcal{L}(\mathcal{A}) = \{\mathsf{ext}_{\alpha_{|V}}(\tau) \mid \tau, \alpha \models \psi'\} = \{\mathsf{ext}_{\alpha_{|V}}(\tau) \mid \tau, \alpha \models \psi\}$. Let a V-extension of a tree τ be some α-extension of τ with $\alpha : V \to \mathsf{domain}_\tau$. We next construct an automaton $\mathcal{A}_{\mathsf{ext}_V}$ which only accepts those trees over Σ_V that are V-extensions of some tree in Σ. This automaton has to check for every first-order variable $X \in V$ and acceptable tree τ that there exists exactly one node in τ whose characteristic function maps X to 1. The automaton $\mathcal{A}_\emptyset$ accepts all trees of Σ. For the general case, let $V_1 \subseteq V$ where V_1 is the set of first-order variables we define $\mathcal{A}_{\mathsf{ext}_V} = \bigcap_{X \in V_1} \mathcal{A}_{\mathsf{ext}_{\{X\}}}$. It only remains to define the automata $\mathcal{A}_{\mathsf{ext}_{\{X\}}}$ for singleton sets $\{X\}$. Let $V = \{Z_1, \ldots, Z_n\}$, for any constant symbol $f \in \Sigma$, the rules are:

$$\begin{array}{lcl}
(f, [\ldots, X{=}0, \ldots])(q_{\mathsf{none}}, \ldots, q_{\mathsf{none}}) & \to & q_{\mathsf{none}} \\
(f, [\ldots, X{=}1, \ldots])(q_{\mathsf{none}}, \ldots, q_{\mathsf{none}}) & \to & q_{\mathsf{once}} \\
(f, [\ldots, X{=}0, \ldots])(q_{\mathsf{none}}, \ldots, q_{\mathsf{none}}, q_{\mathsf{once}}, q_{\mathsf{none}}, \ldots, q_{\mathsf{none}}) & \to & q_{\mathsf{once}}
\end{array}$$

The automaton counts how often $X{=}1$ was seen. It starts with q_{none} and increments to q_{once} when the first occurrence comes, and rejects starting from the second occurrence. The only final state of $\mathcal{A}_{\mathsf{ext}_{\{X\}}}$ is q_{once}.

We next construct automata $\mathcal{A}_\psi$ over the signature Σ_V that check the validity of ψ. The proposition is then always satisfied with $\mathcal{A} = \mathcal{A}_\psi \cap \mathcal{A}_{\mathsf{ext}_V}$. The construction is by structural induction on formulae ψ and the rules are defined for any of the finitely many symbols f of Σ.

1. Case $\psi = X{=}Y$. We construct the following automaton that checks whether $X{=}1$ and $Y{=}1$ occur simultaneously at some node. The only final state q_{equal} of $\mathcal{A}_\psi$ indicates this case.

$$\begin{array}{lcl}(f,[\ldots,X{=}0,\ldots,Y{=}0,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{all}}\\ (f,[\ldots,X{=}1,\ldots,Y{=}1,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{equal}}\\ (f,[\ldots,X{=}0,\ldots,Y{=}0,\ldots])(\ldots,q_{\mathsf{equal}},\ldots) & \to & q_{\mathsf{equal}}\end{array}$$

2. Case $\psi = X \lhd^+ Y$. We construct the following automaton that checks whether $Y{=}1$ is seen properly below $X{=}1$. The final state of $\mathcal{A}_\psi$ is $q_{\mathsf{above}(X)}$.

$$\begin{array}{lcl}(f,[\ldots,X{=}0,\ldots,Y{=}0,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{all}}\\ (f,[\ldots,Y{=}1,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{above(Y)}}\\ (f,[\ldots,X{=}0,\ldots])(\ldots,q_{\mathsf{above(Y)}},\ldots) & \to & q_{\mathsf{above(Y)}}\\ (f,[\ldots,X{=}1,\ldots])(\ldots,q_{\mathsf{above(Y)}},\ldots) & \to & q_{\mathsf{above}(X)}\\ (f,[\ldots])(\ldots,q_{\mathsf{above}(X)},\ldots) & \to & q_{\mathsf{above}(X)}\end{array}$$

3. Case $\psi = X \lhd^* Y$. Tree automata are closed under union, so we define $\mathcal{A}_\psi{=}\mathcal{A}_{X\lhd^+Y} \cup A_{X=Y}$.

4. Case $\psi = \exists X.\psi'$. We can assume without loss of generality that $X \notin V$. Let $\mathcal{A}_{\psi'}$ be the automaton for ψ' but over the extended signature $\Sigma_{V\uplus\{X\}}$. We call a tree τ over Σ_V an X-projection of a tree τ' over $\Sigma_{V\uplus\{X\}}$ if τ is obtained from τ' by restricting all characteristic functions in node labels of τ' to V. We can easily define the automaton $\mathcal{A}_\psi$ such that it accepts all X-projections of trees in $\mathcal{L}(\mathcal{A}_{\psi'})$.

5. Case $\psi = X{:}g(X_1,\ldots,X_n)$. We construct the automaton that checks that the node of X has a label g and is applied over the nodes that are the values of $X_1,\ldots,X_n$. This last condition is ensured using states $q_{\mathsf{X_1}},\ldots,q_{\mathsf{X_n}}$.

$$\begin{array}{lcl}(f,[\ldots,X=0,\ldots,X_i{=}0,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{all}}\\ (f,[\ldots,X=0,\ldots,X_i{=}1,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{X_i}}\\ (g,[\ldots,X{=}1,\ldots])(q_{\mathsf{X_1}},\ldots,q_{\mathsf{X_n}}) & \to & q_{\mathsf{label-ok}}\\ (f,[\ldots])(\ldots,q_{\mathsf{label-ok}},\ldots) & \to & q_{\mathsf{label-ok}}\end{array}$$

   The only final state of $\mathcal{A}_\psi$ is $q_{\mathsf{label-ok}}$.

6. Case $\psi = X \in A$. We construct the following automaton that checks whether the node value of X belongs to the set value of A.

$$\begin{array}{lcl}(f,[\ldots,X{=}0,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{all}}\\ (f,[\ldots,X{=}1,\ldots,A{=}1,\ldots])(q_{\mathsf{all}},\ldots,q_{\mathsf{all}}) & \to & q_{\mathsf{inside}}\\ (f,[\ldots])(\ldots,q_{\mathsf{inside}},\ldots) & \to & q_{\mathsf{inside}}\end{array}$$

The only final state of $\mathcal{A}_\psi$ is q_{inside}.

7. Case $\psi = \psi_1 \ \wedge \ \psi_2$. Tree automata are closed under intersection, so we can set $\mathcal{A}_\psi = \mathcal{A}_{\psi_1} \cap \mathcal{A}_{\psi_2}$.

8. Case $\psi = \neg\psi'$. Tree automata are closed under complementation. We define $\mathcal{A}_\psi = \mathcal{A}_{\mathsf{ext}_V} \setminus \mathcal{A}_{\psi'}$

9. Case $\psi = \exists A.\psi'$. The construction is as in the first-order case.

■

We can now prove that tree-regular constraints can indeed express second-order monadic dominance formulae modulo satisfaction equivalence (but not equivalence).

Corollary 7.26 *For every formula ψ of the Monadic Second-Order Dominance Logic there exists a satisfaction equivalent tree-regular constraint ξ over the same signature.*

Proof: We can assume without loss of generality that ψ is closed. Let $V = \emptyset$ and let $\mathcal{A}$ be a tree automaton according to Proposition 7.25 that satisfies: $\mathcal{L}(\mathcal{A}) = \{\tau \mid \tau \models \psi\}$. We don't need any variable assignment to interpret ψ since ψ is closed. Let X, Y be fresh variables. The following conditional equivalence is valid in all trees:

$$\forall Y.\ X \lhd^* Y \ \rightarrow \ (\psi \ \leftrightarrow \ \mathsf{tree}(X) \in \mathcal{L}(\mathcal{A}))$$

If a tree τ, α satisfies the assumption $\forall Y.\ X \lhd^* Y$ then $\alpha(X)$ must be the root of τ. In this case, $\mathsf{tree}(\alpha(X)) \in \mathcal{L}(\mathcal{A})$ is equal to $\tau \in \mathcal{L}(\mathcal{A})$ which is $\tau \models \psi$. Next note that the assumption $\forall Y.\ X \lhd^* Y$ can be joint, while solving $\mathsf{tree}(X) \in \mathcal{L}(\mathcal{A})$, with ψ. Thus, ψ is satisfaction equivalent to the tree-regular constraint $\mathsf{tree}(X) \in \mathcal{L}(\mathcal{A})$. ■

Now, the previously announced theorem follows:

Theorem 7.27 *Every tree-regular constraint ξ is satisfaction equivalent to some formula ψ of the monadic second-order dominance logic over the same signature, and vice versa.*

Proof: The Theorem holds as a direct consequence of Proposition 7.20 and Corollary 7.26. ■

7.7 Extensions of Parallelism Constraints

Our next goal is to lift Theorem 7.27 to extensions of parallelism constraints with tree-regular constraints and with Second-Order Dominance formulae. This

means that we want to reduce satisfiability of a conjunction $\phi \wedge \xi$ to the satisfiability of some conjunctions $\phi' \wedge \psi$ and vice versa. This is what Theorem 7.30 proves below.

The one direction still follows immediately from Proposition 7.20 (which is modulo equivalence). But we cannot directly apply Theorem 7.27 to prove the converse. This weakness is due to the notion of satisfaction equivalence used in Corollary 7.26 in contrast to ordinary equivalence. We use the following proposition:

Proposition 7.28 *Every conjunction $\phi \wedge \psi$ of a parallelism constraint with a formula of Monadic Second-Order Dominance Logic is satisfaction equivalent to some formula $\bigvee_{i=1}^{k} \phi_i \wedge \xi_i$ with parallelism with tree-regular constraints.*

The proof will take up the rest of this section. The idea is to describe a solution τ, α of $\phi \wedge \psi$ by talking about a large tree that contains τ and $\mathsf{ext}_\alpha(\tau)$ simultaneously. The translation keeps the parallelism constraint ϕ in order to describe τ while it expresses the dominance formula ψ through a tree-regular constraint about $\mathsf{ext}_\alpha(\tau)$. The intended relationship between τ and $\mathsf{ext}_\alpha(\tau)$ is enforced by additional parallelism constraints (Lemma 7.29).

We first introduce formulae $\mathsf{ext}_V(X, Y)$ for finite sets V of variables. The free variables of $\mathsf{ext}_V(X, Y)$ are those in $V \cup \{X, Y\}$. A pair τ', α satisfies $\mathsf{ext}_V(X, Y)$ if the tree below $\alpha(Y)$ in τ' is the $\alpha_{|V}$ extension of the tree below $\alpha(X)$ in τ', in other words,

$$\begin{aligned} &\tau', \alpha \models \mathsf{ext}_V(X, Y) \\ &\quad \text{iff} \\ &\tau'.\alpha(Y) = \mathsf{ext}_{\alpha_{|V}}(\tau'.\alpha(X)) \end{aligned}$$

see Figure 7.12.

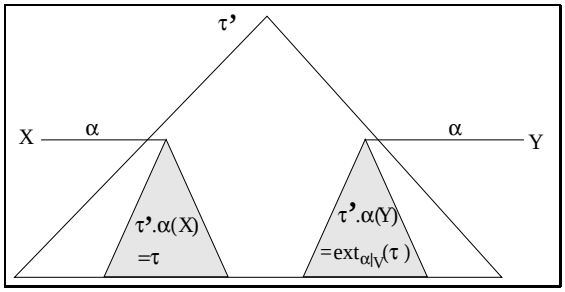


Figure 7.12: The tree τ' containing τ and its corresponding tree with extended labels.

Every solution τ', α of $\mathsf{ext}_V(X, Y)$ indeed contains occurrences of $\tau = \tau'.\alpha(X)$ and its extension $\mathsf{ext}_{\alpha_{|V}}(\tau) = \tau'.\alpha(Y)$ simultaneously. Note that $\alpha_{|V}$ must map to nodes of τ by definition of extensions, while the unrestricted assignment α may map to arbitrary nodes of τ'.

Recall that we have identified the labels f and (f, ch) where ch is the constant 0-valued function, and that this identification makes Σ be a subset of Σ_V. This has an important consequence: if V contains only first-order variables then the trees τ and $\mathsf{ext}_\alpha(\tau)$ have the same structure with finitely many exceptions: for all $Z \in V$ the node $\alpha(Z)$ below $\alpha(X)$ and its correspondent below $\alpha(Y)$ carry distinct labels. The number of exceptions is bounded by the size of V. This property would fail if we permitted second-order variables in V: a single second-order variable $A \in V$ where $\alpha(A)$ contains all nodes of τ makes all corresponding node labels of τ and $\mathsf{ext}_{\alpha_{|V}}(\tau)$ distinct.

Lemma 7.29 *Let V be a set of first-order variables. Every formula $\mathsf{ext}_V(X, Y)$ is equivalent to some positive existentially quantified formula $\bigvee_{l=1}^{k} \exists \vec{Z}^l \phi_l$.*

Proof: We construct a formula η of the above form by induction on the size of V. If $V = \emptyset$ then we set $\eta = X/{\sim}Y/$. Otherwise, we guess node labels for all variables in V and all relationships between them: properly above, properly below the i-th children, equal, or disjoint. These are $O(|V|^2 * M * |\Sigma|)$ guesses where M is the maximal arity of function symbols in Σ. This measure bounds k. We then translate deterministically for all possible choices. Let $X_1, \ldots, X_n$ be some maximal set of top-most situated variables that take distinct values (according to our guesses). We define:

$$\eta = \exists Y_1, \ldots, \exists Y_n.\ X/X_1, \ldots, X_n \sim Y/Y_1, \ldots, Y_n \ \wedge \ \bigwedge_{i=1}^{n} \eta_i$$

The formulae η_i are still to be defined. Let $ch_i : V \to \{0, 1\}$ be the function that maps all variables that take the same value as X_i to 1 and all others to 0 (according to our guesses). Let f_i be the guessed node label of arity n_i for the variable X_i and V_i^j be the set of variables which are below the j-th child of X_i. We then define:

$$\begin{array}{rcl} \eta_i & = & \exists X_i^1 \ldots \exists X_i^{n_i}.\ X_i{:}(f_i(X_i^1, \ldots, X_i^{n_i}) \wedge \\ & & \exists Y_i^1 \ldots \exists Y_i^{n_i}.\ Y_i{:}(f_i, ch_i)(Y_i^1, \ldots, Y_i^{n_i}) \ \wedge \ \bigwedge_{j=1}^{n_i} \mathsf{ext}_{V_i^j}(X_i^j, Y_i^j) \end{array}$$

■

Proof: [of **Proposition 7.28**] We consider a formula $\phi \wedge \psi$ where ψ does not contain free second-order variables without loss of generality. Otherwise, we can produce a satisfaction equivalent formula of the same form by existential quantification.

Let X be a fresh variable and $V = Var(\phi \wedge \psi) \cup \{X\}$ a set of first-order variables. We next define a formula η that we will prove satisfaction equivalent to $\phi \wedge \psi$:

$$\eta \ = \ \phi \ \wedge \ \exists Y.\ \mathsf{ext}_V(X, Y) \ \wedge \ \mathsf{tree}(Y) \in \{\mathsf{ext}_{\alpha_{|V}}(\tau) \mid \tau, \alpha \models \psi\}$$

First note that η can be rewritten into a satisfaction equivalent disjunction of the required form $\bigvee_{i=1}^{k} \phi_i \ \wedge \ \xi_i$. We can express $\mathsf{ext}_V(X, Y)$ by a disjunction

of parallelism constraints up to satisfaction equivalence (Lemma 7.29) and state the membership condition on $\mathsf{tree}(Y)$ by a tree-regular constraint (Proposition 7.25).

It remains to show that η is satisfaction equivalent to $\phi \wedge \psi$. Forward direction: suppose $\tau', \alpha' \models \eta$. We show that $\tau'.\alpha'(X), \alpha'_{|V} \models \phi \wedge \psi$. First note that $\alpha'_{|V}$ maps to nodes below $\alpha'(X)$ since $\tau', \alpha' \models \mathsf{ext}_V(X, Y)$. Second note that $\alpha'_{|V}$ can interpret all variables of $\phi \wedge \psi$ by definition of V. Third, we show that $\tau'.\alpha'(X), \alpha'_{|V}$ solves ϕ: By assumption, $\tau', \alpha' \models \eta$ and thus $\tau', \alpha'_{|V} \models \phi$. But since ϕ contains parallelism literals only, we can restrict this solution to the subtree of τ' to which $\alpha'_{|V}$ maps; thus: $\tau'.\alpha'(X), \alpha'_{|V} \models \phi$. Fourth, we show that $\tau'.\alpha'(X), \alpha'_{|V}$ solves ψ. Since τ', α' satisfies the membership restriction on $\mathsf{tree}(Y)$ there exists a solution $\tau, \alpha \models \psi$ such that:

$$\tau'.\alpha'(Y) = \mathsf{ext}_{\alpha_{|V}}(\tau)$$

Since $\tau', \alpha' \models \mathsf{ext}_V(X, Y)$ we also know $\tau'.\alpha'(Y) = \mathsf{ext}_{\alpha'_{|V}}(\tau'.\alpha'(X))$. The previous two equations combine into $\mathsf{ext}_{\alpha_{|V}}(\tau) = \mathsf{ext}_{\alpha'_{|V}}(\tau'.\alpha'(X))$ such that the Uniqueness Lemma 7.24 yields $\alpha'_{|V} = \alpha_{|V}$ and $\tau'.\alpha'(X) = \tau$. From $\tau, \alpha \models \psi$, we get $\tau, \alpha_{|V} \models \psi$, and hence, $\tau'.\alpha'(X), \alpha'_{|V} \models \psi$.

For the other direction, we assume that $\phi \wedge \psi$ is satisfiable and construct a solution of η. Let τ, α be a solution of $\phi \wedge \psi$. We define $\tau' = f(\tau, \mathsf{ext}_{\alpha_{|V}}(\tau), \ldots)$ where f is some function symbol of arity at least 2. (The children of τ starting from position 3 can be chosen arbitrarily.) Let π_1 be the first child of the root of τ'. It then holds that $\tau', \alpha[X \mapsto \pi_1] \models \eta$ whereby the existentially quantified variable Y can be mapped to the second child of τ'.

■

Theorem 7.30 *The satisfiability problems of Parallelism with Tree-Regular Constraints and of Parallelism Constraints plus Monadic Second-Order Dominance formulae are equal modulo non-deterministic polynomial time transformations.*

Proof: Proposition 7.20 and Proposition 7.28 entail this theorem[3]. It is easy to check that the guesses we have to make are polynomially bounded. ■

7.8 Equivalence Between Parallelism Constraints And Context Unification When Considering Tree-Regular Constraints

We know that Parallelism Constraints and Context Unification have the same expressiveness (Niehren and Koller, 2001). We now prove (in Theorem 7.37 below) that this result can be lifted when extending both languages with tree regular

[3]Note that the signatures are part of the input of both satisfiability problems, hence the satisfaction equivalent formulae need not be defined over the same signature.

constraints. The proof can be obtained by extending the proof in Niehren and Koller (2001), but we show both implications independently. We first translate Context Unification with tree-regular constraints to Parallelism with tree-regular constraints.

Definition 7.31 *Suppose without loss of generality that we are given a single equation and a single tree-regular constraint* $E = \langle t_1 \stackrel{?}{=} t_2,\ Y \in \mathcal{L}(\mathcal{A})\rangle$[4] *We define the translation of* E, $\lceil E \rceil$ *into a Parallelism with tree-regular constraint problem by means of the following process:*
We first introduce fresh node variables for all subterm positions in the equation $t_1 \stackrel{?}{=} t_2$. *We then collect parallelism, children-labeling, and membership literals in four steps.*

1. *We collect children-labeling literals for all subterms in* $t_1 \stackrel{?}{=} t_2$ *that have the form* $f(s_1, \ldots, s_n)$. *Let* X *be the node variable for the position of such a subterm and* $X_1, \ldots, X_n$ *the node variables for the positions of the subterms* $s_1, \ldots, s_n$. *We then add the children-labeling literal:*

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

2. *We collect parallelism literals for all context (and first-order) variables occurring in* $t_1 \stackrel{?}{=} t_2$. *So let* $F(s_1, \ldots, s_n)$ *be an occurrence of some context variable* F *in* $t_1 \stackrel{?}{=} t_2$, X *be the node variable of this occurrence and* $X_1, \ldots, X_n$ *the node variables of the subterms* $s_1, \ldots, s_n$. *Let* $F(s'_1, \ldots, s'_n)$ *be a second possibly equal occurrence of the same context variable* F *in* $t_1 \stackrel{?}{=} t_2$, X' *be the node variable of this occurrence and* $X'_1, \ldots, X'_n$ *the node variables of the subterms* $s'_1, \ldots, s'_n$. *We then add the parallelism literal:*

$$X/X_1, \ldots, X_n \sim X'/X'_1, \ldots, X'_n$$

   Notice that for first-order variables, the segment of the parallelism literal will not have holes.

3. *Suppose that* Y *(the variable of the tree-regular constraint) occurs in the equation* $t_1 \stackrel{?}{=} t_2$ *at some position with node variable* Y'. *We then add:*

$$\mathsf{tree}(Y') \in \mathcal{L}(\mathcal{A})$$

4. *We ensure that both sides of the equation* $t_1 \stackrel{?}{=} t_2$ *denote equal values. Let* X_1 *and* X_2 *be the node variables of the subterm positions of* t_1 *and* t_2 *(the roots). We then add the parallelism literal:*

$$X_1/ \sim X_2/$$

[4]Recall that when considering context equations, Y is a first-order variable, not a node variable. Notice also that in Definition 6.6 we consider tree-regular constraints over any term and here we consider constraints over a first-order variable. It is not difficult to see that both definitions are equivalent.

Example 7.32 For instance, for the context equation $F(f(X)) = f(F(a))$ with regular constraint $X \in \mathcal{L}(\mathcal{A})$, we first introduce node variables for all subterm positions and we make the following node variables association:

$$\begin{array}{ccccccccccc} X_0 & & X_1 & & X_2 & & Y_0 & & Y_1 & & Y_2 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ F(& & f(& & X &)) = & f(& & F(& & a &)) \end{array}$$

Then we translate the equation and the constraint as follows, where the lines contain the literals of the subsequent steps:

1. $X_1{:}f(X_2) \ \wedge \ Y_0{:}f(Y_1) \ \wedge \ Y_2{:}a \ \wedge$
2. $X_0/X_1 \sim Y_1/Y_2 \ \wedge$
3. $\mathsf{tree}(X_2) \in \mathcal{L}(\mathcal{A}) \ \wedge$
4. $X_0/ \sim Y_0/$

In step 2 of this example we have freely omitted parallelism literals between $X_2/ \ \sim X_2/$ and equal segment terms: $X_0/X_1 \sim X_0/X_1$ and $Y_1/Y_2 \sim Y_1/Y_2$. These last literals enforce dominance relations $X_0 \triangleleft^* X_1$ and $Y_1 \triangleleft^* Y_2$ that are entailed by $X_0/X_1 \sim Y_1/Y_2$ anyway. ■

For the sake of readability we identify terms with tree structures in the following proofs.

Lemma 7.33 *Any context unification equation with tree-regular constraints $E = \langle t_1 \stackrel{?}{=} t_2, Y \in \mathcal{L}(\mathcal{A}) \rangle$ is satisfiable if and only if its translation to parallelism with tree-regular constraints $\lceil E \rceil$ is.*

Proof:
Forward direction. Let σ be a minimal ground solution of equation $t_1 \stackrel{?}{=} t_2$, that satisfies $Y \in \mathcal{L}(\mathcal{A})$. Then, let $\tau = f(\sigma(t_1), \sigma(t_2))$, we will construct an α such that $\tau, \alpha \models \lceil E \rceil$. Let us proceed by structural induction on the equation terms. Let X be the topmost node variable of t_1, then $\alpha(X) = 1$ (and $\alpha(X') = 2$ being X' the topmost node variable of t_2), now we have two possibilities,

- if $t_1 = f(s_1, \ldots, s_n)$ then, let $X_1, \ldots, X_n$ respectively be the node variables for $s_1, \ldots, s_n$, then let $\alpha(X_i) = \alpha(X)i$ for $i \in \{1..n\}$.
- Otherwise, if $t_1 = F(s_1, \ldots, s_n)$ then, let $X_1, \ldots, X_n$ respectively be the node variables for $s_1, \ldots, s_n$, let $\sigma(F) = \lambda x_1 \ldots x_n.\ t$, and let p_i be the position of x_i in t for $i \in \{1..n\}$ (recall that $\sigma(F)$ is linear), then let $\alpha(X_i) = \alpha(X)p_i$ for $i \in \{1..n\}$.

Now, it is easy to see that effectively $\tau, \alpha \models \lceil E \rceil$. We proceed on the groups of literals of the four-steps translation.

1. The children-labeling literals are obviously satisfied by construction of α and τ.

2. For the introduced parallelism literals due to occurrences of context (or first-order) variable F, let $\sigma(F) = \lambda x_1 \ldots x_n.\ t$ and let $X/X_1, \ldots, X_n \sim X'/X'_1, \ldots, X'_n$ be one of the literals introduced due to the occurrences of F. It is easy to see by the construction of α that $\alpha(X/X_1, \ldots, X_n) = \alpha(X)/\alpha(X_1), \ldots, \alpha(X_n)$ and $\alpha(X'/X'_1, \ldots, X'_n) = \alpha(X')/\alpha(X'_1), \ldots, \alpha(X'_n)$ are both equal to t (up to bound variables), hence there exists a correspondence function between both segments, and $X/X_1, \ldots, X_n \sim X'/X'_1, \ldots, X'_n$ holds.

3. Let Y' be the node variable of Y then, as far as σ satisfies the tree-regular constraint, $\mathsf{tree}(Y') \in \mathcal{L}(\mathcal{A})$ will also hold.

4. This last step is obvious because $\sigma(t_1) = \sigma(t_2)$.

Backward direction. Let $\tau, \alpha \models \lceil E \rceil$. We will construct a substitution σ and then we will show that it solves E.

For all variable $F \in Var(t_1 \stackrel{?}{=} t_2)$ we proceed as follows. Let the parallelism literal $X/X_1, \ldots, X_n \sim X'/X'_1, \ldots, X'_n \in \lceil E \rceil$ be introduced in the second step of the translation due to occurrences of the variable F (hence, X and X' are node variables for occurrences of F in t_1 or t_2). We define $\sigma(F) = \lambda x_1, \ldots, x_n.\ \tau.\alpha(X)[\alpha(X_i)/x_i]$ for $i \in \{1..n\}$ being $n = arity(F)$. In other words, we obtain $\sigma(F)$ by replacing the "holes" of the segment $\alpha(X)/\alpha(X_1), \ldots, \alpha(X_n)$ by the corresponding bound variables. Notice also that it does not matter what parallelism literal introduced by the occurrences of F we take: all of them are related by parallelism literals and hence we would obtain the same substitution whichever we take hence

$$\lambda x_1, \ldots, x_n.\ \tau.\alpha(X)[\alpha(X_i)/x_i] = \lambda x_1, \ldots, x_n.\ \tau.\alpha(X')[\alpha(X'_i)/x_i] \quad i \in \{1..n\}$$

Let us see that effectively σ is a unifier of $t_1 \stackrel{?}{=} t_2$.

Let X_1 and X_2 be the node variables of t_1 and t_2 respectively, and $\alpha(X_1) = \pi_1$ and $\alpha(X_2) = \pi_2$. We will see that $\sigma(t_1) = \tau.\pi_1$ and that $\sigma(t_2) = \tau.\pi_2$. We proceed by structural induction on t (for t_1 and t_2).

We have two possibilities:

- Let $t = f(s_1, \ldots, s_n)$ for some n-ary function symbol f. Then the children-labeling literal $Y : f(Y_1, \ldots, Y_n)$ (where Y is the node variable of t, and Y_i of s_i for $i \in \{1..n\}$) is in $\lceil E \rceil$, and by induction hypothesis we get $\sigma(s_i) = \tau.\alpha(Y_i)$ for all $i \in \{1..n\}$, hence $\sigma(t) = \tau.\alpha(Y)$.

- Let $t = F(s_1, \ldots, s_n)$ for some n-ary variable F. Then the parallelism literal $Y/Y_1, \ldots, Y_n \sim Y/Y_1, \ldots, Y_n$ (where Y is the node variable of t) is in $\lceil E \rceil$. Notice that, by the construction of σ, we also know that $\sigma(F) = \lambda x_1, \ldots, x_n.\ \tau.\alpha(Y)[\alpha(Y_i)/x_i]$ for $i \in \{1..n\}$, hence $\sigma(F(s_1, \ldots, s_n)) = \tau.\alpha(Y)[\alpha(Y_i)/\sigma(s_i)]$, but by induction hypothesis we get that $\sigma(s_i) = \tau.\alpha(Y_i)$ for all $i \in \{1..n\}$, hence $\sigma(t) = \tau.\alpha(Y)$.

Finally, we know that $\tau.\pi_1 = \tau.\pi_2$ because the parallelism literal $X_1/ \sim X_2/$ introduced in the fourth step of the translation holds, hence we can conclude

that $\sigma(t_1) = \sigma(t_2)$. Moreover, being Y' the node variable of Y and knowing that $\mathsf{tree}(Y') \in \mathcal{L}(\mathcal{A})$ holds, we can also conclude that $\sigma(Y) \in \mathcal{L}(\mathcal{A})$.

■

We now give an inverse reduction that maps Parallelism with tree-regular constraints to Context Unification with tree-regular constraints. The difficulty of this reduction is raised again by the different views on trees. As we have already said: while Parallelism Constraints talk about nodes and segments, Context Unification deals with trees and contexts. So how can we speak about the nodes of a tree in Context Unification? The idea is that we speak about the context between the root of the tree and the node.

We now encode an extended parallelism constraint $\eta = \phi \ \wedge \ \xi$ with the set of node variable $V = Var(\eta)$ into a context unification with tree-regular constraints problem.

Definition 7.34 *Let $\eta = \phi \ \wedge \ \xi$ with the set of node variable $V = Var(\eta)$. We define a set of first-order and context variables as follows:*

- *let X_{all} be a first-order (tree) variable (that will denote a model of η).*
- *For every node variable $X \in V$ let F_X be a unary context variable(that will denote the context from the root of X_{all} to node X), and X' a first-order variable (that will denote the tree below X in X_{all}).*

We express the intended relationships between the introduced first-order and context variables through the context equations e_V:

$$e_V \quad = \quad \bigwedge_{X \in V} X_{all} \stackrel{?}{=} F_X(X')$$

Then we translate η by means of the translation $\lfloor \eta \rfloor$. The translation $\lfloor \ \rfloor$ of the literals of η is given in Figure 7.13.

Example 7.35 For instance, we translate the unsolvable parallelism constraint:

$$X{:}f(X_1, X_2) \ \wedge \ X_1/X_3, X_4 \sim X_2/X_3, X_4$$

into this conjunction of context unification equations, also unsolvable:

$$\begin{aligned}
&F_{X_1}(a) \stackrel{?}{=} F_X(f(a, X_2')) \ \wedge \ F_{X_2}(a) \stackrel{?}{=} F_X(f(X_1', a)) \wedge \\
&F_{X_1}(b) \stackrel{?}{=} F_X(f(b, X_2')) \ \wedge \ F_{X_2}(b) \stackrel{?}{=} F_X(f(X_1', b)) \wedge \\
&F_{X_3}(a) \stackrel{?}{=} F_{X_1}(F(a, X_4')) \ \wedge \ F_{X_3}(a) \stackrel{?}{=} F_{X_2}(F(a, X_4')) \\
&\qquad \wedge F_{X_4}(a) \stackrel{?}{=} F_{X_1}(F(X_3', a)) \ \wedge F_{X_4}(a) \stackrel{?}{=} F_{X_2}(F(X_3', a)) \wedge \\
&F_{X_3}(b) \stackrel{?}{=} F_{X_1}(F(b, X_4')) \ \wedge \ F_{X_3}(b) \stackrel{?}{=} F_{X_2}(F(b, X_4')) \\
&\qquad \wedge F_{X_4}(b) \stackrel{?}{=} F_{X_1}(F(X_3', b)) \ \wedge F_{X_4}(b) \stackrel{?}{=} F_{X_2}(F(X_3', b)) \wedge \\
&X_{all} \stackrel{?}{=} F_X(X') \wedge \ X_{all} \stackrel{?}{=} F_{X_1}(X_1') \ \wedge \ X_{all} \stackrel{?}{=} F_{X_2}(X_2') \wedge \\
&X_{all} \stackrel{?}{=} F_{X_3}(X_3') \ \wedge \ X_{all} \stackrel{?}{=} F_{X_4}(X_4')
\end{aligned}$$

■

$$
\begin{array}{rcl}
\lfloor X : f(X_1,\ldots,X_n)\rfloor & = & F_{X_1}(a) \stackrel{?}{=} F_X(f(a,X'_2,\ldots,X'_n)) \\
 & & \wedge \ldots \wedge \\
 & & F_{X_n}(a) \stackrel{?}{=} F_X(f(X'_1,\ldots,X'_{n-1},a)) \\
 & \wedge & F_{X_1}(b) \stackrel{?}{=} F_X(f(b,X'_2,\ldots,X'_n)) \\
 & & \wedge \ldots \wedge \\
 & & F_{X_n}(b) \stackrel{?}{=} F_X(f(X'_1,\ldots,X'_{n-1},b)) \\
\lfloor X : a\rfloor & = & X' \stackrel{?}{=} a \\
\lfloor X/X_1,\ldots,X_n \sim Y/Y_1,\ldots,Y_n\rfloor & = & F_{X_1}(a) \stackrel{?}{=} F_X(F(a,X'_2,\ldots,X'_n)) \wedge \\
 & & F_{Y_1}(a) \stackrel{?}{=} F_Y(F(a,Y'_2,\ldots,Y'_n)) \wedge \\
 & & \qquad \wedge \ \ldots \ \wedge \\
 & & F_{X_n}(a) \stackrel{?}{=} F_X(F(X'_1,X'_2,\ldots,a)) \wedge \\
 & & F_{Y_n}(a) \stackrel{?}{=} F_Y(F(Y'_1,Y'_2,\ldots,a)) \\
 & \wedge & F_{X_1}(b) \stackrel{?}{=} F_X(F(b,X'_2,\ldots,X'_n)) \wedge \\
 & & F_{Y_1}(b) \stackrel{?}{=} F_Y(F(b,Y'_2,\ldots,Y'_n)) \wedge \\
 & & \qquad \wedge \ \ldots \ \wedge \\
 & & F_{X_n}(b) \stackrel{?}{=} F_X(F(X'_1,X'_2,\ldots,b)) \wedge \\
 & & F_{Y_n}(b) \stackrel{?}{=} F_Y(F(Y'_1,Y'_2,\ldots,b)) \\
 & & \text{(being } F \text{ a fresh context variable)} \\
\lfloor \eta_1 \wedge \eta_2\rfloor & = & \lfloor \eta_1\rfloor \wedge \lfloor \eta_2\rfloor \\
\lfloor \mathsf{tree}(X) \in \mathcal{L}(\mathcal{A})\rfloor & = & X' \in \mathcal{L}(\mathcal{A})
\end{array}
$$

Figure 7.13: Reduction of Parallelism with Tree-Regular Constraints to Context Unification with Tree-Regular Constraints.

Lemma 7.36 *Any parallelism with tree-regular constraints formula $\phi \wedge \xi$ with variable set V is satisfiable if and only if the system of context unification equations with tree-regular constraints $e_V \ \wedge \ \lfloor \phi \ \wedge \ \xi \rfloor$ is.*

Proof: Forward direction. Let $\tau, \alpha \models \phi \ \wedge \ \xi$. We are going to construct a substitution σ from τ and α and show that σ solves the context unification with tree regular constraints problem $e_V \ \wedge \ \lfloor \phi \ \wedge \ \xi \rfloor$. We define the substitution for the three kinds of variables that we create in the translation:

- for the first-order variable X_{all} that denotes τ, let $\sigma(X_{all}) = \tau$.
- For each context variable F_X that denotes the context from the root of τ to node X, let $\sigma(F_X) = \lambda x.\ \tau[\alpha(X)/x]$.
- For each first-order variable X' that denotes the tree below X in τ, let $\sigma(X') = \tau.\alpha(X)$.
- And finally, for each n-ary context variable F introduced when translating $X/X_1,\ldots,X_n \sim Y/Y_1,\ldots,Y_n$, let $\sigma(F) = \lambda x_1,\ldots,x_n.\ \tau.\alpha(X)[\alpha(X_i)/x_i]$

for $i \in \{1..n\}$. Notice again that due to parallelism, we have that $\sigma(F)$ is also equal to $\lambda x_1, \ldots, x_n.\ \tau.\alpha(Y)[\alpha(Y_i)/x_i]$.

Now we have to prove that σ solves $e_V \ \wedge \ \lfloor \phi \ \wedge \ \xi \rfloor$. We will proceed showing that it effectively solves the groups of equations introduced in the translation.

- Consider the equations of e_V like $X_{all} \stackrel{?}{=} F_X(X')$ for any node variable X. By definition of σ we have that $\sigma(X_{all}) = \tau$ and that $\sigma(F_X(X')) = (\lambda x.\ \tau[\alpha(X)/x])\,(\tau.\alpha(X)) = \tau$.

- Consider now the equations introduced when translating the children-labeling literals $X : f(X_1, \ldots, X_n)$. There are two equations like $F_{X_i}(a) \stackrel{?}{=} F_X(f(X'_1, \ldots, a, \ldots X'_n))$ and $F_{X_i}(b) \stackrel{?}{=} F_X(f(X'_1, \ldots, b, \ldots X'_n))$ for $i \in \{1..n\}$, where a and b occur as the i'th child of f. Let us consider the first one (for the second one, the same proof applies). Now, by definition of σ we have that $\sigma(F_{X_i}(a)) = (\lambda x.\ \tau[\alpha(X_i)/x])\,(a) = \tau[\alpha(X_i)/a]$ and that:

$$\sigma(F_X(f(X'_1, \ldots, a, \ldots X'_n))) = \\ (\lambda x.\ \tau[\alpha(X)/x])\,(f(\tau.\alpha(X_1), \ldots, a, \ldots, \tau.\alpha(X_n)))$$

  but as far as $\tau, \alpha \models X : f(X_1, \ldots, X_n)$, we get that the previous term is equal to $\tau[\alpha(X_i)/a]$ as our equation requires.

- Consider now the equations introduced when translating the parallelism literals $X/X_1, \ldots, X_n \sim Y/Y_1, \ldots, Y_n$. There are two equations like $F_{X_i}(a) \stackrel{?}{=} F_X(F(X'_1, \ldots, a, \ldots X'_n))$ and $F_{X_i}(a) \stackrel{?}{=} F_Y(F(Y'_1, \ldots, a, \ldots Y'_n))$ for all $i \in \{1..n\}$, where a is the i'th child of F, and two more equations where b occurs instead of a. Let us consider the first one of the first pair (for the ones with b's, the same proof applies). Now, by definition of σ we have that $\sigma(F_{X_i}(a)) = \tau[\alpha(X_i)/a]$ and we also have that:

$$\sigma(F_X(F(X'_1, \ldots, a, \ldots X'_n))) = \\ (\lambda x.\ \tau[\alpha(X)/x])\,((\lambda \vec{x}.\ \tau.\alpha(X)[\alpha(X_i)/x_i])(\tau.\alpha(X_1), \ldots, a, \ldots, \tau.\alpha(X_n))) = \\ \tau[\alpha(X)/\,(\tau.\alpha(X)[\alpha(X_1)/\tau.\alpha(X_1), \ldots, \alpha(X_i)/a, \ldots, \alpha(X_n)/\tau.\alpha(X_n)])] = \\ \tau[\alpha(X_i)/a]$$

  as our equation requires. To solve the second equation (the one with the Y's variables), we just need to notice that both segments correspond. Hence, the same proof applies.

- The tree regular constraints $X' \in \mathcal{L}(\mathcal{A})$ are also satisfied, because $\sigma(X') = \tau.\alpha(X)$ and $\tau.\alpha(X) = \mathsf{tree}(X) \in \mathcal{L}(\mathcal{A})$.

We can conclude that σ solves $e_V \ \wedge \ \lfloor \phi \ \wedge \ \xi \rfloor$.

Backward direction. Let σ solve $e_V \ \wedge \ \lfloor \phi \ \wedge \ \xi \rfloor$. Then, let $\tau = \sigma(X_{all})$. We will define an α such that $\tau, \alpha \models \phi \ \wedge \ \xi$.

For all node variable $X \in Var(\phi \wedge \xi)$, let X' and F_X be their corresponding first-order and context variables in $e_V \ \wedge \ \lfloor \phi \ \wedge \ \xi \rfloor$. Let $\sigma(F_X) = \lambda x.t$ and let $t.\pi = x$ (recall that $\lambda x.t$ is linear and hence x occurs just once). Accordingly to

the fact that for all node variable X we have an equation $X_{all} \stackrel{?}{=} F_X(X')$, we define $\alpha(X) = \pi$. Now we have to prove that effectively $\tau, \alpha \models \phi \ \wedge \ \xi$, and we do so for the distinct kinds of literals of $\phi \ \wedge \ \xi$.

- For the mother-children literal $X : f(X_1, \ldots, X_n)$ we have to check that effectively for all $i \in \{1..n\}$, $\alpha(X_i) = \alpha(X)i$ and that $\tau(\alpha(X)) = f$.

  Let $\alpha(X) = \pi$ and let $\alpha(X_i) = \pi_i$ for all $i \in \{1..n\}$. First notice that for all $i \in \{1..n\}$ we have the equations:

  $$F_{X_i}(a) \stackrel{?}{=} F_X(f(X'_1, \ldots, a, \ldots, X'_n))$$
  $$F_{X_i}(b) \stackrel{?}{=} F_X(f(X'_1, \ldots, b, \ldots, X'_n))$$

  (being a and b the i'th child of f). The fact that we have two equations with distinct constants (a and b) as the argument of F_{X_i} (hence occurring at position π_i of $\sigma(F_{X_i}(a))$) and as the i'th argument of f (that being the argument of F_X occurs in position π, hence the constants occur at position πi of $\sigma(F_X(f(X'_1, \ldots, a, \ldots, X'_n)))$), allows us to conclude that the a's (and the b's) must occur at the same position in both terms, hence $\pi_i = \pi i$, even more, we can also conclude that $\sigma(F_{X_i}(a))(\pi) = f$.

  It remains to prove that $\tau(\alpha(X)) = f$, hence that $\tau(\pi) = f$. The equations: $F_{X_{all}} \stackrel{?}{=} F_{X_i}(X'_i)$, allow us to infer that $\tau = \sigma(F_{X_i}(X'_i))$ for all $i \in \{1..n\}$. Now, recall that $\sigma(F_{X_i}(a))(\pi) = f$, hence $\sigma(F_{X_i}(X'_i))(\pi) = \tau(\pi) = f$ as required.

- For the parallelism literals $X/X_1, \ldots, X_n \sim Y/Y_1, \ldots, Y_n$, we have to check that X dominates $X_1, \ldots, X_n$ and that these are disjoint, that Y dominates $Y_1, \ldots, Y_n$ and that these are disjoint also, and that a correspondence function exists between both segments.

  Let $\alpha(X) = \pi, \alpha(Y) = \pi'$ and let $\alpha(X_i) = \pi_i$ and $\alpha(Y'_i) = \pi'_i$ for all $i \in \{1..n\}$. Again, for all $i \in \{1..n\}$ we have the equations:

  $$F_{X_i}(a) \stackrel{?}{=} F_X(F(X'_1, \ldots, a, \ldots, X'_n))$$
  $$F_{X_i}(b) \stackrel{?}{=} F_X(F(X'_1, \ldots, b, \ldots, X'_n))$$

  (being a and b the i'th child of the context variable F). Let $\sigma(F) = \lambda x_1, \ldots, x_n.\ t$ and for all $i \in \{1..n\}$ let $t.\pi_i^F = x_i$. Then, following the same reasoning as before, we get that for all $i \in \{1..n\}, \alpha(X_i) = \pi\pi_i^F$. This proves that effectively $X/X_1, \ldots, X_n$ is a segment because the root dominates all the holes and these are pairwise disjoint since $\sigma(F)$ is linear and second-order typed, hence no bound variable can be applied over any bound variable. The same reasoning applies to the segment $Y/Y_1, \ldots, Y_n$.

  Now we have to prove that a correspondence function exists between $\pi/\pi\pi_1^F, \ldots \pi\pi_n^F$ and $\pi'/\pi'\pi_1^F, \ldots \pi'\pi_n^F$ in τ. We can use again the same reasoning as in the previous case and see that both segments are identical because they correspond to instances of the context defined by $\sigma(F) = \lambda x_1, \ldots, x_n.\ t$.

- For the tree-regular constraints, notice that $\sigma(X') = \tau.\alpha(X)$ satisfies the constraint, hence $\mathsf{tree}(X) = \tau.\alpha(X) \in \mathcal{L}(\mathcal{A})$ holds also.

We can conclude that $\tau, \alpha \models \phi \ \wedge \ \xi$ as required.

■

Theorem 7.37 *The extensions of parallelism constraints and context unification with tree-regular constraints are equivalent modulo polynomial time reductions.*

Proof: This theorem follows from Lemma 7.33 and Lemma 7.36. ■

7.8.1 Main Result

We have shown so far how to express Lambda Binding and Parallelism Constraints by means of Parallelism Constraints with First-Order Dominance, Theorem 7.19. Now, by considering Theorem 7.30 in Parallelism Constraints plus First-Order Dominance formulae and then considering Theorem 7.37 we get that Parallelism Constraints plus First-Order Dominance formulae can be expressed by Context Unification plus Tree-Regular Constraints equations. Then, the following theorem follows.

Theorem 7.38 *Satisfiability of Parallelism and Lambda Binding Constraints can be reduced in non-deterministic polynomial time to satisfiability of Context Unification with Tree-Regular Constraints.*

7.9 Limitations

An extension of the Constraint Language for Lambda Structures by Group Parallelism in order to deal with Beta Reduction Constraints is proposed by Bodirsky *et al.* (2001). The question is now whether *Group Parallelism* can be expressed in Context Unification with tree regular constraints. This is a relation between groups of segment terms $(S_1, \ldots, S_n) \sim (S'_1, \ldots, S'_n)$ that behave as a conjunction of parallelism literals $\wedge_{i=1}^{n} S_i \sim S'_i$ but such that hanging binders are defined with respect to groups of segments $(S_1, \ldots, S_n)$ and $(S'_1, \ldots, S'_n)$.

Unfortunately, we cannot extend the encodings of the present thesis. The problem is that Group Parallelism does not satisfy the non-intervenance property as stated for ordinary parallelism in Lemma 7.14. Indeed, it is not always possible to name variables consistently in the presence of group parallelism, so that corresponding binder of parallel groups are named alike. In other words, binding parallelism cannot be reduced to tree parallelism by naming binders. This is illustrated by the lambda structure in Figure 7.14 which satisfies the group parallelism constraint:

$$(X_1/X_2, X_4/X_5) \sim (X_2/X_3, X_3/X_4)$$

Even though the lam-node X_2 corresponds to X_1, X_2 intervenes between X_1 and its bound var-node X_6. We thus cannot name these corresponding nodes alike.

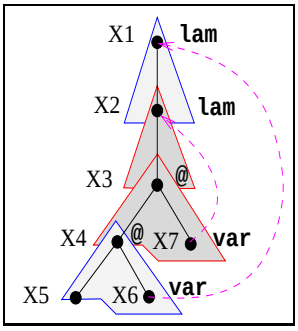


Figure 7.14: Group parallelism between $(X_1/X_2, X_4/X_5) \sim (X_2/X_3, X_3/X_4)$.

7.10 Summary

We have shown that the lambda-binding constraints of the Constraint Language for Lambda Structures can be expressed in Context Unification with tree-regular constraints. This result depends on the non-intervenance property of parallel lambda binding, by which binding parallelism can be reduced to tree parallelism.

The reduction of λ-binding and Parallelism constraints to Context Unification with tree-regular constraints requires several encoding steps. We have defined the Second-Order Dominance Logic and used it as an intermediate step of the translation. In fact, a sublanguage of this logic, the First-Order Dominance Logic, has allowed us to guarantee the soundness of the translation by enabling us to express the non-intervenance property.

We have also proved satisfaction equivalence between Parallelism with tree-regular constraints and Context Unification with tree-regular constraints.

We have also shown the main limitations of the techniques used in our translation when trying to extend it to *group parallelism*.

Chapter 8

Conclusion

In this last chapter we summarise the results of the thesis and we present the main lines of our future work.

8.1 Summary of the Thesis

In this thesis we have studied *Second-Order Unification*, mainly two of its variants: *Context Unification* and *Linear Second-Order Unification*. Both of them require unifiers where instantiations of second-order variables are *linear terms*, i.e. terms where all subterms of the form: $\lambda x.\ t$, satisfy that x occurs free in t once and just once.

While Linear Second-Order Unification equations are like Second-Order Unification equations, Context Unification equations do not allow the use of third (or higher)-order constants, nor of λ-abstractions. Even more, context (second-order) variables are sometimes restricted to be unary.

Figure 8.1 illustrates the main results achieved in this thesis. In the figure, *CU* stands for *Context Unification*, @ for *just one binary symbol*, λ for *Lambda Binding Constraints*, *LSOU* for *Linear Second-Order Unification*, n-ary for *n-ary variables*, *PC* for *Parallelism Constraints*, *RC* for *Regular Constraints*, *SDOM* for *Second-Order Dominance Formulae*, *TRC* for *Tree-Regular Constraints* and *WU* for *Word Unification*.

In the following we enumerate the main contributions of the thesis according to Figure 8.1.

1. In this work we have shown (see Theorem 3.19) that the arity of the variables does not affect to the decidability, therefore we consider Context Unification with n-ary context variables (for $n \geq 1$).

2. We have also shown (see Theorem 4.15) how the signature of Second-Order Unification and Context Unification can be simplified to only one binary function symbol and constants. This result illustrates the fact that the

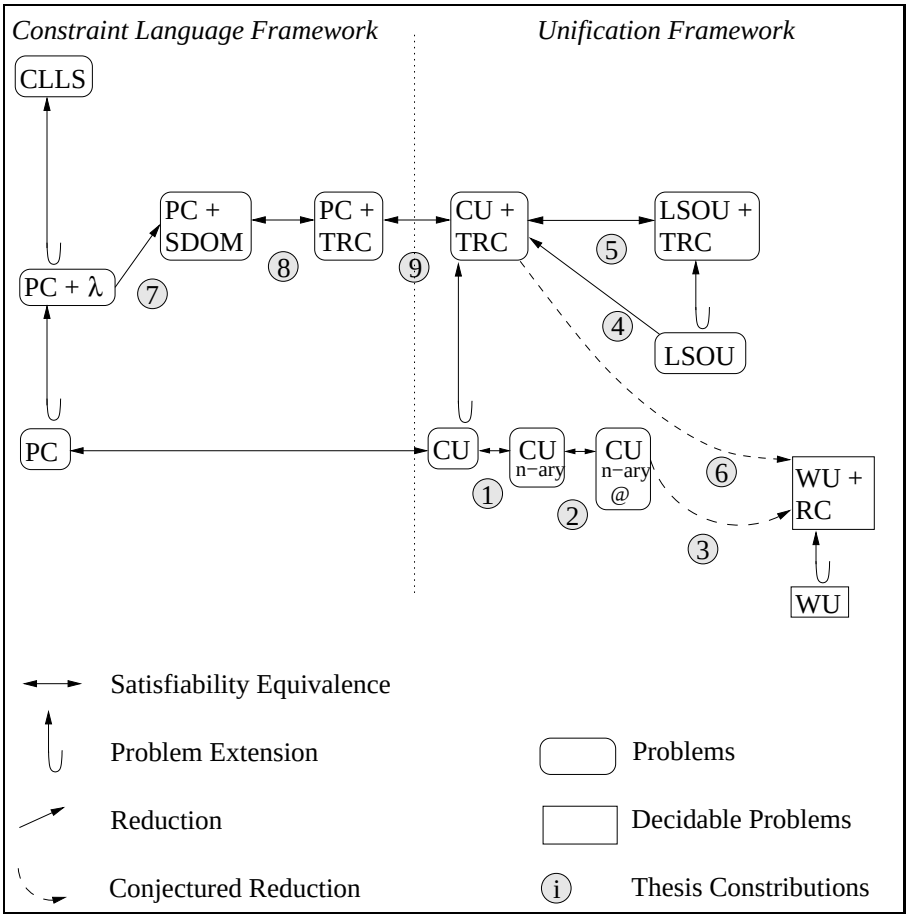


Figure 8.1: Studied problems and their relations.

importance of the signature, when considering decidability, lies in the difference between having at most unary constant symbols (Monadic Second-Order Unification) or having at least a binary symbol that allows branching. This result allowed us to concentrate on Context Unification with just one binary function symbol in this work.

3. The main result of the thesis is the reduction (see Theorem 5.21 and Theorem 5.27), under the assumption of Conjecture 5.22, of Context Unification to Word Unification with regular constraints, which is decidable. The conjecture claims the existence of a computable function Φ that allows us to ensure that: if the given problem $s \stackrel{?}{=} t$ is solvable then there exists a solution σ such that $\operatorname{rank} \sigma(s) \leq \phi(|s \stackrel{?}{=} t|)$.

   The *rank* measure is not a trivial measure as far as it does not imply

any bound on the size of the solutions. The reduction requires several encoding steps and has led us to define *traversal equations* and *rank and permutation-bound traversal equations*. Decidability of the latter has also been shown.

The remaining results of the thesis consist of the study of the relationship between Context Unification and Linear Second-Order Unification and between Context Unification and the Constraint Language for Lambda Structures. In order to be able to relate these problems we have extended Context Unification by means of tree-regular constraints (in the spirit of the extension of Word Unification with regular constraints). In both relationships, we have used tree-regular constraints to ensure the soundness of the reductions.

4,5. First, we have shown that Linear Second-Order Unification can be reduced to Context Unification with tree-regular constraints (see Corollary 6.14). In this reduction, the tree-regular constraints have been used to avoid variable capture and loss of linearity in the instantiations of context variables.

6. When we try to apply the reduction of Context Unification to Word Unification with regular constraints, to the broad case of Context Unification, we are forced to extended Conjecture 5.22 to Conjecture 6.23 to deal also with tree-regular constraints. We have also shown that satisfiability of *rank-bound tree-regular constraints* can be reduced to satisfiability of *regular constraints over traversal sequences*.

Then, we have focussed on the Constraint Language for Lambda Structures, a constraint formalism to talk about lambda structures that is currently used for semantic modelling of ambiguous sentences. We have shown that a part of this language can be reduced to Context Unification with tree-regular constraints.

7. Mainly we have reduced Parallelism and Lambda-binding constraints to Context Unification with tree-regular constraints. These tree-regular constraints ensure that the *non-intervenance property* of the satisfiable parallelism and lambda-binding constraints problems, is not violated in the translated context unification problem (see Theorem 7.19). This non-intervenance property has been introduced for the first time in this thesis.

8. In fact, this property is expressed in First-order Dominance formulae, a sublanguage of the Second-Order Dominance formulae, which are also proved to be satisfaction equivalent to tree-regular constraints, even when considered with parallelism constraints (see Theorem 7.30). This Second-Order Dominance Logic has been introduced for the first time in this thesis.

9. To complete the reduction we have shown that Parallelism with tree-regular constraints and Context Unification with tree-regular constraints are equivalent modulo polynomial time reductions (see Theorem 7.37).

8.2 Future Work

The main line of our future work is to prove the conjectures stated in this thesis: Conjecture 5.22 and Conjecture 6.23. Proving these conjectures would imply decidability of Context Unification, decidability of Linear Second-Order Unification, decidability of Parallelism and lambda binding constraints, and decidability of Parallelism and tree-regular constraints. Obviously this does not seem to be an easy task.

Some other questions that we consider interesting to investigate are:

1. The decidability of traversal systems. Notice that showing their decidability would directly imply decidability of Context Unification.

2. The definition of the precise fragment of Context Unification with practical interest for linguistical applications. Our belief is that a fragment where "overlapping" between distinct occurrences of the same variable is forbidden, is decidable and subsumes and generalises the "well nested fragment" of the Constraint Language for Lambda Structures defined in Erk and Niehren (2003).

3. The expressivity of Context Unification equations, as it has been done for Word Unification equations by Karhumäki *et al.* (1997).

4. The possible applications of Context Unification and of Context Matching in programming languages. In this sense, the work of Schmidt-Schauß and Stuber (2002) relating Context Matching with XML queries could be an starting point. Another possibility is to study the possible applications of Linear Higher-Order Matching in program transformations like de Moor and Sittampalam (2001) and Sittampalam and de Moor (2001), do with general Higher-Order Matching in the MAG system.

5. We have not been able to translate all the features of the Constraint Language for Lambda Structures into Context Unification with tree-regular constraints. We would like to investigate also if anaphoric binding constraints can also be expressed in Context Unification. We also leave as further work the study of how the group parallelism and beta reduction extensions are related with Context Unification.

6. We would also like to improve the signature simplification for Second-Order and for Context Unification to just one binary function symbol and just one 0-ary constant symbol.

Bibliography

Abadi, M., Cardelli, L., Curien, P.-L. and Lévy, J. (1998). Explicit Substitutions. *Journal of Functional Programming* **1**(4), 375–416.

Althaus, E., Duchier, D., Koller, A., Mehlhorn, K., Niehren, J. and Thiel, S. (2003). An Efficient Graph Algorithm for Dominance Constraints. *Journal of Algorithms* **1**(48), 194–219. Special Issue of SODA 2001.

Andrews, P. B. (1981). Theorem Proving through General Matings. *Journal of the ACM* **28**, 193–214.

Baader, F. and Siekmann, J. H. (1993). Unification Theory. In *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, Oxford, UK.

Baader, F. and Snyder, W. (2001). A Proof Theory for General Unification. In *Handbook of Automated Reasoning*, volume 1, chapter 8, 445–532. Elsevier Science Publishers and MIT Press.

Backofen, R., Rogers, J. and Vijay-Shanker, K. (1995). A First-order Axiomatization of the Theory of Finite Trees. *Journal of Logic, Language, and Information* **4**, 5–39.

Barendregt, H. P. (1984). *The Lambda Calculus - It's Syntax and Semantics*. North-Holland, Amsterdam.

Baxter, L. D. (1977). The Complexity of Unification. PhD Thesis, University of Waterloo.

Benzmüller, C. and Kohlhase, M. (1998a). Extensional Higher-Order Resolution. In *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, volume 1421 of *LNAI*, 56–71. Springer, Berlin.

Benzmüller, C. and Kohlhase, M. (1998b). LEO: A Higher-Order Theorem Prover. In *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, volume 1421 of *LNAI*, 139–143. Springer, Berlin.

Bjorner, N. and Muñoz, C. (2000). Absoulte Explicit Unification. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications (RTA'00)*, volume 1833 of *LNCS*, 31–46. Norwich, UK.

Bodirsky, M., Erk, K., Koller, A. and Niehren, J. (2001). Beta Reduction Constraints. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA'01)*, volume 2051 of *LNCS*, 31–46.

Cervesato, I. and Pfenning, F. (1997). Linear Higher-Order Pre-Unification. In *Proceedings of the 12th Annual Symposium on Logic in Computer Science (LICS'97)*, 422–433. IEEE Computer Society Press, Warsaw, Poland.

Church, A. (1940). A simple theory of types. *Journal of Symbolic Logic* **5**, 56–68.

Colmerauer, A. (1988). Final Specifications for PROLOG-III. Technical Report P1219(1106), ESPRIT.

Comon, H. (1991). Disunification: a Survey. In *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press.

Comon, H. (1992a). Completion of Rewrite Systems with Membership Constraints. In *International Colloquium on Automata, Languages and Programming (ICALP'92)*, volume 623 of *LNCS*. Vienna, Austria.

Comon, H. (1992b). On Unification of Terms with Integer Exponents. Technical Report 770, L. R. I., Univ. Paris-Sud.

Comon, H. (1998). Completion of Rewrite Systems with Membership Constraints. *Journal of Symbolic Computation* **25**(4), 397–453.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M. (1997). Tree Automata Techniques and Applications.

Comon, H. and Jurski, Y. (1997). Higher-Order Matching and Tree Automata. In *Proceedings of the 11th Workshop on Computer Science Logic (CSL'97)*, volume 1414 of *LNCS*, 157–176. Springer-Verlag.

Courcelle, B. (2000). The monadic second-order logic of graphs XIII: Graph drawings with edge crossings. *Computational Intelligence* **244**(1-2), 63–94.

Dalrymple, M., Shieber, S. M. and Pereira, F. C. N. (1991). Ellipsis and Higher-Order Unification. *Linguistics and Philosophy* **14**(4), 399–452.

Darlington, J. L. (1971). A Partial Mechanization of Second-order Logic. *Machine Intelligence* **6**, 91–100.

Darlington, J. L. (1973). Automatic Program Synthesis in Second-Order Logic. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI'73)*, 537–542. William Kaufmann, Standford, CA.

Degtyarev, A. and Voronkov, A. (1996). The undecidability of simultaneous rigid E-unification. *Theoretical Computer Science* **166**(1–2), 291–300. Note.

Diekert, V., Matiyasevich, Y. and Muscholl, A. (1997). Solving Trace Equations Using Lexicographical Normal Forms. In *International Colloquium on Automata, Languages and Programming (ICALP'97)*, 336–346. Bologna, Italy.

Doner, J. (1970). Tree Acceptors and Some of Their Applications. *Journal of Computer System Science* **4**, 406– 451. Received December 1967, Revised May 1970.

Dougherty, D. J. and Wierzbicki, T. (2002). A Decidable Variant of Higher-Order Matching. In *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA'02)*, volume 2378 of *LNCS*, 340–351. Springer-Verlag.

Dowek, G., Hardin, T. and Kirchner, C. (2000). Higher-Order Unification via Explicit Substitutions. *Information and Computation* **157**, 183–235.

Dowek, G. (1992). Third-Order Matching is Decidable. In *Proceedings of the 7th Annual Symposium on Logic in Computer Science (LICS'92)*, 2–10. IEEE Computer Society Press, Santa Cruz, California.

Dowek, G. (1993). A Unification algorithm For Second-Order Linear Terms. Manuscript.

Dowek, G. (1994). Third order matching is decidable. *Annals of Pure and Applied Logic* **69**(2–3), 135–155.

Egg, M., Koller, A. and Niehren, J. (2001). The Constraint Language for Lambda Structures. *Journal of Logic, Language, and Information* **10**(4), 457–485.

Egg, M., Niehren, J., Ruhrberg, P. and Xu, F. (1998). Constraints over Lambda-Structures in Semantic Underspecification. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (ACL'98)*, 353–359. Montreal, Quebec, Canada.

Erk, K., Koller, A. and Niehren, J. (2002). Processing Underspecified Semantic Representations in the Constraint Language for Lambda Structures. *Journal of Research on Language and Computation* **1**, 127–169.

Erk, K. and Niehren, J. (2000). Parallelism Constraints. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications (RTA'00), July 10-12*, volume 1833 of *LNCS*, 110–126. Springer, Norwich, UK.

Erk, K. and Niehren, J. (2003). Well-Nested Parallelism Constraints for Ellipsis Resolution. In *Proceedings of the 11th Conference of the European Chapter of the Association of Computational Linguistics*, 115–122.

Farmer, W. M. (1988). A unification algorithm for second-order monadic terms. *Annals of Pure and Applied Logic* **39**, 131–174.

Farmer, W. M. (1991). Simple Second-Order Languages for wich Unification is Undecidable. *Theoretical Computer Science* **87**, 173–214.

Felty, A., Gunter, E., Miller, D. and Pfenning, F. (1990). λProlog. In *Proceedings of the 10th International Conference on Automated Deduction (CADE-10)*, volume 449 of *LNAI*, 682–681. Springer-Verlag, Kaiserslautern, FRG.

Gallier, J. H. and Snyder, W. (1990). Designing Unification Procedures Using Transformations: A Survey. *Bulletin of the EATCS* **40**, 273–326.

Ganzinger, H., Nieuwenhuis, R. and Nivela, P. (2001). Context Trees. In *Proceedings of the 1st International Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *LNCS*, 242–256. Siena, Italy.

Gardent, C. and Kohlhase, M. (1996). Higher-Order Coloured Unification and Natural Language Semantics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, 1–9. Association for Computational Linguistics, Morgan Kaufmann Publishers, San Francisco.

Goldfarb, W. D. (1981). The Undecidability of the Second-Order Unification Problem. *Theoretical Computer Science* **13**, 225–230.

de Groote, P. (2000). Higher-order linear matching is NP-complete. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications (RTA'00)*, volume 1833 of *LNCS*, 127–140. Springer-Verlag.

Gutiérrez, C. (1998). Satisfiability of word equations with constants is in exponential space. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS'98): proceedings: November 8–11, 1998, Palo Alto, California*, 112–119. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA.

Gutiérrez, C. (2000). Satisfiability of equations in free groups is in PSPACE. In *Proceedings of the 32nd annual ACM Symposium on Theory of Computing (STOC'00): Portland, Oregon, May,*, 21–27. ACM Press, New York, NY, USA.

Hindley, J. R. and Seldin, J. P. (1986). *An Introduction to Combinators and the λ-calculus*. Cambrigde University Press.

Horton, R. (1945). Erosional development of streams and their drainage basins; hydrophisical approach to quantitative morphology. *Bulletion of Geological Society of America* **56**, 275–370.

Huet, G. (1973a). A Mechanization of Type Theory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI'73)*, 139–146. William Kaufmann, Standford, CA.

Huet, G. (1973b). The Undecidability of Unification in Third-Order Logic. *Information and Control* **22**(3), 257–267.

Huet, G. (1975). A Unification Algorithm for Typed λ-Calculus. *Theoretical Computer Science* **1**, 27–57.

Huet, G. (1976). Résolutions d'Équations dans des Langages d'ordre 1,2,...,ω. Thèse d'État, Université de Paris VII.

Jaffar, J. (1990). Minimal and Complete Word Unification. *Journal of the ACM* **37**(1), 47–85.

Jensen, D. C. and Pietrzykowski, T. (1976). Mechanizing omega-order type theory through unification. *Theoretical Computer Science* **3**(2), 123–171.

Karhumäki, J., Plandowski, W. and Mignosi, F. (1997). The Expressibility of Languages and Relations by Word Equations. In *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *LNCS*, 98–109. Springer-Verlag, Bologna, Italy.

Knight, K. (1989). Unification: A Multidisciplinary Survey. *ACM Computing Surveys* **21**(1), 93–124.

Knuth, D. E. and Bendix, P. B. (1967). Simple Word Problems in Universal Algebras. In *Computational Problems in Abstract Algebra*, 263–298. Pergamon Press, Oxford. Appeared 1970.

Koller, A. (1998). Evaluating Context Unification for Semantic Underspecification. In *3rd ESSLLI Student Session (ESSLLI '98), August 17-28*, 188–199. Saarbrücken, Germany.

Koller, A., Niehren, J. and Treinen, R. (1998). Dominance Constraints: Algorithms and Complexity. In *Proceedings of the 3rd International Conference on Logical Aspects of Computational Linguistics (LACL'98), December 14-16*. Grenoble, France.

Kościelski, A. and Pacholski, L. (1995). Complexity of Makanin's Algorithm. Technical report, Institute of Mathematics, Polish Academy of Sciences.

Kościelski, A. and Pacholski, L. (1996). Complexity of Makanin's Algorithm. *Journal of the ACM* **43**(4), 670–684.

Kowalski, R. A. (1974). Predicate Logic as Programming Language. In *Information processing 1974; proceedings of IFIP congress 1974*, 569–574. North-Holland.

Levy, J. (1996). Linear Second-Order Unification. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA'96)*, volume 1103 of *LNCS*, 332–346. New Brunsbick, New Jersey.

Levy, J. (1998). Decidable and Undecidable Second-Order Unification Problems. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA'98)*, volume 1379 of *LNCS*, 47–60. Tsukuba, Japan.

Levy, J. and Agustí, J. (1996). Bi-rewrite Systems. *Journal of Symbolic Computation* **22**(3), 279–314.

Levy, J., Schmidt-Schauß, M. and Villaret, M. (2004). Monadic Second-Order Unification is NP-complete. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *LNCS*, 55–69. Aachen, Germany.

Levy, J. and Veanes, M. (1998). On Unification Problems in Restricted Second-Order Languages. In *Annual Conference of the European Association of Computer Science Logic (CSL'98)*. Brno, Czech Republic.

Levy, J. and Veanes, M. (2000). On the Undecidability of Second-Order Unification. *Information and Computation* **159**, 125–150.

Levy, J. and Villaret, M. (1998). Complexity Study of some Classes of Context and Second-Order Unification Problems. Twelfth International Workshop on Unification, (UNIF'98).

Levy, J. and Villaret, M. (2000). Linear Second-Order Unification and Context Unification with Tree-Regular Constraints. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications (RTA'00)*, volume 1833 of *LNCS*, 156–171. Norwich, UK.

Levy, J. and Villaret, M. (2001). Context Unification and Traversal Equations. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA'01)*, volume 2041 of *LNCS*, 169–184. Utrecht, The Netherlands.

Levy, J. and Villaret, M. (2002). Currying Second-order Unification Problems. In *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA'02)*, volume 2378 of *LNCS*, 326–339. Copenhagen, Denmark.

Loader, R. (2003). Higher-Order β Matching is Undecidable. *Logic Journal of the IGPL* **11**(1), 51–68.

Lucchesi, C. L. (1972). The Undecidability of the Unification Problem for Third-Order Languages. Technical Report CSRR 2059, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo.

Makanin, G. S. (1977). The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik* **32**(2), 129–198.

MARTELLI, A. and MONTANARI, U. (1976). Unification in linear time and space: A structured presentation. Internal Report B 76-16, Istituto di Elaborazione della Informazione, Pisa, Italy.

MILLER, D. (1991a). A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. *Journal of Logic and Computation* **1**(4), 497–536.

MILLER, D. (1991b). Unification of Simply Typed Lambda-Terms as Logic Programming. In *Proceedings of the 8th International Logic Programming Conference*, 255–269. MIT Press, Paris, France.

MILLER, D. (1992). Unification Under a Mixed Prefix. *Journal of Symbolic Computation* **14**(4), 321–358.

MILLER, D. and NADATHUR, G. (1986). Some Uses of Higher-Order Logic in Computational Linguistics. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics (ACL'86)*, 247–255. Association for Computational Linguistics, Morristown, New Jersey.

MILLER, D. and NADATHUR, G. (1987). A Logic Programming Approach to Manipulating Formulas and Programs. In *Proceedings of the 5th International Conference and Symposium on Logic Programming*, 379–388. IEEE, Computer Society Press, San Francisco.

MILLER, D., NADATHUR, G., PFENNING, F. and SCEDROV, A. (1991). Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic* **51**, 125–157.

MILNER, R. (1978). A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences* **17**, 348–375.

MONTAGUE, R. (1988). The Proper Treatment of Quantification in Ordinary English. In *Philosophy, Language, and Artificial Intelligence: Resources for Processing Natural Language*, 141–162. Kluwer, Boston.

DE MOOR, O. and SITTAMPALAM, G. (2001). Higher-order matching for program transformation. *Theoretical Computer Science* **269**(1–2), 135–162.

MÜLLER, M. and NIEHREN, J. (1998). Ordering Constraints over Feature Trees Expressed in Second-Order Monadic Logic. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA'98)*, volume 1379 of *LNCS*, 196–210. Tsukuba, Japan.

NADATHUR, G. and MILLER, D. (1998). Higher-Order Logic Programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, chapter 8. Oxford University Press.

NIEHREN, J. (2002). Personal Communication.

Niehren, J. and Koller, A. (2001). Dominance Constraints in Context Unification. In *Logical Aspects of Computational Linguistics (LACL'98)*, volume 2014 of *LNAI*, 199–218.

Niehren, J., Pinkal, M. and Ruhrberg, P. (1997a). On Equality Up-to Constraints over Finite Trees, Context Unification, and One-Step Rewriting. In *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, volume 1249 of *LNCS*, 34–48. Townsville, North Queensland, Australia.

Niehren, J., Pinkal, M. and Ruhrberg, P. (1997b). A Uniform Approach to Underspecification and Parallelism. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the Association for Computational Linguistics (ACL'97)*, 410–417. Madrid, Spain.

Niehren, J., Tison, S. and Treinen, R. (2000). On rewrite constraints and context unification. *Information Processing Letters* **74**(1-2), 35–40.

Niehren, J. and Villaret, M. (2002). Parallelism and Tree Regular Constraints. In *Proceedings of the 9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning: (LPAR'02)*, volume 2514 of *LNCS*, 311–326. Tbilisi, Georgia.

Niehren, J. and Villaret, M. (2003). Describing Lambda Terms in Context Unification. In *4th International Workshop on Inference on Computational Semantics (ICOS-4)*. Nancy, France.

Nieuwenhuis, R. and Rubio, A. (2001). Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, volume 1, chapter 7, 372–444. Elsevier Science Publishers and MIT Press.

Nipkow, T. (1993). Functional Unification of Higher-Order Patterns. In *Proceedings of the 8th IEEE Symposium on Logic in Computer Science (LICS'93)*, 64–74. Montreal, Canada.

Padovani, V. (1995). Decidability of All Minimal Models. In *Types for Proofs and Programs, International Workshop (TYPES'95)*, volume 1158, 201–215. Springer-Verlag.

Padovani, V. (2000). Decidability of Fourth-Order Matching. In *Mathematical Structures in Computer Science*, volume 10(3), 361–372. Cambridge University Press.

Paulson, L. C. (1990). Isabelle: The Next 700 Theorem Provers. In *Logic and Computer Science*, 361–386. Academic Press.

Paulson, L. C. (1993). Introduction to Isabelle. Technical Report UCAM-CL-TR-280, University of Cambridge, Computer Laboratory.

Pietrzykowski, T. (1973). A Complete Mechanization of Second-Order Type Theory. *Journal of the ACM* **20**(2), 333–365.

Pinkal, M. (1995). Radical Underspecification. In *10th Amsterdam Colloquium*, 587–606. University of Amsterdam, Amsterdam, The Netherlands.

Plandowski, W. (1999a). Satisfiability of word equations with constants is in NEXPTIME. In *Proceedings of the 31st annual ACM Symposium on Theory of Computing (STOC'99)*, 721–725. ACM Press, New York, NY, USA.

Plandowski, W. (1999b). Satisfiability of word equations with constants is in PSPACE. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, 495–500. IEEE Computer Society Press, New York City, USA.

Plotkin, G. (1972). Building in Equational Theories. In *Machine Intelligence*, volume 7, 73–90. Edinburgh University Press, Edinburgh, Scotland.

Prehofer, C. (1995). *Solving Higher-Order Equations: From Logic to Programming*. Ph.D. thesis, Technische Universität München.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM* **12**(1), 23–41.

Robinson, J. A. (1969). Mechanizing Higher-Order Logic. *Machine Intelligence* **4**, 151–170.

Robinson, J. A. and Wos, L. (1969). Paramodulation and Theorem-proving in First-Order Theories with Equality. *Machine Intelligence* **4**, 135–150.

Salvati, S. and de Groote, P. (2003). On the complexity of Higher-Order Matching in the Linear λ-calculus. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA'03)*, volume 2706 of *LNCS*, 234–245. Springer-Verlag.

Schmidt-Schauß, M. (1996). An algorithm for distributive unification. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA'96)*, volume 1103 of *LNCS*, 287–301. New Jersey, USA.

Schmidt-Schauß, M. (1998). A decision algorithm for distributive unification. *Theoretical Computer Science* **208**, 111–148.

Schmidt-Schauß, M. (1999a). Decidability of Bounded Second-Order Unification. Technical Report Frank-report-11, FB Informatik, J.W. Goethe Universität Frankfurt.

Schmidt-Schauß, M. (1999b). A Decision Algorithm for Stratified Context Unification. Technical Report Frank-report-12, FB Informatik, J.W. Goethe Universität Frankfurt.

Schmidt-Schauß, M. (2001). Stratified Context Unification Is in PSPACE. In *Proceedings of the 15th Workshop on Computer Science Logic (CSL'01)*, volume 2142 of *LNCS*, 498–512.

Schmidt-Schauß, M. (2002). A Decision Algorithm for Stratified Context Unification. *Journal of Logic and Computation* **12**, 929–953.

Schmidt-Schauß, M. (2004). Decidability of Bounded Second-Order Unification. *Information and Computation* **188**(2), 143–178.

Schmidt-Schauß, M. and Schulz, K. U. (1998). On the Exponent of Periodicity of Minimal Solutions of Context Equations. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA'98)*, volume 1379 of *LNCS*, 61–75. Tsukuba, Japan.

Schmidt-Schauß, M. and Schulz, K. U. (1999). Solvability of context equations with two context variables is decidable. In *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *LNAI*, 67–81.

Schmidt-Schauß, M. and Schulz, K. U. (2002a). Decidability of Bounded Higher-Order Unification. In *Proceedings of the 16th Workshop on Computer Science Logic (CSL'02)*, volume 2471 of *LNCS*, 522–536.

Schmidt-Schauß, M. and Schulz, K. U. (2002b). Solvability of Context Equations with Two Context Variables is Decidable. *Journal of Symbolic Computation* **33**(1), 77–122.

Schmidt-Schauß, M. and Stuber, J. (2002). On the complexity of linear and stratified context matching problems. In *Complexity in Automated Deduction*, volume 02-08 of *DIKU technical reports*, (unpaginated).

Schubert, A. (1998). Second-order unification and type inference for Church-style polymorphism. In *Conference Record Symposium on Principles of Programming Languages*, 279–288. ACM Press.

Schulz, K. U. (1991). Makanin's algorithm for word equations — Two improvements and a generalization. In *Word Equations and Related Topics*, number 572 in LNCS, 85–150. Springer, Berlin-Heidelberg-New York.

Schulz, K. U. (1993). Word Unification and Transformation of Generalized Equations. *Journal of Automated Reasoning* **11**(2), 149–184.

Shieber, S., Pereira, F. and Dalrymple, M. (1996). Interaction of Scope and Ellipsis. *Linguistics & Philosophy* **19**, 527–552.

Siekmann, J. and Szabó, P. (1984). Universal Unification. In *Proceedings 7th International Conference on Automated Deduction (CADE-7), Napa Valley (California, USA)*, volume 170 of *LNCS*, 1–42. Springer-Verlag, Napa Valley (California, USA).

Sittampalam, G. and de Moor, O. (2001). Higher-Order Pattern Matching for Automatically Applying Fusion Transformations. In *Symposium on Programs as Data Objects*, volume 2053 of *LNCS*, 218–237.

Strahler, A. N. (1952). Hypsometric (area-altitude) analysis of erosional topography. *Bulletion of Geological Society of America* **63**, 1117–1142.

Thatcher, J. W. and Wright, J. B. (1967). Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory* **2**(1), 57–81.

Venturini-Zilli, M. (1975). Complexity of the Unification Algorithm for first-Order Expressions. *Calcolo XII, Fasc. IV* 423–434.

Zaionc, M. (1986). The Set of Unifiers in Typed Lambda-Calculus as Regular Expression. In *Proceedings of the 1st International Conference on Rewriting Techniques and Applications (RTA'85)*, volume 202 of *LNCS*, 430. Springer-Verlag, Berlin-Heidelberg-New York.

Index

(τ, λ), 124
A, 103
A-unification, 4
B, 103
C, 43
E-unification, 3
Q, 101
Q_{fin}, 139
Q_f, 101
R, 79
R^k_Σ, 79
R_q, 110
V, 138
X_{all}, 150
[ ], 74
$:$, 121
$=$, 121
$\perp$, 121
$\lhd^*$, 121
$\lhd^+$, 121
$\mathcal{A}_{\mathsf{ext}_V}$, 141
Δ, 101
domain_τ, 138
equ_ϕ, 132
$\mathcal{V}_{\mathrm{node}}$, 125
$\mathbb{NF}(\ )$, 77
Φ, 133
$\Phi'(\ )$, 115
$\Phi(\ )$, 83
Π_n, 74
$\mathsf{proj}(\ )$, 134
$\mathsf{root}(\ )$, 121
$\mathsf{root}(\ ,\ )$, 134
$\mathcal{V}_{\mathrm{set}}$, 138
$\Sigma\phi$, 133
Σ, 20
Σ', 103
Σ^L, 62
Σ^Π, 75
Σ^c, 61
Σ_V, 140
Σ_Π, 75
α-equivalence, 23
α-extension, 140
$\alpha(\ )$, 126
$arity(\ )$, 20
β-equivalence, 23
β-redex, 23
β-reduction, 23
$\beta\eta$-long normal form, 23
$\mathsf{bind}_f(\ ,\ )$, 132
$\circ$, 25
rank( ), 74
rank($\mathcal{A}$), 109
$\mathcal{C}(\ )$, 61
$Dom(\ )$, 21
ϵ, 74
$\equiv$, 79
$\equiv_b$, 82
$=_\lambda$, 22
η, 145, 150
η-equivalence, 23
η-expansion, 23
η-redex, 23
η-reduction, 23
$\mathsf{ext}_\alpha(\ )$, 140
$\mathsf{external{-}binder}_e(\ ,\ )$, 134
$\widehat{\mathcal{L}}$, 62
$\mathsf{inside}(\ ,\ )$, 134
$\mathcal{L}$, 62
$\mathsf{lam}(\ )$, 123
$\mathsf{lam}_u(\ )$, 129

λ-Prolog, 8
λ-abstraction, 20
λ-calculus, 19
λ-equivalence, 22
λ-term, 20
    linear, 39
    order, 20
    size, 20
    typed, 20
$\lambda(\ )$, 123, 124
$\lambda^{-1}(\ )$, 124
$\lceil\ \rceil$, 147
$\lfloor\ \rfloor$, 150
$\mapsto$, 21
$\mathcal{A}$, 101
$\mathcal{A}^{A,B}$, 104
$\mathcal{L}(\mathcal{A})$, 101
$\mathcal{L}istsof\ (B)$, 103
$\mathcal{T}$, 19
$\mathcal{T}(\ ,\ )$, 88
$\mathcal{T}(\Sigma, \mathcal{X})$, 20
$\mathcal{W}$, 79
$\mathcal{X}$, 20
$\models$, 127
μ, 126
names_e, 133
$\mathsf{nodes}_\tau^-(\ )$, 122
nodes_τ, 121
$\mathsf{nodes}_\tau(\ )$, 122
$\mathsf{no{-}free{-}var}_e$, 134
$\mathsf{no{-}hang{-}binder}_e(\ )$, 134
ν, 126
ϕ, 126
ψ, 138
$Range(\ )$, 21
$\pi/\pi_1, \ldots, \pi_n$, 122
$\sigma|_A$, 21
$\sim$, 123
$\mathsf{intervene}_f(\ ,\ )$, 132
$|\ |$, 20
$\mathsf{step}_{\mathcal{A}}(\ ,\ )$, 139
$\tau.\pi$, 121
τ, 121
$\tau(\ )$, 20, 121
$\rightarrow$, 19
$\rightarrow_\beta$, 23
$\rightarrow_\eta$, 23
$trans^{A,B}(\ )$, 103
$[\ ]_e$, 133
trav( ), 75
tree( ), 138
$\mathsf{tree}_e(\ ,\ )$, 136
$\stackrel{?}{=}$, 25
$\stackrel{?}{=}_{cu}$, 102
$\stackrel{?}{=}_{lsou}$, 102
var, 123
var_u, 129
$Var(\ )$, 20
$\vec{x}$, 102
$\widehat{\ }$, 62
ξ, 138
$b_{\vec{x}}$, 103
$c(\ )$, 123
c_z, 103
$ch(\ )$, 140
$e(\ )$, 132
e_V, 150
f^ρ, 75
n^G, 27
$o(\ )$, 19
$p(\ )$, 134
p_X, 104
q_X, 104
$t|_p$, 74
@, 57, 61, 123
1-in-3-SAT, 53

$\alpha_{|V}$, 127

anaphoric binding, 120
application, 20
    explicit, 57
argument, 20
arity, 20
automated theorem proving, 7

beta reduction constraints, 120
binder, 20
bounded higher-order unification, 6, 52

bounded second-order unification, 6, 52

common instance, 25
Comon's restricted case, 48
constant symbols, 20
constants, 20
constraint language for lambda structures, 14, 119
context, 43
context matching, 53
context unification, 5, 44
  currying, 57
  decidable fragments, 48
  perspectives, 43, 45
  stratified, 50
context unification with tree-regular constraints, 102
context variable, 43
  arity, 45
correspondence function, 123
curried signature, 61
currying function, 61
currying terms, 61
cycle, 50

decurried form, 24
disjointness relation, 121
distributive unification, 50
disunification, 1
dominance constraint, 125
dominance constraints, 14
dominance relation, 121

ellipses, 13
equation, 25
equations system, 25
  size, 25
explicit unification, 58
exponent of periodicity, 49

first-order dominance formula, 126
first-order unification, 1, 34
flexible, 29
functions, 20

Goldfarb numbers, 27
graph representation, 128
group parallelism constraint, 120
group parallelism constraints, 154

Haskell, 11
hat, 62
head, 23
higher-order β-matching, 37
higher-order logic programming, 8
higher-order matching, 36
  currying, 68
higher-order patterns, 28
higher-order unification, 6, 25
Hilbert's tenth problem, 27
hole, 43
holes, 122

instance, 22
  common, 25
intervenance, 129
Isabelle, 8

labeling function, 62
labeling relation, 121
lambda binding constraint, 126
lambda binding constraints, 126
lambda structure, 124
language
  order, 24
linear higher-order matching, 53
linear second-order matching, 53
linear second-order unification, 6, 39
linear second-order unification procedure, 41

MAG, 11
matching, 1
  context, 53
  higher-order, 25, 36, 53
  linear higher-order, 53
  linear second-order, 53
  second-order, 25
matings, 2
monadic, 28
monadic second-order dominance logic, 138

monadic second-order unification, 28, 35

natural language semantics, 12
node variables, 125
non-intervenance, 129
non-intervenance property, 130
non-touching variables, 64

parallel lambda binding axioms, 124
  external binder, 124
  internal binder, 124
  no hanging binder, 124
parallelism constraint, 126
parallelism constraints, 14, 126
parallelism relation, 123
parametric terms, 35
paramodulation, 3
Pascal, 10
pattern, 34
permutation, 74
pre-unification, 32
program synthesis, 9
program transformation, 9
Prolog, 2

rank, 72
rank-bound conjecture, 83
  extended, 115
rank-bound tree-regular constraint, 114
regular constraint, 79
  rank-bound, 79
regular search trees, 33
renaming, 25
resolution, 2
rigid, 29
root, 121

scope ambiguity, 12
second-order unification, 6, 25
  currying, 57
  decidable fragments, 28, 34
  infinitary, 29
  procedure, 29
  undecidability, 27
  undecidable fragments, 28
second-order unification procedure
  completeness, 32
  soundness, 32
segment, 122
sequence, 75
  k-bound traversal, 78
  normal traversal, 77
  rank, 76
  traversal, 75
  width, 76
signature, 20
  extended, 75
  extended labels, 140
  labeled, 62
simple equations, 28
simply typed λ-calculus, 19
simultaneous rigid E-unification, 28
Snobol, 9
solved state, 29
stratified context unification, 50
strict dominance, 121
substitution, 21
  application, 22
  composition, 25
  context, 44
  domain, 21
  extension, 21
  ground, 22
  linear, 39
  more general, 25
  range, 21
  renaming, 25
  restriction, 21
  size, 22
  word, 79
subterm, 20

term, 43
  closed, 20
  curry form, 57
  first-order, 21
  position, 74
  rank, 72, 74
  second-order, 21
transformation rule, 29

elimination, 31
flexible-flexible, 31, 41
flexible-rigid, 30
identification, 32
imitation, 31, 41
iteration, 31
projection, 30, 41
rigid-rigid, 30
simplification, 30, 31, 41
traversal equation, 79
permutation-bound, 82
traversal sequence, 75
permutation-bound, 82
traversal system, 79
permutation and rank-bound, 82
rank-bound, 79
tree, 121
tree automaton, 101
rank-bound, 109
tree node, 121
tree structure, 121
tree-regular constraint, 102, 137
rank-bound, 114
tree-regular language, 101
rank-bound, 114
two context variables fragment, 51
type, 19
atomic, 19
order, 19
typed λ-term *see* λ-term, 20

unarise, 46
underspecified semantic representation, 14
unification, 1, 25
bounded higher-order, 52
bounded second-order, 52
context, 44
explicit, 58
finitary, 26
first-order, 34
higher-order, 25
infinitary, 26
linear second-order, 39
monadic second-order, 35
nullary, 26
pattern, 34
problem, 26
second-order, 25
unitary, 26
word, 48
unifier, 25
minimal complete set of most general, 26
most general, 25

variable, 20
bound, 20
capture, 21, 129
context, 43
free, 20
fresh, 22
variable assignment, 126

word equation, 79
word unification, 4, 48
word unification with regular constraints, 50

XML, 53

Z-context unification, 52