FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Unification in Higher-order Resolution

DISSERTATION

zur Erlangung des akademischen Grades

Doktor/in der technischen Wissenschaften

eingereicht von

Tomer Líbal
Matrikelnummer 0627906

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr.phil. Alexander Leitsch

Diese Dissertation haben begutachtet:

(Univ.Prof. Dr.phil. Alexander Leitsch)

(Prof. Dr. Manfred Schmidt-Schauß)

Wien, 15.12.2012

(Tomer Líbal)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Unification in Higher-order Resolution

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor/in der technischen Wissenschaften

by

Tomer Líbal
Registration Number 0627906

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dr.phil. Alexander Leitsch

The dissertation has been reviewed by:

(Univ.Prof. Dr.phil. Alexander Leitsch)

(Prof. Dr. Manfred Schmidt-Schauß)

Wien, 15.12.2012

(Tomer Líbal)

Erklärung zur Verfassung der Arbeit

Tomer Líbal
Pettenkofengasse 3/9, 1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

I would like to express my deepest gratitude to my advisor, Alexander Leitsch. First for allowing me to participate in the interesting multidisciplinary research done in his group. Second, for showing great patience when my own interest deviated a little from the project goals before converging again and most importantly, for all the help and guidance he provided during the last four years. I am greatly indebted also to Manfred Schmidt-Schauß, whose work on unification led to this thesis and who did not hesitate spending valuable time helping an anonymous student.

My colleagues in the theory and logic group, Cvetan Dunchev, Stefan Hetzl, Daniel Weller, Bruno Woltzenlogel Paleo, Martin Riener, David Cerna, Mikheil Rukhaia, Paolo Baldi, Lara Spendier and Giselle Machado N. Reis, created not only a scientific stimulating atmosphere but also a social one, without which, a foreign student in Vienna cannot survive.

I would like to give special thanks for many interesting discussions (and arguments) to Stefan, Daniel, Bruno and Martin, who also helped me with written German, to David, who has helped me with written English and to Cvetan for helping me (and my family) with almost everything else. I also thank Paolo, Lara, Mikheil and Giselle for many nice "Mahlzeits" together.

Last but not least, I thank my wife and best friend, Irina, for her support, patience and love; I thank my daughter, Eniko, for playing with me when I got bored with logic and my parents, Smadar and Zvi Líbal for their continuous support.

This thesis is dedicated to my grandparents, Heni and Meir Ginsburg and Olga and Moshe Líbal.

Abstract

The mathematical analysis of proofs and the creation of modern proof calculi were originally aiming more at analyzing the properties of existing proofs and less at the creation and discovery of new proofs. This has changed with the introduction of computers. The advances in computer technology which resulted in increasing computing power made it more practical to search for new proofs as well. This process has culminated with the invention of the resolution calculus, a logical calculus which is extremely suitable for mechanical processing.

The resolution calculus was first introduced for first-order logic, in which the calculus enjoys many advantages and the search complexity is relatively small. Since the resolution calculus is based on the unification principle, one of the main advantages is the fact that the first-order unification problem is decidable and unitary. When lifting the calculus to higher-order logic, in which the formalizing of mathematical problems is more natural, several issues arise which render the calculus less useful in practice. The foremost of these issues is the complexity of the higher-order unification problem, which is now both infinitary and undecidable.

The majority of the higher-order resolution calculi and their implementations are based on either an unrestricted higher-order unification or on strongly restricted higher-order unification in which the size of the generated unifiers is restricted. The existence of more refined higher-order unification algorithms does not translate directly into more efficient resolution calculi as these algorithms are normally not well suited for automated deduction.

The main aim of this thesis is to bridge the gap between the practicability of the higher-order resolution calculus and the efficiency of the more refined unification algorithms. The weakness of these unification algorithms with regard to automated deduction is investigated and more suitable algorithms are defined. On the other hand, the search strategies in the resolution calculi are modified in order to better suit the unification algorithms.

The obtained resolution calculi are compared with existing ones using a set of test cases. The conclusion drawn from this comparison is that there is a real advantage in considering the calculi introduced in this thesis when considering various classes of problems and in particular, second-order arithmetical problems.

Kurzfassung

Die Methoden der Beweisanalyse und die in ihrem Kontext entwickelten logischen Kalküle wurden ursprünglich zur Erlangung theoretischer Ergebnisse herangezogen. Die zunehmende Rechenkapazität von Computern erlaubt jedoch inzwischen auch die praktische Anwendung dieser Methoden zur Beweissuche. Einer der zentralen Fortschritte war Robinsons Einführung des Resolutionskalküls, der sich speziell zur automatisierten Beweissuche eignet. Gerade im ursprünglichen Anwendungsgebiet der Prädikatenlogik erster Stufe besitzt es eine verhältnismäßig niedrige Suchkomplexität. Unifikation, das Kernstück des Resolutionskalküls, ist hier effizient entscheidbar und unitär, d.h. es gibt nur höchstens einen allgemeinsten Unifikator.

Zum Formalisieren von mathematischen Beweisen sind die Prädikatenlogiken höherer Stufe wesentlich besser geeignet als diejenige erster Stufe. In diesem Fall ist die Unifikation jedoch nur mehr semi-entscheidbar: die Menge der allgemeinsten (Prä-)Unifikatoren ist von abzählbar unendlicher Kardinalität, die Suche nach einem Unifikator terminiert im Allgemeinen auch nicht.

Wissenschaftlich behandelt wurden bisher hauptsächlich der allgemeine Fall und syntaktische Einschränkungen wie Anzahl und Typ der vorkommenden Variablen oder die Termintiefe, mit dem Ziel das Teilproblem entscheidbar zu machen. Das Zusammenspiel von Unifikation und Resolution erfordert noch eine gesonderte Betrachtung, da die Unifikation etwa oft aufgeschoben werden muss, eine möglichst frühe Elimination von nicht unifizierbaren Termen aber von Vorteil ist.

Thema der Dissertation ist, die verfeinerten Unifikationsalgorithmen für die Anwendung in der Resolution praktikabel zu machen. Sie behandelt die Nachteile der aktuellen Algorithmen und stellt sie den hier entwickelten Verbesserungen gegenüber. Ein besonderes Augenmerk wird dabei auf die unterschiedlichen Suchstrategien gelegt. Die Gegenüberstellung findet anhand einer Reihe von Testfällen statt und zeigt exemplarisch die Effizienz der neuen Algorithmen, was speziell im Fall der Theorie der Arithmetik zweiter Ordnung gelingt.

Contents

CHAPTER 1

Introduction

Logic is the study of the principles of valid reasoning, inference and demonstration. The study of logic spans from prehistoric times until our days and new results are continuously emerging [14]. While logical thinking has probably taken place as early as the time organisms were able to think at all, logical thinking about logical thinking has evolved much later. The ancient Egyptians have managed to empirically obtain logical truths with regard to Geometry [51] while the ancient Babylonians have observed and wrote down axioms and assumptions in their study of Medicine [46]. The first use of demonstrations and proofs had taken place in Greece. Already in the 6th century B.C. scholars like Thales [31] and Pythagoras had used such methods and might have even developed deductive systems. The Greeks were also responsible for the development of the first logical calculi. The Stoics had developed a calculus of propositions [51] while the foundations of calculi for predicate logic and inductive arguments were laid down by Aristotle and his school [33].

The first mathematical models of logical inference, i.e. the first algebras of logic, are the calculi of Boole [13] and Frege [30]. In the beginning of the 20th century, Hilbert type [44] and Gentzen type [35] calculi were developed in order to formalize all mathematical knowledge and in order to investigate the notion of provability, but the actual use of these calculi in order to find proofs for mathematical problems was very limited.

The mechanical task of proving mathematical theorems can be traced back to the 17th century and to Leibniz's "calculus ratiocinator" [54], a method which attempts to obtain a numerical proof of correct theorems. Since a complete mechanization of mathematics is not possible (see Gödel's Incompleteness theorem [40] and Turing's Halting problem [79]) the only thing that could be hoped for is an efficient proof search for subsets of mathematics.

The invention of the computer lead to a renewed interest in the mechanization of mathematics. Since the proof search in predicate logic is infinite, early provers [36], [23] were based on a search for propositional instances based on Herbrand's theorem [43]. These provers failed, however, to prove even simple mathematical theorems. One of the reasons for their failure is that they were based on two search procedures, for obtaining the right propositional instances and then for validating each one of them. Both these procedures are very expensive to execute, even with today's computation power.

A major breakthrough came with the invention of unification and the resolution calculus [68]. The new calculus was proved to be complete with regard to searching for refutations of unsatisfiable first-order problems of finite size and remained the most efficient calculus for automated reasoning to this day.

The strength of the calculus can be attested by it being the first fully automated method to solve mathematical problems [60]. Still, such results are very rare and failures to overcome the main obstacles during the last decades have lead to the consideration of second and higher-order automated theorem provers.

When considering arithmetic, the restriction to first-order logic poses severe limitations. The majority of the theorems in arithmetic involves the uses of variants of the induction principle and this principle cannot be encoded in a finite way in first-order logic. In second-order logic, this principle can easily be encoded in the form of an induction axiom. The discipline of "reverse mathematics" has identified subsystems of second-order arithmetic which are sufficient to prove many interesting theorems [75]. One of these subsystems, called $\mathtt{ACA}_0$, even poses strong restrictions on the possible instantiations of the induction axiom.

On the other hand, the consideration of higher-order arithmetic poses strong computational difficulties. The unification principle [49], which was used in order to obtain a (most) general substitution, of which is sufficient for the refutations of first-order problems, fails to do the same for higher-order problems. The computations of such substitutions even for second-order problems may never terminate [37] and in case of termination, there might exist an infinite number of such general substitutions.

Nevertheless, research into higher-order unification provided several subclasses of terms which can be (relatively) easily unified. The most straightforward extension of first-order unification is patterns unification [61] which has applications to programming languages. Another important subclass of second-order logic is stratified context unification [70], a method with applications in computational linguistics [27]. The most famous of these subclasses is string unification, shown decidable in [59].

There is also a considerable number of higher-order theorem provers for classical logic, mainly based on the higher-order resolution calculus [3] and its variants. Among the most popular ones are Isabelle [66], TPS [2] and LEO II [9]. These theorem provers target general higher-order problems and therefore do not contain refinements of the unification principle. In particular, they also target problems in second-order arithmetic but fail to fully automate solutions even to simple ones. One additional complexity which higher-order resolution theorem provers have to deal with is the requirement of terms to be in normal forms. This requirement is of low complexity in first-order logic but introduces into the higher-order calculus new rules which greatly increase its complexity.

1.1 Resolution for Second-order Arithmetic

In the previous section we have mentioned that a full mechanization of all logical statements (even in first-order logic) is impossible. One should note that, when considering higher-order logic, even the mechanical construction of all valid formulas is no longer possible. Take for example the Gödel formulas, which are used in his incompleteness theorem. These formulas are true in second-order arithmetic but cannot be proved.

This non-completeness of higher-order logic closely relates to the comprehension axioms, which state that higher-order variables are quantified over sets which contain at least all definable elements of the right type. If we restrict the models of higher-order logic to contain exactly all definable elements then we obtain a complete semantics for higher-order logic called Henkin's semantics [42].

While the use of non-standard semantics for arithmetic does not always make sense, it does so in the context of higher-order resolution. Since substitution is a form of comprehension, Henkin's completeness is the strongest semantics we can hope to achieve. Indeed, most higher-order theorem provers aim of obtaining Henkin's completeness, or even a weaker one.

A major characteristic of second-order arithmetic is the usage of set variables as the only type of higher-order variables in the problems. Usually second-order arithmetical problems also contain a very simple set of functions symbols, such as the addition, successor and zero. The necessity of terms to be in normal forms, however, requires the application of Skolemization [62] and may result in terms of complex types.

When considering automated theorem provers for second-order arithmetic, resolution theorem provers might appear out of place. Moreover, the existence of the induction axiom poses severe complications to any theorem prover. The non-admissibility of cut-elimination [32] prevents resolution, with its atomic cuts only, from being complete in any sensible sense. One can consider only complete fragments of second-order arithmetic in which the cut rule is admissible [58], but the resulting fragments may not be very expressive. A more natural approach might be to use inductive theorem provers, such as INKA [12], Oyster-Clam [15] and ACL2 [50].

Another motivation for an efficient proof search for second-order arithmetic can be found in the CERes method [5]. This cut elimination method requires the refutation of intermediate clauses sets. The method is implemented [26] within the GAPT[1] framework but its dependency on resolution theorem provers restricts its usefulness on second-order arithmetical problems.

Currently there are two approaches for overcoming this weakness of CERes. The existence of a schematic propositional theorem prover [4] has allowed for a version of CERes for schematic propositional proofs [25] and work is currently in progress for supporting schematic first-order proofs as well[2]. Another approach is to find a resolution calculus which is more suitable for the refutation of second-order arithmetical problems, either by having a decidable unification algorithm or one which is more natural for the unification of arithmetical terms, either by having a good interactiveness with the user or by refining the search space.

This last point has motivated the current thesis, which initially targeted efficient resolution calculi for subsystems of second-order arithmetic and tried to find a variant of Huet's unification algorithm which decides the unification problem for such terms. Early attempts were not successful and have resulted in a proof for the undecidability of the unification problem over second-order arithmetical terms[3]. The proof can be obtained by a reduction to a language which

[1]http://code.google.com/p/gapt/

[2]http://www.logic.at/asap/

[3]Personal communication with Daniel Weller (2009)

contains only one binary function symbol and unary second-order variables and in which the unification problem was proved to be undecidable [29].

Further attempts concentrated on subsystems of second-order arithmetic. The restrictions on $\mathtt{ACA}_0$ mentioned above lead to the investigation of the application of unification algorithms for ramified type theory [41] whose unification problem is decidable [38]. Unfortunately, the introduction of Skolem symbols prevented the ramification of terms and the problem turned out to be non-ramified.

These failures have lead to the investigation of properties of higher-order unification algorithms in general.

1.2 Properties of Higher-order Unification

The first property of higher-order unification one might want to observe is its relationship with first-order logic. Such a comparison can be obtained by translating higher-order problems into first-order ones. For example by using the function f_1 which replaces all higher-order variables with first-order ones and simplifies the types of the terms. Consider for example the higher-order unification problem:

$$f(\lambda z.Y(a,z), X(\lambda z.z)) \doteq f(g, Y(a,a)) \tag{1.1}$$

By applying the function f_1 from above we obtain the following problem:

$$f(y,x) \doteq f(g,y) \tag{1.2}$$

which is unifiable by the substitution $[g/y, g/x]$. By translating the substitution back using an extension of f_1^{-1} we obtain the substitution $[\lambda z_1, z_2.g/Y, \lambda z.g/X]$ which is a unifier to the original problem.

We can easily prove that given a higher-order unification problem P, if $f_1(P)$ is unifiable by σ, then P is unifiable as well.

What about the other direction? Clearly if we could conclude from the unifiability of a higher-order problem P that $f_1(P)$ is unifiable as well then the two problems will be equivalent. But since the first-order unification problem is decidable while the higher-order one is not, we know that such a relationship is impossible.

It is possible, however, to define a function f_2 from higher-order problems to first-order problems which exhibits the above property. Let us call higher-order terms whose head symbol is a variable by the name flex terms. We can define f_2 to replace every occurrence of a flex term in a higher-order problem by a fresh first-order variable. We can even improve the definition and map flex terms which are applied to the same arguments to the same first-order variables. Consider the higher-order unification problem:

$$Xfa \doteq fXa \tag{1.3}$$

and its translation to a first-order problem by f_2:

$$x_1 \doteq fx_2 \tag{1.4}$$

The higher-order problem is unifiable and so is its first-order translation. It is easy to see that its translation by f_1:

$$x \doteq fx \tag{1.5}$$

cannot be unifiable by finite terms. It is, however, unifiable by the substitution t/x where $t = f^*$ is an infinite term.

Can we claim that a higher-order problem is unifiable if its translation by f_1 is unifiable, possibly by infinite terms?

An interesting property is that if a first-order problem can be unifiable by finite or infinite terms then it has, not only a most general unifier, but a regular most general unifier as well [22].

The meaning of a regular substitution in this chapter, and indeed in the whole thesis, is a substitution that maps variables to regular terms where regular terms may contain the Kleene star, denoting unbound repetitiveness.

This result, which was obtained by using Huet's first-order unification algorithm [47], hints that the answer to the question above is negative. This is due to the fact that we can, in this case, decide the unifiability problem of higher-order logic and indeed consider the following example:

$$Xfa \doteq fXb \tag{1.6}$$

which is not unifiable but its translation by f_1 is still unifiable by the regular substitution we found before.

The question whether we can use a first-order algorithm in order to solve higher-order problems is interesting from another point of view as well. Since higher-order problems may possess an infinite number of most general unifiers, by reducing the problems to first-order problems we might be able to compute a finite representation of all the most general ones.

As we have seen in the last example, the question is not trivial because any first-order reduction will ignore the information contained in the arguments of the higher-order variables and therefore a unifier for the first-order problem might not unify the higher-order one.

We might also ask what do we mean by a finite representation of all most-general unifiers as the unification problem itself forms a finite representation of them. Indeed, this strategy was deployed by Huet in order to define a resolution calculus which can deal with infinitely-many most general unifiers [48]. But our notion of a finite representation requires more than that. A finite representation, in our opinion, is a representation which can answer the membership question of possible unifiers. A task which Huet's higher-order pre-unification algorithm cannot perform [49] and indeed it seems that this task cannot be achieved. Even when considering string unification [59], which is a very simple subclass of higher-order unification, it was proved that one cannot obtain such a finite representation for problems containing more than three variables [45]. Nevertheless, a positive result exists for the case of up to three variables [1] and recently such a result was also obtained for the more general context unification problem if the problem contain at most one higher-order variable [34].

These negative (and positive) results allow us to rephrase our previous question in a more general way.

1. Given a solvable higher-order problem, do regular solutions always exist?

2. If they do, can we always compute them?
3. If we can, can we prove that they describe at least all possible unifiers of the higher-order problem?

These questions will be investigated further along the thesis and partial answers will be given in Chap. 4.

Huet's first-order unification algorithm [47] manages to deal with regular unifiers by processing cyclic equations as well as acyclic ones, as in the traditional first-order unification algorithms. In his technical report [53], Le Chenadec took this result and attempted to build a finite-state automaton which describes these regular unifiers. The construction of such an automaton depends on the possibility to obtain a normal form for cyclic problems [18]. Another result about the regular form of unifiers was obtained by Zaionc [80]. The requirement for normal forms and the regularity of the terms occurs also in the algorithms by G. S. Makanin, M. Schmidt-Schauß, K. U. Schulz and others with regard to minimal solutions for unification problems and will be discussed in the next section.

1.3 Higher-order Unification in Practice

In order to be used in practice, unification algorithms need not deal with the problems discussed in the previous section. In fact, most higher-order resolution theorem provers obtain a finite set of most general unifiers by restricting their syntactic forms. The LEO II [9] theorem prover obtains such a restricted set by computing new terms of fixed bounded depth only. Although this solution works in practice and can help to solve many unification problems, there are many interesting problems for which no such bound exists.

Consider the following example from [52].

Example 1.3.1. *We define as a string unification problem the problem of finding a unifier to a first-order problem containing first-order variables and constants and the binary infix function symbol* $'.'$*, which is associative. For example* $a.x.b = y.c$ *is a string unification problem. Note that we do not need parentheses as the only binary function symbol is associative. Since* $'.'$ *is the only binary symbol, we drop it altogether from the notation and assume its existence between any two constants and variables.*

The following string equation over strings and under the assumption of the associativity of the $'.'$ *infix function has a unique unifier* σ *such that* $\sigma(x_1) = a^{3^n}$*. The depth of terms we need to search for cannot be smaller than* 3^n *but the size of the problem is only* $6n - 2$*.*

$$x_1 b x_2 b ... b x_n = x_2 x_2 x_2 b x_3 x_3 x_3 b ... b x_n x_n x_n b a a a \tag{1.7}$$

Some representations of arithmetic, for example those using Church's numbers, cannot be bounded in such a way as well.

Another approach is to look for unifiers without any fixed bound but which are minimal with respect to other unifiers of the problem. The advantage of such a method is that the unifiability problem is then decidable. This method is based on the regularity result which was mentioned in the previous section. We first have to prove that for each variable in the problem we can compute a set containing the mappings to this variable in all possible unifiers of the problem. If each member in these sets is regular, then a bound can be computed on the size of these members which will make each set finite. The unifiability test proceeds by trying all possible mappings in each of the sets.

This is, more or less, the method used in order to prove the decidability of the unification problem in string unification [59], in monadic second-order problems [28], in distributive unification [69] and in several subclasses of context unification [70], [73].

A general result for the bound used in this method and which is applicable to all these problems can be found in [72].

A common property to all these problems is that the terms which are mapped to higher-order variables have a restricted form. In string unification, a variable can be mapped to λ-abstractions with exactly one occurrence of the bound variable. The following example shows the elevation of Ex. 1.3.1 to such a form:

Example 1.3.2. *Let $X_1, .., X_n$ be second-order variables and assume each one of them can be mapped to a $\lambda-$term containing exactly one occurrence of the bound variable, then the following equation has exactly the same solutions, up to interpretation, as the one from Ex. 1.3.1. I.e the unique unifier σ satisfies $\sigma(X_1) = \lambda z.a^{3^n} z$.*

$$X_1 b X_2 b ... b X_n c = X_2 X_2 X_2 b X_3 X_3 X_3 b ... b X_n X_n X_n baaac \tag{1.8}$$

In monadic second-order problems, the form of the function symbols allows us again to have at most one occurrence of the bound variable. The remaining algorithms are all based on context unification, in which the restriction on the number of bound variables is artificial. The higher-order variables allowed in these problems, the context variables, can by definition be mapped to λ-abstractions with exactly one occurrence of a bound variable.

At this point one can ask if this requirement is necessary in order to obtain the decidability of unification. In order to answer this question, we need to consider the role of such a restriction. There is some information in the problem which is not and cannot be taken into account when computing the regular terms. This information is contained in the arguments of the variables under consideration. Since we cannot know the form of the terms replacing these variables in possible unifiers, we cannot also know how the arguments will be introduced, if they are introduced at all. Since by β-reduction these arguments are going to replace all occurrences of bound variables in the terms, if we have information about the number of these variables we might be able to factor out the importance of the missing information. This is exactly the idea in the above algorithms. By having or allowing at most one such bound variable, we factor out the effect of the arguments.

Assume that we do not have this restriction, then it is still possible to compute the regular terms as above but now we might need to compute infinitely-many such regular terms. In this case the unification problem will not be decidable anymore.

Can we relax this requirement? I.e. can we allow a restricted number of bound variables greater than one? The answer to this question is positive as was shown both for second-order [71] and for higher-order logic [74].

To summarize what we have obtained so far. The two known ways for obtaining termination of higher-order unification and for deciding the unifiability of higher-order terms is either by bounding the size of generated terms as is usually done in higher-order theorem proving, or by bounding their number of bound variable occurrences.

As was seen in Ex. 1.3.2, the second way might allow for a much more expressive unification as although the size of terms grow exponentially with the size of the problem, the number of bound variable occurrences remains one.

Still, in the decade or so since the introduction of these unification algorithms, they were not used, as far as we know, in automated theorem proving. The reason for that is probably the fact that these unification algorithms only decide the unification problem and do not generate most general unifiers, which are necessary for the process of resolution. The results obtained in Chap. 4 will allow us to overcome this restriction by computing, not only minimal unifiers as in the above algorithms, but all possible unifiers based on regular terms. Still, the number of the unifiers might be infinite. We overcome this problem in Chap. 5 by combining results from both minimal and regular unification algorithms in order to obtain a refutational complete higher-order resolution calculus which can use the minimal unification algorithms in a greedy way, of which perform better in many cases when compared to current calculi. A comparisons of the different calculi is also given in Chap. 5.

Outline

Theorem proving in higher-order logic is the focus of much research in the last decades. There are many approaches, like mating [2] and interactiveness [66], which were developed in order to overcome the different natural difficulties posed by higher-order problems. The greatest difficulty though, is the undecidability and infinitary nature of higher-order unification. Most of the research done in higher-order unification was with regard to different unification problems and not with regard to automated deduction in general and resolution theorem proving in particular. Because of that, many very sophisticated theorem provers, while implementing the most efficient data structures and search strategies, are still depending on Huet's pre-unification algorithm [49] in order to solve the unification constraints. Since the algorithm is non-terminating, these theorem provers are commonly utilizing incomplete termination procedures. In this thesis we analyze the different procedures which are being used in existing theorem provers as well as in theoretical ones. A main part of the thesis is devoted for the development of a new unification algorithm for higher-order logic which we believe is especially suitable for higher-order resolution

In Chap. 3 we develop the technique which allows us to enumerate all pre-unifiers of a problem in a certain way. Unlike traditional unification algorithms in which the enumeration is in a sense arbitrary, the unification algorithm which will be developed in this chapter will enumerate regular unifiers according to increasing size, a method which will be very suitable for higher-order resolution as we will see in Chap. 5. Beside giving a new way of enumerating all possible pre-unifiers, we will give in this chapter also a new proof for the decidability of the stratified context unification problem [70].

The method and algorithm obtained in Chap. 3 will be developed further in Chap. 4 and be extended to full higher-order logic. This will enable us to define a unification algorithm for higher-order logic, which closely relates to Huet's pre-unification algorithm [49]. The fact that this algorithm is based on the computation of regular terms and its unique way of enumerating unifiers will make it a suitable candidate for a resolution calculus.

In Chap. 5 we define a resolution calculus which is based on the unification algorithm from Chap. 4 and compare it to other resolution calculi, of which some are used in implemented theorem provers while other have not been implemented thus far.

Chap. 2 contains preliminary definitions and results which are required for the understanding of the rest of the chapters.

CHAPTER 2

Preliminaries

2.1 Typed Lambda Calculus

In this section we will present the logical language that will be used throughout the thesis. The language is a version of Church's simple theory of types [19] with an η-conversion rule as presented in [6] and [76] and with implicit α-conversions. Most of the definitions in this section are adapted from [76].

Definition 2.1.1 (Sets)**.** Sets are defined as usual as an unordered collection of elements (sets or others) as well as the set operations $\subset$, $\subseteq$, $\setminus$, $\cup$ and $\cap$. The empty set is denoted by $\emptyset$.

Definition 2.1.2 (Functions)**.** Functions are defined as usual as well as the functions composition operator $\circ$. We will detonate the domain of a function f by $\mathtt{dom}(f)$ and its co-domain by $\mathtt{co\text{-}dom}(f)$. The functions used in this thesis are all total.

Definition 2.1.3 (Tetrations [74])**.** We will define $2_m(n)$ as a special version of a tetration inductively as:

- $2_0(n) = n$
- $2_m(n) = 2^{2_{m-1}(n)}$

As a convenience, we will allow all the denotations appearing in this thesis to have numerical superscripts and subscripts.

Types and terms

Definition 2.1.4 (Types)**.** Let $\mathfrak{T}_0$ be a set of base types, then the set $\mathfrak{T}$ of all types is generated by the grammar:

$$\mathfrak{T} ::= \mathfrak{T}_0 \mid \mathfrak{T} \to \mathfrak{T} \tag{2.1}$$

We denote types using the lowercase Greek letters α, β and γ and let $\rightarrow$ associates to the right (i.e. $\alpha \rightarrow \beta \rightarrow \gamma$ is $\alpha \rightarrow (\beta \rightarrow \gamma)$). The number n in $\gamma = \alpha_1 \rightarrow \alpha_2 \rightarrow .. \rightarrow \alpha_n \rightarrow \beta$ where $\beta \in \mathfrak{T}_o$ is called the arity of γ and is denoted by $\mathtt{ar}(\gamma) = n$.

Before we define the set of all terms, we will make a distinction between the two types of symbols we have: constant symbols and variable symbols.

Definition 2.1.5 (Signature)**.** The signature for building higher-order terms is a set Σ of function symbols, such that for each $\alpha \in \mathfrak{T}$, we require Σ to contain (countably) infinitely many function symbols of type α. We denote function symbols using the letters f and g and denote the type α of a function symbol f by $\tau(f) = \alpha$. We further denote the arity of function symbols by $\mathtt{ar}(f) = \mathtt{ar}(\tau(f))$.

Definition 2.1.6 (Variables)**.** For every type $\alpha \in \mathfrak{T}$, we further assume the existence of a (countably) infinite set $\mathfrak{V}^\alpha$ of variables of type α. The union of all these sets is denoted by $\mathfrak{V}$. Variables are denoted using the letters x, y and z. Given a set of variables W, a fresh variable with regard to W is a variable not occurring in W. When W can be inferred from the context, we will omit it.

We sometimes write explicitly the type of function symbols and variables using superscripts as in f^α.

Definition 2.1.7 (Terms)**.** Terms are denoted by the letters t and s with their type sometimes written explicitly as a superscript. For every type $\alpha \in \mathfrak{T}$, we define the set $\mathtt{Term}^\alpha$ and the size ($\mathtt{size}$) of terms inductively:

- $f^\alpha \in \mathtt{Term}^\alpha$ and $\mathtt{size}(f^\alpha) = 1$.
- $x^\alpha \in \mathtt{Term}^\alpha$ and $\mathtt{size}(x^\alpha) = 1$.
- if $t \in \mathtt{Term}^{\beta \rightarrow \alpha}$ and $s \in \mathtt{Term}^\beta$ then $(ts) \in \mathtt{Term}^\alpha$ and $\mathtt{size}((ts)) = \mathtt{size}(t) + \mathtt{size}(s)$.
- if $t \in \mathtt{Term}^\gamma$, $x \in \mathfrak{V}^\beta$ and $\alpha = \beta \rightarrow \gamma$ then $\lambda x.t \in \mathtt{Term}^\alpha$ and $\mathtt{size}(\lambda x.t) = \mathtt{size}(t)$.

We extend τ to be a function over terms by $\tau(t^\alpha) = \alpha$. The set $\mathtt{Term}$ denotes the union of all $\mathtt{Term}^\alpha$.

Terms of the form (ts) are called applications while $\lambda x.t$ are called abstractions. When possible, we will flatten successive applications or abstractions. For example, $\lambda x_1.\lambda x_2.t$ is flattened to $\lambda x_1, x_2.t$ while $((t_1 t_2) t_3)$ is flattened to $(t_1 t_2 t_3)$. Note that applications associate to the left. When t_1 in the above application is a function symbol, we will also write it as $f(t_2, t_3)$. Flattened abstractions and functional applications will also be denoted using vector notation such as in $\lambda \overline{x_n}.t$ which denotes $\lambda x_1, .., x_n.t$ and $f(\overline{t_n})$ which denotes $f(t_1, .., t_n)$. If a function symbol f or a variable x are applied to only one argument t, then we will also denote this term by ft and xt respectively.

Definition 2.1.8 (Head symbols and flex terms)**.** The head symbol of a term is defined inductively:

- $\mathtt{hd}(f) = f$.
- $\mathtt{hd}(x) = x$.
- $\mathtt{hd}((ts)) = \mathtt{hd}(t)$.
- $\mathtt{hd}(\lambda x.t) = \mathtt{hd}(t)$.

A term whose head is a variable is called a flex term . It is called a rigid term otherwise.

Definition 2.1.9 (Positions and subterms)**.** Positions within terms are defined inductively where ϵ denotes the empty position:

- $t|_\epsilon = t$.
- if $t|_\rho = s$ and t_0, t_1 terms with the correct type, then $(tt_0)|_{1.\rho} = s$ and $(t_1 t)|_{2.\rho} = s$.
- if $t|_\rho = s$ then $(\lambda x.t)|_{1.\rho} = s$.

If there is a position ρ such that $t|_\rho = s$ then s is called a subterm of t.

In the algorithms discussed in this thesis, we would normally be interested only in a subset of the positions of a term.

Definition 2.1.10 (Rigid positions)**.** A position ρ is called a rigid position in a term t, if ρ is a position in t and for every prefix ρ' of ρ, $\mathtt{hd}(t|_{\rho'})$ is not a variable. Occurrences of subterms in t, whose positions are rigid, are called rigid occurrences. The set of all rigid positions in a term t is denoted by $\mathtt{rigid\text{-}pos}(t)$. The depth of a term t, denoted as $\mathtt{d}(t)$ is the size of the biggest position in $\mathtt{rigid\text{-}pos}(t)$.

Example 2.1.11. *Given the term $t = f(\lambda y.z(a, xy), b)$, the set $\mathtt{rigid\text{-}pos}(t) = \{1, 1.1, 1.2\}$.*

Definition 2.1.12 (Bound and free variables)**.** Given a term t, the sets of bound and free variables of t, denoted $\mathtt{BV}(t)$ and $\mathtt{FV}(t)$ respectively, are defined inductively:

- if $t = x$ then $\mathtt{BV}(t) = \emptyset$, $\mathtt{FV}(t) = \{x\}$.
- if $t = f$ and $\tau(f) \in \mathfrak{T}_0$ then $\mathtt{BV}(t) = \emptyset$, $\mathtt{FV}(t) = \emptyset$.
- if $t = (s_1 s_2)$ then $\mathtt{BV}(t) = \mathtt{BV}(s_1) \cup \mathtt{BV}(s_2)$, $\mathtt{FV}(t) = \mathtt{FV}(s_1) \cup \mathtt{FV}(s_2)$.
- if $t = \lambda x.s$ then $\mathtt{BV}(t) = \mathtt{BV}(s) \cup \{x\}$ and $\mathtt{FV}(t) = \mathtt{FV}(s) \setminus \{x\}$.

Definition 2.1.13 (Replacements)**.** Let t be a term and $p \in \mathtt{rigid\text{-}pos}(t)$, then $t[s|_p]$ is the term obtained by replacing the subterm at position p in t with s.

Given terms t and s, we would sometimes write $t[s|_p]$ in order to stress that s is a subterm of t at position p. These two overloading definitions of the $[|]$ operator refers always to the same element.

Substitutions

Definition 2.1.14 (Substitutions)**.** A substitution σ is a total function between variables and terms which fulfills the following two requirements:

- for all variables x in the domain of σ, $\tau(x) = \tau(\sigma(x))$.
- there is a finite set $V \subset \mathfrak{V}$, called the support of σ, such that $\forall x \in \mathfrak{V} \setminus V\ \sigma(x) = x$.

When we refer to the domain of a substitution σ, we will mean from now on the support of σ. We will sometimes denote a substitution explicitly as $\sigma = [t_1/x_1, .., t_n/x_n]$ where $\mathtt{size}(\mathtt{dom}(\sigma)) = n$, $x_i \in \mathtt{dom}(\sigma)$ and $t_i = \sigma(x_i)$ for all $0 < i \leq n$.

Definition 2.1.15 (Restrictions)**.** The restriction of a substitution σ to a set W, denoted by $\sigma|_W$, is the substitution σ' such that:

- if $x \in W$ then $\sigma'(x) = \sigma(x)$.
- otherwise $\sigma'(x) = x$.

Definition 2.1.16 (Extending substitutions)**.** Assume σ is a substitution and let t be a term, the function $\hat{\sigma} : \mathtt{Term} \to \mathtt{Term}$ extends σ and is defined inductively:

- if $t \in \mathfrak{V}$ then $\hat{\sigma}(t) = \sigma(t)$.
- if $t \in \Sigma$ then $\hat{\sigma}(t) = t$.
- if $t = (t_1 t_2)$ then $\hat{\sigma}(t) = (\hat{\sigma}(t_1)\hat{\sigma}(t_2))$.
- let σ_{-x} denotes $\sigma|_{\mathtt{dom}(\sigma)\setminus\{x\}}$ if $t = \lambda x.t_1$ then $\hat{\sigma}(t) = \lambda x.\hat{\sigma}_{-x}(t_1)$.

We normally denote the extension $\hat{\sigma}$ of σ by σ as well. We define the set of variables introduced by σ as $\mathfrak{I}(\sigma) = \cup_{x \in \mathtt{dom}(\sigma)} \mathtt{FV}(x\sigma)$. The notion of applying a substitution is extended to sets of terms by applying the substitution to each term in the set. We also write the application of a substitution σ to a term t by $t\sigma$. Note that we write the application of the extending substitution to terms using postfix notation while the application of substitutions to variables is written in prefix notation.

Remark 2.1.17. In Remark 2.1.23 we adopt the convention that the two sets of variables FV and BV are disjoint. Under this convention σ in Def. 2.1.16 is always equal to σ_{-x}.

Later in the thesis it will be convenient to compare terms which are instances of one another.

Definition 2.1.18 (Subsumption)**.** Let t_1 and t_2 be terms, then we say that t_1 subsumes t_2 and denotes it by $t_1 \leq_s t_2$ if there is a substitution σ such that $t_1\sigma = t_2$. Let S be a set of terms and t a term, then t is subsumed by the set S if there is a term t' in S which subsumes t. We denote this by $t \in_s S$.

Example 2.1.19. *Let $t_1 = \lambda z.f(x, z)$ and $t_2 = \lambda z.f(a, z)$, then we have $t_1 \leq_s t_2$ and $t_2 \in_s \{t_1\}$.*

Reductions and normal forms

Definition 2.1.20 (Lambda calculus rules)**.** The lambda calculus has the following three rules:

- if $y \notin \mathtt{FV}(t) \cup \mathtt{BV}(t)$ then $(\lambda x.t) \succ_\alpha (\lambda y.t[y/x])$ (α-rule).
- $((\lambda x.s)t) \succ_\beta s[t/x]$ (β-rule).
- $(\lambda x.(tx)) \succ_\eta t$ (η-rule).

Definition 2.1.21 (Reductions)**.** Let t be a term and let $\bullet$ be either α, β, η or $\beta\eta$, then a $\bullet$-reduct of t is a term $t[s'|_p]$ such that $t|_p = s$ and $s \succ_\bullet s'$. We denote a $\bullet$-reduction by $t[s] \to_\bullet t[s']$. We denote the reflexive, symmetric and transitive closure of $\to_\bullet$ by $=_\bullet$. The reflexive and transitive closure of $\to_\bullet$ is denoted by $\overset{*}{\to}_\bullet$.

Definition 2.1.22 (α-equality)**.** Given two terms t_1 and t_2, we say that they are α-equal if $t_1 =_\alpha t_2$.

Remark 2.1.23. In the rest of the thesis, equality between terms will mean α-equality. In our construction of terms, free variables may become bound but not vice-versa. We would like to reserve the set $\mathfrak{V}$ to denote the set of free variables only and therefore we will keep an implicit (countable) infinite set for the bound variables, which is disjoint from $\mathfrak{V}$. Another implicit operation that will take place when abstracting over a term will be to replace the newly bounded variable with a fresh variable from the set of bound variables such that the two terms will be α-equal. Since we will assume equality between terms to denote α-equality all along the thesis, this (implicit) operation is valid. This will allow us to abstract over the notion of α-equality and therefore, to simplify our presentation. This will also allow us to avoid the issue of free variables capture as binding free variables will cause them to be replaced by fresh bound variables.

Definition 2.1.24 (Normal forms)**.** Let t be a term and let $\bullet$ be either β, η or $\beta\eta$, if t has no $\bullet$-reducts, then t is said to be in $\bullet$-normal form.

From the definition of a replacement it follows that the type of terms is preserved under the reduction rules.

Remark 2.1.25. The η-reduction rule can be considered as a weak form of the axiom of functional extensionality, which asserts that two functions are equal if they behave the same on all arguments. We will discuss the role of the extensionality axiom when we discuss the semantics of the resolution calculus later in this chapter.

Two major results about the typed lambda calculus (see [6]) are:

Theorem 2.1.26 (Strong normalization)**.** Every sequence of $\beta\eta$-reductions is finite.

Theorem 2.1.27 (Church-Rosser theorem)**.** if $t_1 =_{\beta\eta} t_2$, then there is a term s such that $t_1 \overset{*}{\to}_{\beta\eta} s \overset{*}{\leftarrow}_{\beta\eta} t_2$.

It follows from these two results that the rules are confluent, i.e. for each term there exists a unique normal form. This in turn implies that the equivalence between two terms can be decided by a syntactic comparison of their normal forms.

Remark 2.1.28. From now on we will assume (unless otherwise specified) that a given term is in β normal form. We will explicitly denote the β normal form of a term t by $t\downarrow$. The η-normal form of a term t will be denoted by $t\downarrow_\eta$.

Definition 2.1.29 (Order over substitutions)**.** Given a set W of variables and two substitutions σ and θ, we say that σ is equal to θ over W, denoted $\sigma =^{|_W} \theta$ if $\forall x \in W\ \sigma(x) = \theta(x)$. We say that σ is more general than θ over W, denoted $\sigma \leq^{|_W} \theta$, if there exists a substitution δ such that $\theta =_{|_W} \sigma \circ \delta$. When $W = \mathfrak{V}$, we drop the notation $|_W$. $=_\beta$, $=_{\beta\eta}$, $\leq_\beta$ and $\leq_{\beta\eta}$ are defined analogously.

Lemma 2.1.30. For substitutions σ and θ, if $\sigma =_\beta \theta$, then for every term t, $t\sigma =_\beta t\theta$. The same holds for $=_{\beta\eta}$.

Definition 2.1.31 (Idempotency)**.** A substitution σ is idempotent if $\sigma \circ \sigma =_{\beta\eta} \sigma$.

Lemma 2.1.32. A substitution σ is idempotent if $\mathfrak{I}(\sigma) \cap \mathtt{dom}(\sigma) = \emptyset$.

Lemma 2.1.33 ([77])**.** For any substitution σ and a set of variables W containing $\mathtt{dom}(\sigma)$, there exists an idempotent substitution θ such that $\mathtt{dom}(\sigma) = \mathtt{dom}(\theta)$, $\sigma \leq_{\beta\eta} \theta$ and $\theta \leq_{\beta\eta}^{|_W} \sigma$.

Remark 2.1.34. In the rest of the thesis we assume all substitutions to be idempotent.

Implicit treatment of extensionality

Next we will follow the results in [76] which allow us to abstract also over the η-rule and consider only β-reductions and normal forms.

Definition 2.1.35 (η-expanded forms)**.** We first define the following rule, which is the converse of the η-rule:

- $t^{\alpha\to\beta} \succ_{\overline{\eta}} \lambda x^\alpha.tx$ ($\overline{\eta}$-rule).

and define the η-expanded form as the normal form obtained under the application of this rule in a similar way to the η-normal form in definitions 2.1.21 and 2.1.24. The η-expanded form of a term t is denoted by $\eta[t]$.

Lemma 2.1.36 ([6])**.** For any two terms t_1 and t_2, we have $t_1 \overset{*}{\to}_{\beta\eta} t_2$ iff there exists a term s such that $t_1 \overset{*}{\to}_\beta s \overset{*}{\to}_\eta t_2$.

From this lemma and the uniqueness of the η-expanded form it follows the following:

Theorem 2.1.37. For every two terms t and s, we have $t =_{\beta\eta} s$ iff $\eta[t\downarrow] = \eta[s\downarrow]$.

We can now show that by considering only terms in η-expanded form, we can leave out the η-conversion rule.

Definition 2.1.38 (η-expanded terms)**.** Let $\texttt{Term}_{exp} = \{\eta[t] \mid t \in \texttt{Term}\}$ and let $\texttt{Term}_\eta$ be the minimal set containing $\texttt{Term}_{exp}$ and closed under application and abstraction.

Lemma 2.1.39 ([47])**.** The following closure properties hold for $\texttt{Term}_\eta$:

- if $t, s \in \texttt{Term}_{exp}$ then $(\lambda x.t) \in \texttt{Term}_{exp}$ and $(ts) \downarrow\in \texttt{Term}_{exp}$.
- if $t \in \texttt{Term}_\eta$ then $t \downarrow\in \texttt{Term}_{exp}$.
- if $t, s \in \texttt{Term}_\eta$ then $(\lambda x.t) \in \texttt{Term}_\eta$ and $(ts) \in \texttt{Term}_\eta$.
- if $t \in \texttt{Term}_\eta$ and $t \overset{*}{\rightarrow}_\beta s$ then $s \in \texttt{Term}_\eta$.
- if $t, s \in \texttt{Term}_\eta$ then $s[t/x] \in \texttt{Term}_\eta$.

Definition 2.1.40 (Normalized substitutions)**.** A substitution σ is said to be normalized if $\forall x \in \texttt{dom}(\sigma)\ \sigma(x) \in \texttt{Term}_{exp}$.

The following corollary, which is based on Lemma 2.1.39, concludes our discussion about the implicit treatment of η-conversions and allows us to assume that all the terms dealt with and produced by the algorithms in this thesis are in η-expanded form.

Corollary 2.1.41. If σ is a normalized substitution and $t \in \texttt{Term}_{exp}$, then $t\sigma \in \texttt{Term}_\eta$ and $t\sigma \downarrow\in \texttt{Term}_{exp}$.

Remark 2.1.42. In the rest of the thesis, all substitutions are assumed to be normalized.

Remark 2.1.43. In some cases, such as in solved forms which will be defined later, it is more convenient to denote variables in their η-normal form instead of in their expanded forms. For example, we would denote the variable $x^{\alpha\rightarrow\beta}$ by x instead of by $\lambda z^\alpha.xz$.

First-order terms and contexts

In later sections we will sometimes be concerned with only a subset of all lambda calculus generated terms. In this section we will present some definitions which will allow us to discuss these restricted terms.

Definition 2.1.44 (Order of terms)**.** The order of a term is defined to be the order of its type. The order of a type α is defined inductively:

- if $\alpha \in \mathfrak{T}_0$ then $\texttt{ord}(\alpha) = 1$.
- if $\alpha = \beta \rightarrow \gamma$ then $\texttt{ord}(\alpha) = \texttt{max}(\texttt{ord}(\beta) + 1, \texttt{ord}(\gamma))$.

A language of order n is a language which contains function symbols of order $n+1$ and variables of order n.

When discussing first-order terms, we will assume that the set $\mathfrak{T}_0$ of basic types contains the type *i* for individuals only. We will further call function symbols of type *i* constants.

The set $\texttt{Term}_i$ of first-order terms can also be defined directly.

Definition 2.1.45 (First-order terms)**.** The set $\texttt{Term}_i$ of first-order terms is defined inductively:

- if $\tau(t) = i$ then $t \in \texttt{Term}_i$
- if $\tau(f) = i \to .. \to i \to i$, $\texttt{ar}(f) = n > 0$ and $t_i \in \texttt{Term}_i$ for all $0 < i \leq n$ then $f(t_1, .., t_n) \in \texttt{Term}_i$.

Next we will define contexts, which are abstractions of type $\alpha \to \beta$ for $\alpha, \beta \in \mathfrak{T}$ containing exactly one occurrence of the bound variable in the abstracted term. Contexts play a central role in the unification algorithms presented in this thesis. In order to simplify the algorithms and the proofs, we will use a distinct notation for these terms.

Definition 2.1.46 (Contexts)**.** Given a signature Σ, let $\Sigma_c = \Sigma \cup \{[.]^\alpha \mid \alpha \in \mathfrak{T}\}$. The symbol $[.]^\alpha$ is a new symbol of type α and is called a hole of type α. The set $\texttt{Context}_\alpha$ over the signature Σ_c of contexts with holes of type α is not so easily defined inductively as such a definition will require the replacement of subterms. Since this definition is only a syntactic-sugar, we will give a descriptive definition instead:

- if $t[s] \in \texttt{Term}$ and $s \in \texttt{Term}^\alpha$ a subterm occurrence of t, then $t[.] \in \texttt{Context}_\alpha$.

The position ρ of $[.]$ in C is called the main path of the context C and is denoted by $\texttt{mpath}(C)$. The size of the position $\texttt{mpath}$ is called the main depth of the context and is denoted by $\texttt{mdepth}(C)$. We normally denote contexts by C. Let ρ be the main path of a context C and ρ' any prefix of that position, then the context $C[[.]|_{\rho'}]$ is called a prefix of C. Contexts whose type is the same as the type of the hole (i.e. $\tau(C) = \tau(C|_{\texttt{mpath}(C)})$) are called simple contexts. Otherwise, we call them complex contexts.

Example 2.1.47. *An example of a context is $f(\lambda z.g([.]^i, z), h(x))$ whose main path is the position* 1.1. *This context is simple as it has the same type as the hole.*

The context $(t[.]^\alpha)^\beta$ denotes the term $\lambda x^\alpha.t[x] \in \texttt{Term}^{\alpha \to \beta}$. More formally, we define an homomorphism from contexts to terms and when we speak about contexts, we refer to their homomorphic terms.

Definition 2.1.48 (Contexts homomorphism)**.** Let $t \in \texttt{Term}$ be a term with position p and $t[[.]^\alpha|_p]$ a context, then the homomorphism $\mathfrak{H} : \texttt{Context} \to \texttt{Term}$ is $\mathfrak{H}(t[[.]|_p] = \lambda x^\alpha.t[x|_p]$ for $x \notin \texttt{FV}(t) \cup \texttt{BV}(t)$.

Example 2.1.49. *the homomorphic term for the context $f(\lambda x.g(x, x), [.])$ is (up to α-equivalence) $\lambda y.f(\lambda x.g(x, x), y)$.*

Remark 2.1.50. When we would like to stress that a certain argument t of a function symbol f is at position k, we would sometimes write $f(.., t_{@k}, ..)$

As with first-order terms, also first-order contexts can be defined directly.

Definition 2.1.51 (First-order contexts)**.** The set $\texttt{Context}_i$ of first-order contexts is defined inductively:

- $[.]_i \in \texttt{Context}_i$
- if $\tau(f) = i \rightarrow .. \rightarrow i \rightarrow i$, $\texttt{ar}(f) = n > 0$, $t_i \in \texttt{Term}_i$ for all $0 < i \leq n$, one $0 < j \neq i \leq n$ and $C \in \texttt{Context}_i$ for $0 < j \leq n$ then $f(t_1, .., C_{@j}, .., t_n) \in \texttt{Context}_i$.

An important characteristic of first-order contexts is that they are always simple.

Definition 2.1.52 (Composition of contexts)**.** If $C \in \texttt{Context}_\alpha$ and $s \in \texttt{Term}_\alpha$ then we denote by $C(s)$ the term $t' \in \texttt{Term}$ which is obtained by replacing the hole with s. Note that this operation is compatible with application and β-reduction on their homomorphic terms. We will also write the application of a context on a term without parentheses as in Cs. Given a context C, if the main path of C is equal to ϵ then C is called a trivial context and non-trivial otherwise. If C is a simple context, we define the composition of C iterated n times by $C^0 = C$ and $C^{n+1} = C(C^n)$ for $n \geq 0$. Note that if $C \in \texttt{Context}_\alpha$ then also $C^n \in \texttt{Context}_\alpha$.

Remark 2.1.53. When applying several contexts of monadic symbols or variables, we will drop the parentheses and associate them to the right. For example CX_1a means $C(X_1a)$.

Logical symbols

The main aim of this thesis is to give a method for the refutation of higher-order falsities. In order to achieve that aim, we will require our signature to contain some logical symbols with pre-defined semantics.

Definition 2.1.54 (Boolean terms)**.** Terms of boolean type, which is denoted by o, can be either true or false. In the remaining of this thesis we will require the set of basic types to contain the boolean type and the signature to contain the two constants $\texttt{T}$ and $\texttt{F}$, denoting true and false respectively.

Definition 2.1.55 (Logical connectives)**.** As well as truth values, we will require the signature to contain the following symbols:

- $\neg$ of type $o \rightarrow o$.
- $\vee$ of type $o \rightarrow o \rightarrow o$.
- Π_α of type $(\alpha \rightarrow o) \rightarrow o$ for all $\alpha \in \mathfrak{T}$.

Remark 2.1.56. The set of logical connectives given above is sufficient to define all other logical connectives:

- $A \wedge B$ can be denoted by $\neg(\neg A \vee \neg B)$.
- $\forall x^\alpha . f^{\alpha \rightarrow o} x$ can be denoted by $\Pi_\alpha(\lambda x.fx)$ and
- $\exists x^\alpha . f^{\alpha \rightarrow o} x$ can be denoted by $\neg \forall x . \neg(fx)$.

In the remaining of the thesis we might use all logical connectives but will prove the results over the three defined in Def. 2.1.55 only.

Remark 2.1.57. From now on we will consider the symbol $\doteq_\alpha$ to be of type $\alpha \rightarrow \alpha \rightarrow o$.

2.2 Pre-unification

Pre-unification Problems

In this section we will describe a higher-order unification algorithm [49] which is the main unification algorithm used in higher-order automated deduction systems. The definitions presented here are adapted from [76].

Definition 2.2.1 (Unification constraints)**.** Given a signature Σ, let $\Sigma_u = \Sigma \cup \{\doteq^\alpha \mid \alpha \in \mathfrak{T}\}$. A unification constraint (or just constraint) is a term over Σ_u whose head symbol is $\doteq^\alpha$ and the symbol $\doteq^\alpha$ does not occur elsewhere in the term. We will write unification constraints in infix notation and drop the typing labels when they can be inferred from the context. A set of unification constraints is called a unification system or just system. We will denote the set of free variables of a system S by $\mathtt{FV}(S)$. A unification constraint is called rigid-rigid if both immediate subterms in it are rigid. It is called flex-rigid if one is rigid only and flex-flex if none of them is rigid. A unification system will always be closed under the symmetry of $\doteq$. I.e. whenever a term $t \doteq s$ is in the problem, then so is $s \doteq t$.

Definition 2.2.2 (Solved forms)**.** Let S be a system and $x \doteq s \in S$ be a unification constraint in η-normal form. Then, $x \doteq s$ is in solved form in system S if x does not occur elsewhere in S. x is called a solved variable. A system is in solved form if all its constraints are in solved form.

Definition 2.2.3 (Pre-solved forms)**.** A unification constraint $\lambda\overline{z_n}.x(\overline{t_m}) \doteq s$ is in pre-solved form if it is either in solved form or s is a flexible term. A system is in pre-solved form if all its constraints are in pre-solved form. Given a system S in pre-solved form, we denote by σ_S the substitution whose domain consists all the solved variables in S such that $\sigma_S(x) = t$ if $x \doteq t$ is a solved constrained in S.

Example 2.2.4. *The first system is in solved form while the second one is in pre-solved form:*

1. $\{x \doteq f(a, y), z \doteq g(y)\}$
2. $\{x \doteq f(z, y), z \doteq y\}$

Definition 2.2.5 (Unifiers)**.** A substitution σ is a unifier of a unification constraint $t \doteq s$ if $t\sigma = s\sigma$ and it is a unifier of a unification system if it unifies all unification constraints in it. Given a unification system S, we denote the set of all its unifiers by $\mathtt{Unifiers}(S)$.

In general, the set of all unifiers of the problem might be too big for an efficient enumeration by an algorithm. Huet's well-known solution for this was to search for pre-unifiers instead of unifiers.

Definition 2.2.6 (Pre-unifiers)**.** Let $\stackrel{\sim}{=}$ be the least congruence relation on $\mathtt{Term}$ which contains $\{(t, s) \mid \mathtt{hd}(t), \mathtt{hd}(s) \in \mathfrak{V}\}$. A substitution σ is a pre-unifier of a constraint $t \doteq s$ if $t\sigma \stackrel{\sim}{=} s\sigma$. It is a pre-unifier of a system if it pre-unifies all its constraints.

Example 2.2.7. *The substitution $[f(a, y)/x, g(y)/z]$ is a unifier of the first system from Ex. 2.2.4 while $[f(a, y)/x]$ is a pre-unifier of the second system.*

The search for pre-unifiers is justified by the following.

Definition 2.2.8 (Completing substitution)**.** For every $\alpha \in \mathfrak{T}$ such that $\alpha = \beta_1 \to .. \to \beta_n \to \gamma$ let $\widehat{e}_\alpha = \lambda x_1^{\beta_1}..x_n^{\beta_n}.y^\gamma$ such that y is a fresh variable (with regard to the current unification system). Let V be a finite set of variables, then the completing substitution is the substitution $\xi^V = [\widehat{e}_{\tau(x)}/x \mid x \in V]$. Let S be a system in pre-solved form and S' be the set of all unsolved equations in S, then $\xi_S = \xi^{\mathrm{FV}(S')}$.

Lemma 2.2.9. If S is a system in pre-solved form, then $\sigma_S \circ \xi_S$ is a unifier of S.

Example 2.2.10. *Let the systems in Ex. 2.2.4 be first-order systems, then* $\xi = [w/x, w/y, w/z]$ *and* $[f(a, y)/x] \circ \xi = [f(a, w)/x, w/y, w/z]$ *is a unifier of the second system.*

A useful notion in automated deduction is the notion of most general unifiers.

Definition 2.2.11 (Most general unifiers)**.** Given a system S, a substitution σ is called a most general unifier of S if σ is a unifier of S and for every unifier θ of S, $\sigma \leq \theta$ (see Def. 2.1.29).

It is well known that any unifiable first-order system has a most general unifier. It is also known that unifiable higher-order systems may have no most-general unifier [39].

The following notion extends that of most general unifiers for higher-order systems.

Definition 2.2.12 (Complete sets of pre-unifiers)**.** Given a unification system S, its complete set of pre-unifiers is the set $\mathtt{PreUnifiers}(S)$ of (normalized) substitutions such that:

- $\{\sigma \circ \xi_S \mid \sigma \in \mathtt{PreUnifiers}(S)\} \subseteq \mathtt{Unifiers}(S)$.
- for every normalized $\theta \in \mathtt{Unifiers}(S)$ there exists $\sigma \in \mathtt{PreUnifiers}(S)$ such that $\sigma|_{dom(\theta)} \leq \theta$.

Definition 2.2.13 (Partial bindings)**.** A partial binding of type $\alpha_1 \to .. \to \alpha_n \to \beta$ where $\beta \in \mathfrak{T}_o$ is a term of the form
$\lambda\overline{y_n}.a(\lambda\overline{z^1_{p_1}}.x_1(\overline{y_n}, \overline{z^1_{p_1}}), .., \lambda\overline{z^m_{p_m}}.x_m(\overline{y_n}, \overline{z^m_{p_m}}))$ for some atom a where

- $\tau(y_i) = \alpha_i$ for $0 < i \leq n$.
- $\tau(a) = \gamma_1 \to .. \to \gamma_m \to \beta$ where $\gamma_i = \delta^i_1 \to .. \to \delta^i_{p_i} \to \gamma'_i$ for $0 < i \leq m$.
- $\tau(z^i_j) = \delta^i_j$ for $0 < i \leq m$ and $0 < j \leq p_i$.
- x_i is a fresh variable and $\tau(x_i) = \alpha_1 \to .. \to \alpha_n \to \delta^i_1 \to .. \to \delta^i_{p_i} \to \gamma'_i$ for $0 < i \leq m$.
- $\gamma'_1, .., \gamma'_m \in \mathfrak{T}_o$.

Partial bindings fall into two categories, imitation bindings, which for a given atom a and type α, are denoted by $\mathtt{PB}(a, \alpha)$ and projection bindings, which for a given index $0 < i \leq n$ and a type α, are denoted by $\mathtt{PB}(i, \alpha)$ and in which the atom a is equal to the bound variable y_i. Since partial bindings are uniquely determined by a type and an atom (up to renaming of the fresh variables $\overline{x_m}$), this defines a particular term.

Example 2.2.14. *We will compute the set of partial bindings for the second-order equation* $x^{\mathrm{i}\to\mathrm{i}}(b^{\mathrm{i}}) \doteq f^{\mathrm{i}\to\mathrm{i}\to\mathrm{i}}(a^{\mathrm{i}}, y^{\mathrm{i}})$.

- $\mathit{PB}(f, \mathrm{i}\to\mathrm{i}) = \lambda x_0^{\mathrm{i}}.f(y_1^{\mathrm{i}\to\mathrm{i}}(x_0), y_2^{\mathrm{i}\to\mathrm{i}}(x_0))$.
- $\mathit{PB}(1, \mathrm{i}\to\mathrm{i}) = \lambda x_0^{\mathrm{i}}.x_0$.

where y_1 *and* y_2 *are fresh free variables.*

Definition 2.2.15 (The set of rules `PUA` (Pre-unification algorithm))**.** Let S be a unification system, then the set of rules `PUA` is defined in Fig. 2.1. The rules `(Imitate)` and `(Project)` are always followed by a `(Bind)`. Note that in the conclusions of `(Bind)`, `(Imitate)` and `(Project)` we write x in η-normal form. The reason for that is that as these new equations are in solved form we would like to stress the fact they are also a part of any pre-unifier of the system. This is the only place in the thesis where we make this exception.

$$\frac{S \cup \{A \doteq A\}}{S}\ \texttt{(Delete)} \qquad \frac{S \cup \{\lambda\overline{z_k}.f(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_n})\}}{S \cup \{\lambda\overline{z_k}.s_1 \doteq \lambda\overline{z_k}.t_1, .., \lambda\overline{z_k}.s_n \doteq \lambda\overline{z_k}.t_n\}}\ \texttt{(Decomp)}$$

$$\frac{S \cup \{\lambda\overline{z_k}.x(\overline{z_k}) \doteq \lambda\overline{z_k}.t\} \qquad x \notin \mathtt{FV}(t), \sigma = [\lambda\overline{z_k}.t/x]}{\sigma(S) \cup \{x \doteq \lambda\overline{z_k}.t\}}\ \texttt{(Bind)}$$

$$\frac{S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m}) \in S, u = \mathtt{PB}(f, \alpha)}{S \cup \{x \doteq u\}}\ \texttt{(Imitate)}$$

$$\frac{S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m}) \in S, 0 < i \leq n, u = \mathtt{PB}(i, \alpha)}{S \cup \{x \doteq u\}}\ \texttt{(Project)}^{1}$$

1. a can be either a function symbol or a bound variable z_i for $0 < i \leq k$.

Figure 2.1: `PUA` - Huet's Pre-Unification Rules

Theorem 2.2.16 (Soundness)**.** If S' is obtained from a unification system S using `PUA` and is in pre-solved form, then $\sigma_{S'}|_{\mathtt{FV}(S)} \in \mathtt{PreUnifiers}(S)$.

Theorem 2.2.17 (Completeness)**.** If θ is a pre-unifier of a unification system S, then there exists a pre-solved system S', which is obtainable from S using `PUA` such that $\sigma_{S'}|_{\mathtt{FV}(S)} \leq \theta$.

Remark 2.2.18. Another important result about `PUA` is that the only non-determinism that affects the form of the obtainable systems is the choice of applying either `(Imitate)` or `(Project)`. The other sources of non-determinism, such as the choice of which constraint to process next, do not affect the form of the obtained system (up to variables renaming).

Cycles

The notion of cyclic sets of equations plays an important role in this thesis and is proved later to be the only source for the existence of infinitely-many pre-unifiers.

Definition 2.2.19 (Cycles)**.** Let S be a unification system, then a sequence of constraints $s_1 \doteq t_1..s_n \doteq t_n$ in S is called a cycle if:

- $\mathtt{hd}(s_i) = x_i \in \mathfrak{V}$ for $0 < i \leq n$.
- for all $0 < i \leq n$ there is a position $p_i \in \mathtt{rigid\text{-}pos}(t_i)$ such that $\mathtt{hd}(t_i|_{p_i}) = x_{(i \ mod \ n)+1}$
- there is an index $0 < j \leq n$ such that $p_j \neq \epsilon$.

We denote the fact that the variables x_i occur in the cycle c by $x_i \in c$.

Example 2.2.20. *Cycles can be explicitly denoted using positions as in the following example:*

- $\{x_1(s_1) \doteq t_1[x_2(s'_1)|_{p_1}], x_2(s_2) \doteq t_2[x_1(s'_2)|_{p_2}]\}$ *for some positions* p_1 *and* p_2.

When the positions are not important (but also not empty), we will drop them and denote cycles just as:

- $\{x_1(s_1) \doteq t_1[x_2(s'_1)], x_2(s_2) \doteq t_2[x_1(s'_2)]\}$

Definition 2.2.21 (Standard cycles)**.** Let c be the cycle $s_1 \doteq t_1..s_n \doteq t_n$ and let x_i and p_i for $0 < i \leq n$ be defined as in the previous definition, then c is called a standard cycle if there is exactly one index $0 < j \leq n$ and at least one position p_j such that $\mathtt{hd}(t_j|_{p_j}) = x_{(j \ mod \ n)+1}$ and $p_j \neq \epsilon$.

Definition 2.2.22 (Unique standard cycles)**.** Let c be a standard cycle $s_1 \doteq t_1..s_n \doteq t_n$ and let x_i and p_i for $0 < i \leq n$ be defined as in the two previous definitions, then c is called a unique standard cycle if there is exactly one position p_j such that $t_j|_{p_j} = x_{(j \ mod \ n)+1}$ and $p_j \neq \epsilon$. We call the context $t_j[[.]|_{p_j}]$ the cycle context and denote it by $\mathtt{ccon}(c)$. The cycle size is defined to be n and is denoted by $\mathtt{size}(c) = n$. A standard cycle which is not unique is called a non-unique standard cycle. Since we will be mainly interested in unique standard cycle, we will call them standard cycles and will write non-unique standard cycles explicitly when referring to a non-unique one.

Example 2.2.23. *The cycle in Ex. 2.2.20 is neither a unique standard cycle nor non-unique. This is so because both positions* p_1 *and* p_2 *are not empty. Clearly a unique standard cycle is always a non-unique standard cycle as well. The following cycle is a non-unique standard cycle:*

- $\{x_1(s_1) \doteq x_2(s'_1), x_2(s_2) \doteq f(x_1(s'_2), x_1(s''_2))\}$

The following cycle is a unique standard one:

- $\{x_1(s_1) \doteq x_2(s'_1), x_2(s_2) \doteq f(x_1(s'_2), a)\}$

Note that (unique) standard cycles can be denoted using contexts, which will be useful in the next chapters:

- $\{x_1(s_1) \doteq x_2(s'_1), x_2(s_2) \doteq f([.], a)(x_1(s'_2))\}$

2.3 Constrained Resolution

In this section we will describe the constrained resolution calculus ([48]) as presented in [16] and [7].

Full automation of the search for proofs of first-order theorems was first introduced with the resolution method ([68]). A key concept in the method is the first-order unification principle. Unification allows the algorithm to apply cuts and contractions on non-equal terms by obtaining most general unifiers of these terms. The unification of first-order terms was shown to be unary and terminating and therefore, the search for a most general unifier could always be done eagerly.

When we consider the higher-order unification algorithm `PUA` from Def. 2.2.15, we see that it has two characteristics which may make a fully-automated search for proofs impossible. The first one is the undecidability of the unifiability question. When attempting to apply a cut or a contraction on two non-equal terms, the question whether there is a substitution that can make them equal is undecidable. Even if we can determine that the two terms can be made equal, the complete set of pre-unifiers (see Def. 2.2.12) might be infinite, which renders a fully-automated search impractical.

The constrained resolution calculus helps solving the second problem. By delaying the computation of the complete set of pre-unifiers, we have a finite representation of all possible unifiers in the form of the unification system itself. These unification systems are carried along the proof until we wish to check if they are unifiable. In this way, the unification component of the proof search is restricted to the question of unifiability only. A major drawback of this though is the increased search space, which includes searches for non-unifiable terms as well.

The calculus presented in this section will be used as the basis of the calculi presented in Chap. 5. Of which one integrates the unification algorithm from Chap. 4 and applies unification eagerly.

Definition 2.3.1 (Literals)**.** Pseudo literals are terms of type o which are labeled by either true or false, which represent their intended truth value and are denoted by $[A]^\mu$ for term A of type o and $\mu \in \{\mathtt{T}, \mathtt{F}\}$. If A does not contain logical symbols, then A is called a literal.

Remark 2.3.2. Since terms of type o play an important role in resolution, we will denote set variables, which are variables of type $\alpha \to o$ for $\alpha \in \mathfrak{T}$, using capital letters X, Y etc. We will use this denotation in this section and in Chap. 5.

Definition 2.3.3 (Unification constraints)**.** Unification constraints are literals whose head is $\doteq$ and which are labeled by $\mathtt{F}$.

Example 2.3.4. *The literal* $[f^{\mathrm{i}\to\mathrm{o}}a]^T$ *is not a unification constraint while the literal* $[x(a, y) \doteq f(g(x(a, b)), y)]^F$ *is a unification constraint.*

Definition 2.3.5 (Clauses)**.** Clauses are disjunctions of pseudo literals.

Example 2.3.6. *The following is a clause:*

$$[f^{\mathrm{i}\to\mathrm{o}}a]^T \vee [x(a,y) \doteq f(g(x(a,b)),y)]^F \tag{2.2}$$

In Fig. 2.2 we present the pre-unification rules from Def. 2.2.15 in the form of clauses.

$$\frac{S \vee [A \doteq A]^{\mathrm{F}}}{S}\ \texttt{(Delete)} \qquad \frac{S \vee [\lambda\overline{z_k}.f(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_n})]^{\mathrm{F}}}{S \vee [\lambda\overline{z_k}.s_1 \doteq \lambda\overline{z_k}.t_1]^{\mathrm{F}} \vee .. \vee [\lambda\overline{z_k}.s_n \doteq \lambda\overline{z_k}.t_n]^{\mathrm{F}}}\ \texttt{(Decomp)}$$

$$\frac{S \vee [\lambda\overline{z_k}.x(\overline{z_k}) \doteq \lambda\overline{z_k}.t]^{\mathrm{F}} \qquad x \notin \texttt{FV}(t), \sigma = [\lambda\overline{z_k}.t/x]}{\sigma(S) \vee [x \doteq \lambda\overline{z_k}.t]^{\mathrm{F}}}\ \texttt{(Bind)}$$

$$\frac{S \qquad [\lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m})]^{\mathrm{F}} \in S, u = \texttt{PB}(f,\alpha)}{S \vee [x \doteq u]^{\mathrm{F}}}\ \texttt{(Imitate)}$$

$$\frac{S \qquad [\lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m})]^{\mathrm{F}} \in S, 0 < i \leq n, u = \texttt{PB}(i,\alpha)}{S \vee [x \doteq u]^{\mathrm{F}}}\ \texttt{(Project)}^1$$

1. a can be either a function symbol or a bound variable z_i for $0 < i \leq k$.

Figure 2.2: `PUA` - Huet's Pre-Unification Rules

An important aspect of clause normalization is Skolemization. We will use the Skolem terms defined in [62]

Definition 2.3.7 (Skolemization)**.** Given a clause C, let $x_1^{\alpha_1},..,x_n^{\alpha_n}$ be the set of all free variables occurring in C, then a Skolem term of type α for C, which will be denoted by s_α is the term $f(x_1,..,x_n)$ for f a new function symbol of type $\alpha_1 \to .. \to \alpha_n \to \alpha$.

The constrained resolution calculus is based on literals and clauses. Therefore, it is necessary to have rules for the normalization of terms into clauses.

Definition 2.3.8 (Simplification rules)**.** The set of simplification rules, which are used for normalizing terms into clauses, is given in Fig. 2.3 where x is a new variable and s_α is a new Skolem term.

$$\frac{C \vee [\neg D]^{\mathrm{T}}}{C \vee [D]^{\mathrm{F}}}\ (\neg^T) \qquad \frac{C \vee [\neg D]^{\mathrm{F}}}{C \vee [D]^{\mathrm{T}}}\ (\neg^F)$$

$$\frac{C \vee [D_1 \vee D_1]^{\mathrm{T}}}{C \vee [D_1]^{\mathrm{T}} \vee [D_2]^{\mathrm{T}}}\ (\vee^T) \quad \frac{C \vee [D_1 \vee D_2]^{\mathrm{F}}}{C \vee [D_1]^{\mathrm{F}}}\ (\vee_l^F) \quad \frac{C \vee [D_1 \vee D_2]^{\mathrm{F}}}{C \vee [D_2]^{\mathrm{F}}}\ (\vee_r^F)$$

$$\frac{C \vee [\Pi_\alpha A]^{\mathrm{T}}}{C \vee [Ax^\alpha]^{\mathrm{T}}}\ (\Pi^T)^1 \qquad \frac{C \vee [\Pi_\alpha A]^{\mathrm{F}}}{C \vee [As_\alpha]^{\mathrm{T}}}\ (\Pi^F)^2$$

Figure 2.3: Simplification Rules

The resolution and factorization rules, given next, correspond to cuts and contractions over terms which are not syntactically equal and their correctness is based on the unifiability of the added unification constraint.

Definition 2.3.9 (Resolution and factorization rules)**.** The resolution and factorization rules are given in Fig. 2.4.

$$\frac{[A]^p \vee C \qquad [B]^{\neg p} \vee D}{C \vee D \vee [A \doteq B]^{\mathrm{F}}}\;\texttt{(Resolve)} \qquad \frac{[A]^p \vee [B]^p \vee C}{[A]^p \vee C \vee [A \doteq B]^{\mathrm{F}}}\;\texttt{(Factor)}$$

Figure 2.4: Resolution and factorization rules

Since the simplification rules eliminate logical constants, such symbols cannot occur inside unification constraints and therefore a search for unifiers containing logical symbols will always fail. Huet's solution to the problem was to add splittings rules which try to instantiate set variables with different terms containing logical symbols.

Definition 2.3.10 (Splitting rules)**.** The set of splitting rules is given in Fig. 2.5 where Y, Z and z are new variables and s_α a new Skolem term.

$$\frac{C \vee [X(\overline{t_n})]^{\mathrm{T}}}{C \vee [Y]^{\mathrm{T}} \vee [Z]^{\mathrm{T}} \vee [X(\overline{t_n}) \doteq (Y \vee Z)]^{\mathrm{F}}}\,(S_\vee^T) \qquad \frac{C \vee [X(\overline{t_n})]^p}{C \vee [Y]^{\neg p} \vee [X(\overline{t_n}) \doteq \neg Y)]^{\mathrm{F}}}\,(S_\neg^{TF})$$

$$\frac{C \vee [X(\overline{t_n})]^{\mathrm{F}}}{C \vee [Y]^{\mathrm{F}} \vee [X(\overline{t_n}) \doteq (Y \vee Z)]^{\mathrm{F}}}\,(S_\vee^{F_l}) \qquad \frac{C \vee [X(\overline{t_n})]^{\mathrm{F}}}{C \vee [Z]^{\mathrm{F}} \vee [X(\overline{t_n}) \doteq (Y \vee Z)]^{\mathrm{F}}}\,(S_\vee^{F_r})$$

$$\frac{C \vee [X(\overline{t_n})]^{\mathrm{T}}}{C \vee [Yz^\alpha]^{\mathrm{T}} \vee [X(\overline{t_n}) \doteq \Pi_\alpha Y]^{\mathrm{F}}}\,(S_\Pi^T) \qquad \frac{C \vee [X(\overline{t_n})]^{\mathrm{F}}}{C \vee [Ys_\alpha]^{\mathrm{F}} \vee [X(\overline{t_n}) \doteq \Pi_\alpha Y]^{\mathrm{F}}}\,(S_\Pi^F)$$

Figure 2.5: Splitting Rules

Definition 2.3.11 (Variants)**.** Let C be a clause, V the set of all free variables in C and σ a substitution mapping each variable in V to a new variable, then $C\sigma$ is a variant of C.

Definition 2.3.12 (Constrained resolution calculus)**.** The constrained resolution calculus contains the rules given in figures 2.2, 2.4, 2.3 and 2.5. All resolution calculi presented in this thesis also have axioms which are clauses from an initial clauses set.

Definition 2.3.13 (Derivations)**.** A derivation in the constrained resolution calculus is a sequence of rule applications such that each rule is applied to variants of clauses occurring earlier in the sequence. A derivation can also be described by an acyclic directed graph.

Definition 2.3.14 (Substitutions (of derivations))**.** Given a derivation of a clause C and let $\sigma_1, .., \sigma_n$ be all substitutions computed in the derivation, then the substitution $\sigma_1 \circ .. \circ \sigma_n$ is called the substitution of the derivation.

The following lemma will be used later in the thesis.

Lemma 2.3.15. If a clauses set S is refutable using `CRC` then we can obtain a refutation of S containing a clause C, such that below it we have only rule applications from Fig. 2.2 and above it only rule applications from the other sets.

Proof. We need to show that no rule among the ones from figures 2.4, 2.3 and 2.5 depends on a substitution generated by the unification rules. For the splitting rules it is clear as if they are applicable after a substitution is applied they are also applicable before. For the simplification rules, we might generate a pseudo-literal by applying a substitution, which can then be subjected to simplification, but if that is the case, we can apply splitting on the same variable and allow unification (later) to decompose the term. `(Resolve)` and `(Factor)` are applied to literals only and therefore if they can be applied before, they can be applied (using splittings if necessary) also afterwards. □

Remark 2.3.16. The above lemma shows that in general, the simplification rules are not required for completeness in a constrained resolution which postpones unification until a clause containing only unification constraints is found.

A Note About Semantics

The main purpose of the resolution calculus given in this thesis is to serve as a practical tool for fully automated higher-order theorem proving. Therefore, the semantical properties of the calculus will not be investigated too deeply.

Since any higher-order calculus will be incomplete with regard to standard semantics, designer of such calculi aim to other semantics. The original resolution calculi of Andrews [3] and Huet [48] were proven to be complete with regard to V-complexes [3]. In the last two decades though and especially through the work of Benzmüller and Kohlhase [8], newer calculi were designed which enjoy completeness with regard to Henkin's semantics [42].

In [16] it is shown that one can obtain a constrained resolution calculus which is Henkin complete by either adding infinitely-many extensionality axioms or extensionality rules. The completeness results given in this thesis are proved with relation to the completeness of Huet's constrained resolution calculus and the problem of obtaining Henkin's completeness is not discussed any further. It is important to note that extensionality is not required to prove many interesting theorems, for example in second-order arithmetic, so a non-extensional higher-order resolution calculus is useful in practice.

Theorem 2.3.17 (Completeness)**.** If a clauses set is unsatisfiable with regard to V-complexes [3], then we can refute it using `CRC`.

CHAPTER 3

Context Unification

One of the earliest decidability results in higher-order unification was the decidability of the problem, whether two terms which are freely generated over a group are unifiable [59]. The context unification problem [21] can be seen as a generalization of the theory of free groups as well as a restriction of second-order logic. Context unification problems augment first-order problems with unary second-order variables, which can be substituted by first-order contexts (see Def. 2.1.51).

Context unification has applications in several fields in computer science, such as computational linguistics ([63], [64]) and rewrite systems ([65]). The decidability question of context unification is still an open problem. Despite that, there are several subclasses of the problem, which were shown to be decidable ([70], [21], [55], [73]).

The algorithms in these papers efficiently decide the unifiability problem but do not enumerate unifiers. When one is interested in the enumeration of all pre-unifiers, the pre-unification algorithm (see Def. 2.2.15) is the only option.

In this section we give an algorithm that combines the efficient unification decision procedure with the ability to enumerate all pre-unifiers. The main motivation for such an algorithm is as a possible replacement for the unification rules in higher-order resolution calculi (see Chap. 5). The relative simplicity of the algorithm and the fact that it is based on the pre-unification algorithm and can use its correctness proofs, form two lesser motivation arguments. In light of the fact that there are many unification algorithms even for the sub-problem of unifying strings and all of them are far more complicated than Huet's pre-unification algorithm, these two motivations are still important.

The techniques used in this chapter will be developed further in Chap. 4 and the algorithms in this chapter can be seen as simplified versions of the more complex algorithms used in Chap. 4.

3.1 Context Unification Problems

In this section we will describe the class of context unification problems and describe a variant of the pre-unification algorithm, which is applicable to these problems.

Since the language used in this section is a restricted version of the typed lambda calculus, including only a first-order signature and restricted second-order variables, we will redefine it in the next definition. We would like to make a distinction between valid terms as defined next and between contexts and context variables. The later, when not applied to terms, are not valid terms in our language.

Definition 3.1.1 (Terms and context variables)**.** Let Σ be a first-order signature. In this chapter we will use different denotations for first and second-order variables. First-order variables ($\mathfrak{V}^{i}$) will be denoted by the lower case letters x, y and z while context variables ($\mathfrak{V}^{i \to i}$) will be denoted by the capital letters X, Y and Z. We will also denote the set of context variables by $\mathfrak{V}_c$. The set $\mathtt{Term}_c$ is defined inductively:

- if $f \in \Sigma$ and $\tau(f) = i$ then $t \in \mathtt{Term}_c$.
- if $x \in \mathfrak{V}^{i}$ then $x \in \mathtt{Term}_c$.
- if $X \in \mathfrak{V}_c$ and $t \in \mathtt{Term}_c$ then $X(t) \in \mathtt{Term}_c$.
- if $f \in \Sigma$, $\mathtt{ar}(f) = n$ and $t_i \in \mathtt{Term}_c$ for $0 < i \leq n$ then $f(t_1, .., t_n) \in \mathtt{Term}_c$.

Remark 3.1.2. The language defined above, when restricted to a monadic signature Σ containing also the constant ϵ, is exactly the language of words over the alphabet Σ. The unification problem over this language is called string unification and was shown to be decidable [59].

When searching for unifiers for context unification problems, context variables will be replaced by first-order contexts (see Def. 2.1.51).

Definition 3.1.3 (Context substitutions)**.** Context substitutions are substitutions whose domain is a subset of $\mathfrak{V}^{i} \cup \mathfrak{V}_c$ and their range is a subset of $\mathtt{Term}_c \cup \mathtt{Context}_i$ such that first-order variables are mapped to terms and context variables are mapped to contexts. Context unifiers and pre-unifiers are defined analogously to unifiers and pre-unifiers.

Definition 3.1.4 (Context unification constraints and systems)**.** A context unification constraint is a unification constraint $t \doteq s$ where $t, s \in \mathtt{Term}_c$. A pseudo constraint is a unification constraint $x \doteq s$ where $x \in \mathfrak{V}_c$ and $s \in \mathtt{Context}_i$ (see Def. 2.1.46). A context unification system is a system of constraints and pseudo constraints. The notion of pre-solved systems is the same as for regular unification systems and constraints.

Example 3.1.5. *The context unification problem:*

- $\{X(a) \doteq f(b, y), f(y, a) \doteq f(a, a), Y(y) \doteq z\}$

is pre-unifiable by $[f(b, [.])/X, a/y]$.

Definition 3.1.6 (Context partial bindings)**.** A context partial binding for a function symbol f of arity n and index $0 < k \leq n$ is the context $f(x_1^i, .., x_k^{i \to i}([.]), .., x_n^i)$ where x_i for $0 < i \leq n$ are new variables. We will denote context partial bindings by $\mathtt{PB}(f, k)$.

Example 3.1.7. *We will compute the set of partial bindings for the context unification constraint: $X(a) \doteq f(b, y)$ from Ex. 3.1.5.*

- $\mathtt{PB}(f, 1) = f(Y([.]), z)$.
- $\mathtt{PB}(f, 2) = f(z, Y([.]))$.

where z and Y are fresh free variables.

Remark 3.1.8. In the rest of this section, we will refer to context constraints, systems, substitutions, unifiers, pre-unifiers and partial bindings as constraints, systems, substitutions, unifiers, pre-unifiers and partial bindings respectively.

Definition 3.1.9 (The set of rules $\mathtt{PUA}_C$ (pre-unification algorithm for context unification))**.** Let S be a unification system, then the set of rules $\mathtt{PUA}_C$ is defined in Fig. 3.1.

$$\frac{S \cup \{A \doteq A\}}{S}\ \mathtt{(Delete)} \qquad \frac{S \cup \{f(\overline{s_n}) \doteq f(\overline{t_n})\}}{S \cup \{s_1 \doteq t_1, .., s_n \doteq t_n\}}\ \mathtt{(Decomp)}$$

$$\frac{S \cup \{x \doteq t\} \qquad x \notin \mathtt{FV}(t), \sigma = [t/x]}{\sigma(S) \cup \{x \doteq t\}}\ \mathtt{(Bind)}$$

$$\frac{S \qquad X(s) \doteq f(\overline{t_n}) \in S, 0 < k \leq n, u = \mathtt{PB}(f, k), \sigma = [u/X]}{\sigma(S) \cup \{X \doteq u\}}\ \mathtt{(Imitate)}$$

$$\frac{S \qquad X(s) \doteq t \in S, \sigma = [[.]/X]}{\sigma(S) \cup \{X \doteq [.]\}}\ \mathtt{(Project)}$$

Figure 3.1: $\mathtt{PUA}_C$- Huet's pre-unification rules for context unification

Remark 3.1.10. Please note that the title "Huet's pre-unification rules" is a bit misleading as the last two rules are different from the ones in Fig. 2.1. While in the original algorithm we force any execution to have the `(Bind)` rule executed after each application of `(Imitate)` or `(Project)`, in the above algorithm the `(Bind)` application is integrated into the `(Imitate)` and `(Project)` rules.

Remark 3.1.11. Note that if a unification system does not contain pseudo constraints, applying $\mathtt{PUA}_C$ will result in systems having pseudo constraints in solved form only. From now on we will assume that the initial system does not contain pseudo constraints.

Example 3.1.12. *The unifier $[ff([.])/X]$ of the context system:*

- $Xfa \doteq fXa$

can be obtained from the solved form of the following derivation:

$$\begin{array}{cl}
\{Xfa \doteq fXa\} & \\ \hline
\{fX'fa \doteq ffX'a, X \doteq f(X'[.])\} & \texttt{(Imitate)} \\ \hline
\{X'fa \doteq fX'a, X \doteq f(X'[.])\} & \texttt{(Decomp)} \\ \hline
\{fX''fa \doteq ffX''a, X \doteq f(f(X''[.]), X' \doteq f(X''[.])\} & \texttt{(Imitate)} \\ \hline
\{X''fa \doteq fX''a, X \doteq f(f(X''[.]), X' \doteq f(X''[.])\} & \texttt{(Decomp)} \\ \hline
\{fa \doteq fa, X \doteq f(f([.])), X' \doteq f([.]), X'' \doteq [.]\} & \texttt{(Project)} \\ \hline
\{X \doteq f(f([.])), X' \doteq f([.]), X'' \doteq [.]\} & \texttt{(Delete)}
\end{array}$$

When attempting to define a complete set of pre-unifiers for context unification problems we face the problem that the completing substitution (see Def. 2.2.8) maps context variables to terms which are not contexts. Indeed, the pre-unification algorithm $\texttt{PUA}_C$ may compute pre-unifiers which are not more general than any unifier. The reverse direction still holds however and we will use it in order to define the complete sets.

Definition 3.1.13 (Complete sets of pre-unifiers)**.** Given a unification system S, its complete set of pre-unifiers is the set $\texttt{PreUnifiers}(S)$ of (normalized) substitutions such that:

- for every normalized $\theta \in \texttt{Unifiers}(S)$ there exists $\sigma \in \texttt{PreUnifiers}(S)$ such that $\sigma|_{dom(\theta)} \leq \theta$.

This weaker notion of soundness also means that even if we can decide, for a given system S, if the set $\texttt{PreUnifiers}(S)$ is non-empty, that still does not imply that the system S is unifiable. A simple example demonstrating this is the constraint $X(a) \doteq X(b)$ which is not unifiable despite being in pre-solved form. Nevertheless, the set $\texttt{PreUnifiers}$ is still of interest as in some cases it can be processed further in order to decide unifiability. Such a case is presented in Sec. 3.5.

Corollary 3.1.14 (Completeness)**.** If θ is a pre-unifier of a unification system S, then there exists a pre-solved system S', which is obtainable from S using $\texttt{PUA}_C$ such that $\sigma_{S'}|_{\texttt{FV}(S)} \leq \theta$.

Proof. Follows directly from Thm. 2.2.17 □

While proving soundness with regard to $\texttt{PreUnifiers}$ does not make sense, we can prove a weaker notion of soundness:

Corollary 3.1.15 (Soundness)**.** If S' is obtained from a unification system S using $\texttt{PUA}_C$ then $\texttt{PreUnifiers}(S')|_{\texttt{FV}(S)} \subseteq \texttt{PreUnifiers}(S)$ where $\texttt{PreUnifiers}(S')|_{\texttt{FV}(S)} = \{\sigma|_{\texttt{FV}(S)} \mid \sigma \in \texttt{PreUnifiers}(S')\}$.

Proof. Follows directly from Thm. 2.2.16 □

3.2 Regular Contexts and Bounds

When comparing the algorithms $\mathtt{PUA}_C$ and $\mathtt{PUA}$, we notice that in $\mathtt{PUA}_C$ the number of higher-order unsolved variables never increases and does actually decrease on each application of `(Project)`. This implies that there are at most n applications of the rule `(Project)` in any execution of $\mathtt{PUA}_C$ on a system S, where n is the number of context variables in S.

A major difference between first and higher-order unification is the non-determinism added by the later in the form of the two rules `(Imitate)` and `(Project)`. The fact that in context unification problems we have a bound on the number of possible applications of the `(Project)` rule, means we can restrict this non-determinism within two applications of the `(Project)` rule.

When analyzing how the `(Imitate)` rule can be applied in this way, we make a distinction between two possible cases. When the rule is applied on an acyclic variable, we claim that the `(Imitate)` applications can be simulated by a first-order algorithm and therefore their number can be bounded by the same bounds as in first-order unification algorithms. When we apply it on cyclic variables, we claim that the imitations must be of a specific form, which can be described by a regular expression.

When we disallow the application of the `(Project)` rule on an acyclic problem, any run of the algorithm can be simulated in a sense by a run of a first-order unification algorithm. The following definition is motivated by the fact that the depth of terms generated during the execution of first-order unification algorithms is bounded.

Definition 3.2.1 (Maximum depths)**.** Given a system S, we denote the maximum size $\mathtt{d}(t)$ for all term t in S by $\mathtt{md}(S)$.

Definition 3.2.2 (First-order bound)**.** Given a system S, its first-order bound is $(k+1)\cdot\mathtt{md}(S)$, where k is the number of unsolved variables in S. It is denoted by $\mathtt{fbound}(S)$.

The regular contexts defined next also serve the role of bounds.

Regular contexts

Instead of bounding the depth of terms, regular contexts bound the form of the partial bindings.

Definition 3.2.3 (Regular contexts)**.** The set $\mathtt{Context}_r$ extends the set $\mathtt{Context}_i$ and is defined inductively:

- if $C \in \mathtt{Context}_i$ then $C \in \mathtt{Context}_r$.
- if $C \in \mathtt{Context}_i$ and $D \in \mathtt{Context}_r$ then $C(D) \in \mathtt{Context}_r$.
- if $C \in \mathtt{Context}_i$ and $D \in \mathtt{Context}_r$ then $C^*(D) \in \mathtt{Context}_r$ where $*$ is called a Kleene star.
- we will also define the set of regular contexts which contains at least one Kleene star as $\mathtt{Context}_* = \mathtt{Context}_r \setminus \mathtt{Context}_i$.

Example 3.2.4. *The regular context* $f(f([.])) \in \texttt{Context}_{\text{i}}$ *while the regular context* $(f[.])^*(g([.])) \in \texttt{Context}_*$.

Remark 3.2.5. Note that the iterated composition of the regular context $C[[.]]^n$ can be computed for every natural number n (see Def. 2.1.52).

Extracting regular contexts from cycles

In order to give the intuition behind why we need regular contexts and how we plan to use them, we will give an informal discussion of how it is constructed. Let us have another look at the cyclic system from Ex. 3.1.12:

$$\{Xfa \doteq fXa\} \tag{3.1}$$

We have seen in Ex. 3.1.12 how to obtain one unifier. Using similar derivations containing a number n of sequential applications of the `(Imitate)` and `(Decomp)` rules we can obtain a unifier $[f^n([.])/X]$. Can we claim that all the unifiers of this system can be generated by the regular expression $[f^*([.])/X]$? In this case, it is not difficult to prove this is indeed the case. Let us assume there is a unifier $\sigma = [C/X]$ which is not described by this regular expression, then C must contain a function symbol g other than f. Let q be the position of the hole in C. If the system 3.1 is unifiable by σ, then the system:

$$\{C[fa|_q] \doteq fC[a|_q]\} \tag{3.2}$$

obtained by applying σ to the system 3.1 is unifiable as well and there is a derivation of it ending in a pre-solved form. Let p be the position of the first occurrence of g in C and let m be the size of p. Clearly, no derivation with less than m applications of the `(Imitate)` and `(Decomp)` rules can end in a pre-solved form. The resulted system after m such applications will have, up to renaming of variables, the constraint $g(\overline{t_n}) \doteq f(g(\overline{s_n}))$. Such a system cannot be processed further and therefore, no pre-solved form can be obtained, in contrary to our assumption. Therefore, all unifiers are described by the regular expression $[f^*([.])/X]$.

Can we generalize our results to arbitrary cycles? It turns out the answer to this question is both yes and no. We can answer yes as we can, for every cycle, give a finite set of regular expressions defining a superset of the mappings, for all unifiers, of at least one variable in the cycle. We can answer no as if we want to extend these regular expressions to describe the unifiers instead of the individual mappings, we will need infinitely-many such regular expressions. In fact, it was shown [45] that there is no finite parametric representation of all unifiers for the simpler string unification problems involving more than three variables.

But it turns out our partial results are sufficient for deciding some classes of unification problems as will be seen in Sec. 3.5 in this chapter and in Chap. 4 and even are sufficient for the implementation of an higher-order resolution calculus with eager unification (see Chap. 5).

Going back to our attempt to generalize the results, we see that the extractions of regular expressions can be much more complex than what we have seen so far. The complexity arises from two different factors. The first is that we may have more than one function symbol in the cycle context and the second is that we may have more than one constraint in the cycle.

We will consider each of these cases using an example.

Assume first we have the system:

$$\{X(y_1) \doteq e(f(x_1, h(g(X(y_2), x_2))))\} \tag{3.3}$$

and we would like to compute from the cycle a superset of the mappings for X in all the unifiers of the system. We first build the cycle context, as was done before and obtain $e(f(x_1, h(g([.], x_2))))$. Following the same construct from above we can obtain the following regular context $e(f(x_1, h(g([.], x_2))))^*$ which indeed generates infinitely-many mappings for the variable X in unifiers of system 3.3. Despite that, it fails to generate mappings for some other unifiers of the system, such as $[e([.])/X]$. We can generate this unifier if we consider, in addition to the previous regular context, also the regular context $e(f(x_1, h(g([.], x_2))))^*(e([.]))$. Are these two regular contexts enough to generate all possible mappings? It is easily seen that for any other prefix of the context $e(f(x_1, h(g([.], x_2))))$, such as $e(f(x_1, [.]))$, we need an additional regular context. The complete set of regular contexts for generating the mappings for X in all possible unifiers of the system 3.3 is then the set:

$$\{e(f(x_1, h(g([.], x_2))))^*(C') \mid \exists C''\ C'C'' = e(f(x_1, h(g([.], x_2))))\} \tag{3.4}$$

where the contexts C' are all the possible prefixes. Since the two trivial prefixes create the same regular context, the complete set contains four elements.

This argument leads us to the following lemma, which is based on an essential step from the proof of Lemma 5.8 in [70]. More information about this essential step can be found in [70], [71] and [56]. Despite the similarities of the proofs, our proof differs in several important points. First, the proof is about the completeness using regular languages and not about searching for minimal unifiers. Second, our algorithm is using a minimal set of rules, in contrast to a much bigger set of rules in the other papers. A consequence of this is that we indirectly treat cases, which are treated directly in the other papers, such as the treatment of ambiguous cycles.

Lemma 3.2.6. Given a standard cycle $\{Xt \doteq C[Xs]\}$ in system S, then for every ground unifier σ of S, $\sigma(X) \geq_s (C^k(C'))$ for $k \geq 0$ and C' a prefix of C.

Proof. Let A be the greatest common prefix of $(C^{k+1})\sigma$ and $\sigma(X)$. Then, $A = (C\sigma)^l(C')$ for some $l \leq k$ where C' is a proper prefix of $C\sigma$ and let C'' be a context such that $C'(C'') = C\sigma$. Since A is a prefix of $\sigma(X)$, we can choose k, without loss of generality, to be the depth of $\sigma(X)$. Assume that $\sigma(X) \neq A$, then $\sigma(X) = A(f(t_1, .., [.]_{@k}, .., t_n))(D)$ for some context D, where $A(f(t_1, .., [.]_{@k}, .., t_n))$ is not a prefix of $((C\sigma))^{k+1}$. Applying σ to the unification constraint, we get:

$$A(f(t_1, .., t'_k, .., t_n)) = (C\sigma)(A(f(t_1, .., t''_k, .., t_n))),$$

where t'_k, t''_k are terms. Applying `(Decomp)`s we get:

$$f(t_1, .., t'_k, .., t_n) \doteq C''(C'(f(t_1, .., t''_k, .., t_n)))$$

Now we consider the head component of the main path of C'' (see Def. 2.1.46). If it is k, then we get a contradiction to the maximality of A as since σ is a ground unifier, it should include f as well. Otherwise, it is $i \neq k$ and let $C''[.] = f(s_1, .., D'_{@i}, .., s_n)$, then we get from the constraint, after one `(Decomp)`, that

$$t_i \doteq D'(C'(f(t_1,..,t_i,..,t_n)))$$

which is a contradiction to the unifiability of the pair as the positions of the holes in D' and C' are rigid (according to the definition of standard cycles). $\square$

We can now consider the more complex case of standard cycles containing more than one constraint:

$$\{X_1t_1 \doteq X_2a, X_2t_2 \doteq hg(X_1s_2, b)\} \tag{3.5}$$

and let us consider the substitution $[hg(y,[.])/X_2]$ which gives us the system:

$$\{X_1t_1 \doteq hg(y,a), hg(y,t_2) \doteq hg(X_1s_2, b)\} \tag{3.6}$$

after applying a couple of `(Decomp)` s followed by a `(Bind)` we obtain the following standard cycle:

$$\{X_1t_1 \doteq hg(X_1s_2, a)\} \tag{3.7}$$

whose unifiers, we already know, must be of the form $[((hg([.],a))^kC')/X_1]$ for some prefix C' and natural number k.

Despite the close similarity between the original cycle context $hg([.],b)$ and the new $hg([.],a)$ they are different and when trying to compute, from a cycle context, a superset of mappings for some variable and for all unifiers of the cycle, we must take this into account. If we look for the reason for the different cycle context, we find that as we substituted for X_2 a context whose main path is not a prefix of the main path of the cycle context (instead of position 1.1 we chose position 1.2) we replaced the original cycle context with almost the same version, but with another term at position 1.2. This "derailing" [70] of the hole in the mapping for the variable X_2 turns out to be the only source of complication and as can be seen from the example, it can happen at most $n-1$ times where n is the number of constraints in the standard cycle.

Definition 3.2.7 (Derailing)**.** Given a context C let p be a prefix of its main path such that $\texttt{hd}(C|_p) = f$ and $\texttt{ar}(f) = n > 1$ and let $0 < k \leq n$ be the head component of the main path of $C|_p$. Then, the context $C[f(y_1,..D_{@k},..,y_n)|_p]$, where $D = C|_{p.k}$ and the y_i for $0 < i \leq n$ are new first-order variables, is called a derailing of C. We call the context $C[f(y_1,..,[.]_{@k},..,y_n)|_p]$ the pre-context of the derailing and the context D the post-context of the derailing. We denote by $\texttt{one-derail}(C)$ the set of all possible derailings of a context C. This set always includes also the empty derailing, i.e. the context C itself. In this case the pre-context is empty and the post-context is equal to C. Note that this set is finite and its size is equal to the number of non-unary symbols occurring at prefix positions of the main path. Given a derailed context D, we sometimes denote by D_{pre} and D_{post} its pre and post-contexts.

Example 3.2.8. *The set of all derailings for the context $hg([.],b)$ from the above example is $\{hg([.],b), hg([.],y)\}$. the context $hg([.],y)$ is the pre-context and $[.]$ is the post-context of the non-trivial element in the set.*

Let us consider again system 3.5 from the discussion above. We can say that based on the cycle context $hg([.],b)$ and since it has only one possible derailing $\{hg([.],y)\}$, we can describe a superset of mappings $\{hg([.],y)^k \mid k \geq 0\} \cup \{hg([.],y)^k(h([.])) \mid k \geq 0\}$ such that for every unifier of the system one of the variables X_1 or X_2 will be mapped to a context C such that there is a context C_0 in the set and a substitution δ such that $C = C_0\delta$.

When considering longer standard cycles with more complex cycle contexts the set can be defined recursively:

Definition 3.2.9 (Iterated derailing)**.** Let $X_1t_1 \doteq X_2s_1, .., X_mt_m \doteq C[X_1s_m]$ be a standard cycle, then the set $\texttt{derail}(m, C)$ of the iterated derailed contexts for the cycle context C is defined as follows:

- if $m = 1$ then $\texttt{derail}(1, C) = \{C^*C' \mid \exists C''\ C'C'' = C\}$.
- if $m > 1$ then $\texttt{derail}(m, C) =$
  $\{C^*(D_{pre}(C^r)) \mid D \in \texttt{one-derail}(C), C^r \in \texttt{derail}(m-1, D_{post}(D_{pre}))\}$

Example 3.2.10. *The following tables show the construction of all (non-trivial) iterated derailings for the cycle context $hg(ef(a, [.]), b)$ of the system:*

$$\{X_1t_1 \doteq X_2s_1, X_2t_2 \doteq hg(ef(a, X_1s_2), b)\} \tag{3.8}$$

For $m = 2$ we have:

C	D_{pre}	D_{post}	C^r
$hg(ef(a, [.]), b)$	$hg([.], y)$	$ef(a, [.])$	$\in \texttt{derail}(1, ef(a, hg([.], y))$
	$hg(ef(y, [.]), b)$	$[.]$	$\in \texttt{derail}(1, hg(ef(y, [.]), b)$

For $m = 1$ we have:

C	$\texttt{derail}(1, C)$
$ef(a, hg([.], y))$	$(ef(a, hg([.], y)))^*$
	$(ef(a, hg([.], y)))^*(e([.]))$
	$(ef(a, hg([.], y)))^*(e(f(a, [.])))$
	$(ef(a, hg([.], y)))^*(e(f(a, h([.]))))$
$hg(ef(y, [.]), b)$	$(hg(ef(y, [.]), b))^*$
	$(hg(ef(y, [.]), b))^*(h([.]))$
	$(hg(ef(y, [.]), b))^*(h(g([.], b)))$
	$(hg(ef(y, [.]), b))^*(h(g(e([.]), b)))$

The first regular context generated according to the tables is:

$$(hg(ef(a, [.]), b))^*hg([.], y)(ef(a, hg([.], y)))^* \tag{3.9}$$

Definition 3.2.11 (Instantiations of regular contexts)**.** Let C be a context, then the infinite set $\texttt{insts}(C, n) = \{C'' \mid C' \in \texttt{derail}(n, C)\}$ where C'' is obtained from C' by replacing each Kleene star with a natural number is called the set of instantiations of the regular contexts for the cycle context C and n iterations.

We first prove the following property of instantiations.

Lemma 3.2.12. Given a system S containing a standard cycle with a cycle context C and let $D \in_s \texttt{insts}(C, l)$ for some context D and $l \geq 1$, then $D \in_s \texttt{insts}(C, k)$ for all $k > l$ (see Def. 2.1.18).

Proof. For the corresponding iterations we replace the Kleene star with 0 and choose an empty prefix (C_{pre}) in a trivial derailing. $\square$

We now prove the following relationship between instantiations.

Lemma 3.2.13. Let C be a context, C' and C'' contexts such that $C'C'' = C$ and $C'' = f(v_1, .., D'_{@k}, .., v_n)$. Assume that:

- $C_0 \in_s \texttt{insts}(D'C'f(z_1, .., [.], .., z_n), m)$ and
- $C_1 \geq_s (C^l C' f(z_1, .., C_0, .., z_n))$ for some $l \geq 0$.

Then, $C_1 \in_s \texttt{insts}(C, m+1)$.

Proof. From the assumptions we know that there are substitutions θ_1 and θ_2 and a context C_2 such that:

- $C_2 \in \texttt{insts}(D'C'f(z_1, .., [.], .., z_n), m)$.
- $C_0 = C_2'\theta_1$.
- $C_1 = (C^l C' f(z_1, .., C_0, .., z_n))\theta_2$.

Furthermore, since $C_2 \in \texttt{insts}(D'C'f(z_1, .., [.], .., z_n), m)$ we know that there is a regular context $C_r \in \texttt{derail}(m, D'C'f(z_1, .., [.], .., z_n)$ such that C_2 is an instance of C_r. We choose now $C_{pre} = C'f(z_1, .., [.], .., z_n)$ and $C_{post} = D'$. Clearly, $C_{pre}C_{post} \in \texttt{one-derail}(C)$ and therefore, $C^* C_{pre} C_r = C^* C' f(z_1, .., C_r, .., z_n) \in \texttt{derail}(m+1, C)$ and $C^l C_{pre} C_0 \in \texttt{insts}(C, m+1)$. Now, since $C_1 = (C^l C_{pre} C_0)\theta_2$, we get that $C_1 \in_s \texttt{insts}(C, m+1)$. $\square$

The next lemma states that for each ground unifier of a cyclic problem, there is one variable in the cycle whose mapping in the unifier is subsumed by the relevant set of instantiations. The proof proceeds by induction on the size of the cycle and attempts to reduce the size of the cycle. For an example of how the induction step in the proof takes place please refer to Ex. 3.2.15.

Lemma 3.2.14. Given a standard cycle $X_1 t_1 \doteq X_2 s_1, .., X_n t_n \doteq C[X_1 s_n]$, then for any ground unifier σ of S there is an index $0 < i \leq n$ such that $\sigma(X_i) \in_s \texttt{insts}(C, n)$.

Proof. By induction on the number of constraints in the cycle. Let A be the greatest common prefix of $\sigma(X_i)$ and $(\sigma(C))^h$ for $0 < i \leq n$ where h is the minimal depth of all $\sigma(X_i)$. Then, $A = \sigma(C)^l(C')$ for some $l \leq h$ where C' is a proper prefix of $\sigma(C)$ and let C'' be a context such that $C'(C'') = \sigma(C)$.

- for $n = 1$ we can use Lemma 3.2.6 in order to show that $\sigma(X_1) = A$ and therefore that $\sigma(X_1) \in_s \texttt{insts}(C, 1)$.
- for $n > 0$, if there is $0 < i \leq n$ such that $\sigma(X_i) = A$ then we are done since $A \in_s \texttt{insts}(C, r)$ for all $r > 0$. Assume otherwise, then A is a proper prefix of $\sigma(X_i)$ for all $0 < i \leq n$. Let f be the first function symbol in C'' (of arity ar) and assume the head component of the position of the hole in C'' is k such that $C'' = f(v_1, .., D'_{@k}, ., v_{ar})$. We consider now the cycle after the application of σ.

$$Af(t_1^1, .., D^1_{@k^1}(t_1), .., t^1_{ar}) \doteq Af(t_1^2, .., D^2_{@k^2}(s_1), .., t^2_{ar}), ..,$$
$$Af(t_1^n, .., D^n_{@k^n}(t_n), .., t^n_{ar}) \doteq C[Af(t_1^1, .., D^1_{@k^1}(s_n), .., t^1_{ar})]$$

for some D^i,t^i_j, t_i and s_i for $0 < i \leq n$ and $0 < j \leq ar$. Note that the last constraint can also be displayed as:

$$Af(t_1^n, .., D^n_{@k^n}(t_n), .., t^n_{ar}) \doteq Af(v_1, .., D'_{@k}(C'f(t_1^1, .., D^1_{@k^1}(s_n), .., t^1_{ar})), .., v_{ar})$$

It is clear that f must not be a unary symbol and that there must be at least one index $0 < i \leq n$, such that $k^i \neq k$. This is so as σ is a ground unifier and if all the positions of the holes are k, then all terms t_i^j for all j will be equal (to v_i) and the symbol f will be included in the maximal common prefix. We would like now to show how we can obtain a smaller cycle. Let $I = \{i_1, .., i_p\}$ contain all the indices $0 < i \leq n$ such that $k^i = k$. (i.e. the head component of the position of the hole in f is k). Let $\texttt{size}(\texttt{mpath}(A)) = m$, we consider the following cycle after $(m + 1) \cdot n$ applications of the `(Decomp)` rule.

$$u_1 \doteq u'_1, .., u_n \doteq u'_n \tag{3.10}$$

where:

- for every $0 < i \leq n$, $u_i = D^i(t_i)$ if $i \in I$ and $u_i = t^i_k$ otherwise.
- for every $0 < i < n$, $u'_i = D^{i+1}(t_{i+1})$ if $i + 1 \in I$ and $u'_i = t^{i+1}_k$ otherwise.
- $u'_n = D'C'f(t^1_1, .., D^1_{@k^1}(s_n), .., t^1_{ar})$.

Now let us consider the substitution δ such that $\delta(y_i) = t^i_k$ for $0 < i \leq n$ where y_i are fresh variables. and consider the equations

$$r_1 \doteq r'_1, .., r_n \doteq r'_n \tag{3.11}$$

where:

- for every $0 < i \leq n$, $r_i = D^i(t_i)$ if $i \in I$ and $r_i = y_i$ otherwise.
- for every $0 < i < n$, $u'_i = D^{i+1}(t_{i+1})$ if $i + 1 \in I$ and $r'_i = y_{i+1}$ otherwise.
- $r'_n = D'C'f(t^1_1, .., D^1_{@k^1}(s_n), .., t^1_{ar})$.

Clearly, these equations are pre-unifiable by $\sigma \circ \delta$. After applying `(Bind)` on all equations containing y_i, we get:

$$D^{i_1}(t'_{i_1}) \doteq D^{i_2}(s'_{i_2}), .., D^{i_p}(t'_{i_p}) \doteq D'C'f(t^1_1, .., D^{i_1}_{@k}(s'_{i_p}), .., t^1_{ar}) \tag{3.12}$$

where, as applying `(Bind)` may change the indices of the arguments $t_{i_1}, .., t_{i_p}$ and $s_{i_1}, .., s_{i_p}$, we take the permutations $t'_{i_1}, .., t'_{i_p}$ and $s'_{i_1}, .., s'_{i_p}$ of them. Let us take now the substitution θ such that $\theta(Y^{i_j}) = D^{i_j}([.])$ for $0 < j \leq p$ where Y_{i_j} are fresh variables and consider the standard cycle:

$$Y^{i_1}(t'_{i_1}) \doteq Y^{i_2}(s'_{i_2}), .., Y^{i_p}(t'_{i_p}) \doteq D'C'f(t^1_1, .., Y^{i_1}_{@k}(s'_{i_p}), .., t^1_{ar}) \tag{3.13}$$

which is clearly pre-unifiable by $\sigma \circ \theta$. Assume further that we have the substitution η such that $\eta(z_i) = t_i^1$ for $0 < i \leq ar$, then the standard cycle:

$$Y^{i_1}(t'_{i_1}) \doteq Y^{i_2}(s'_{i_2}), .., Y^{i_p}(t'_{i_p}) \doteq D'C'f(z_i, .., Y^{i_1}_{@k}(s_n), .., z_{ar}) \tag{3.14}$$

is pre-unifiable by $\sigma \circ \theta \circ \eta$. Since $p < n$, we can apply the induction hypothesis in order to obtain that there is and index $0 < j \leq p$ such that $\theta(Y^{i_j}) = D^{i_j}([.]) \in_s \mathtt{insts}(D'C'f(z_i, .., [.]_{@k}, .., z_{ar})), p)$. Since $\sigma(X^{i_j}) = Af(t_1^{i_j}, .., D^{i_j}([.]), .., t_{ar}^{i_j}) = (C\sigma)^l C'f(t_1^{i_j}, .., D^{i_j}([.]), .., t_{ar}^{i_j})$, we can use Lemma 3.2.13 in order to obtain that $\sigma(X^{i_j}) \in_s \mathtt{insts}(C, p+1) \subseteq \mathtt{insts}(C, n)$ (Lemma 3.2.12). Note, that I cannot be empty as we would get from the standard cycle 4.17, after applying the `(Bind)` rule n times, the constraint:

$$y_1 \doteq D'C'(f(t_1^1, ..y_1, ..t_{ar}^1)) \tag{3.15}$$

which cannot be unified as the positions of the holes in D' and C' are rigid (according to the definition of cycles).

□

The following example illustrates the induction step in the above proof.

Example 3.2.15. *The following example extends Ex. 3.2.10. Assume we have the following standard cycle containing the context variables X_1 and X_2: $\{X_1 t_1 \doteq X_2 s_1, X_2 t_2 \doteq hg(ef(a, X_1 s_2), b)\}$ where $t_1 = ef(a, hg(d, c))$, $t_2 = b$, $s_1 = c$ and $s_2 = d$. Let σ be the following ground unifier of the problem:*

$$[(hg(ef(a, [.]), b))hg([.], c)(ef(a, hg([.], c)))/X_1, (hg(ef(a, [.]), b))hg(v, [.])/X_2] \tag{3.16}$$

where $v = ef(a, [.])hg([.], c)ef(a, hg(d, c))$. After applying σ to the cycle, we get:

$$\begin{gathered}(hg(ef(a, [.]), b))hg([.], c)(ef(a, hg(ef(a, hg(d, c)), c))) \doteq (hg(ef(a, [.]), b))hg(v, c)) \\ (hg(ef(a, [.]), b))hg(v, b)) \doteq hg(ef(a, [.]), b)(hg(ef(a, [.]), b))hg([.], c)(ef(a, hg(d, c)))\end{gathered} \tag{3.17}$$

with the maximal common prefix $A = (hg(ef(a, [.]), b))h([.])$. The two mappings can be written also as: $\sigma(X_1) = Ag([.], c)(ef(a, hg([.], c)))$ and $\sigma(X_2) = Ag(v, [.])$. Since A is a strict prefix of both mappings, we must have a derailing. The derailing occurs at $Ag(.., ..)$ as the second mapping has the hole of the context in the second position while the first mapping and the cycle context itself (after two iterations) have the context's hole in the first position. Therefore, we would like to eliminate variable X_2. This can be done as follows:

first decompose the problem in order to obtain

$$\begin{gathered}g([.], c)(ef(a, hg(ef(a, hg(d, c)), c))) \doteq g(v, c) \\ g(v, b)) \doteq g(ef(a, [.]), b)hg([.], c)(ef(a, hg(d, c)))\end{gathered} \tag{3.18}$$

another decomposition will give us the following two equations (out of four equations)

$$\begin{gathered}ef(a, hg(ef(a, hg(d, c)), c))) \doteq v \\ v \doteq ef(a, [.])hg([.], c)(ef(a, hg(d, c)))\end{gathered} \tag{3.19}$$

we now obtain the more general and unifiable problem

$$\begin{aligned} ef(a, hg(ef(a, hg(d, c)), c))) &\doteq y \\ y &\doteq ef(a, [.])hg([.], c)(ef(a, hg(d, c))) \end{aligned} \tag{3.20}$$

by replacing v with the new variable y. Since applying `(Bind)` *preserves the set of unifiers, we can apply it in order to obtain the unifiable problem*

$$ef(a, hg(ef(a, hg(d, c)), c))) \doteq ef(a, [.])hg([.], c)(ef(a, hg(d, c))) \tag{3.21}$$

Now, by replacing $ef(a, hg([.], c)))$ with X_1' we get the unifiable problem

$$X_1'(ef(a, hg(d, c))) \doteq ef(a, [.])hg(X_1'(d), c) \tag{3.22}$$

and we managed to obtain a unifiable system with a smaller cycle. We can now apply the induction hypothesis in order to obtain $\theta(X_1') \in_s \mathtt{insts}(ef(a, [.])hg([.], c), 1)$ where $\theta = [ef(a, hg([.], c)))/X_1']$ and the chosen instance is $ef(a, [.])hg([.], c)$. We can now construct the mapping for $\sigma(X_1)$ from the mapping of $\theta(X_1')$ as follows. We know that $\sigma(X_1) = Ag([.], c)(\theta(X_1'))$ and by using Lemma 3.2.13, we obtain the required result.

3.3 Pre-unification Using Regular Terms

The algorithm we will define next differs from $\mathtt{PUA}_C$ by having an environment which will be used in order to restrict the possible partial bindings in the `(Imitate)` rule. The environment will contain two types of constraints. The binding constraints will force the imitations and projections of variables to be according to regular constraints. The depth constraints will restrict the depth of terms we can obtain.

Definition 3.3.1 (Binding constraints, depth constraints and environments)**.** Let C be a regular context and X a context variable, the formula $X \lhd C$ is called a binding constraint and the formula $\mathtt{d}(X) \leq n$ is called a depth constraint. A set of binding and depth constraints is called an environment. We define the application of a substitution to a binding constraint as the constraint obtained after applying the substitution to the regular context and define the application of a substitution to environments in the obvious way. Environments contain at most one binding and one depth constraint per variable. As we will see in the algorithm, it is not possible to add more than one depth and binding constraint for each variable.

The notion of pre-solved systems is the same as for $\mathtt{PUA}_C$, ignoring the environment.

Remark 3.3.2. In order to simplify the presentation of the algorithm, we will use the following abbreviation:

- $X \lhd \notin E$ in order to denote $(X \lhd C) \notin E$ for some regular context C.

Definition 3.3.3 (The set of rules `CUA` (Context unification algorithm))**.** Let S be a unification system and E an environment, then the set of rules `CUA` is defined in Fig. 3.2. Let $\mathtt{reset}(S)$ be the function generating the environment containing a depth constraint $\mathtt{d}(X) \leq 2 \cdot \mathtt{fbound}(S)$ for each unsolved context variable X occurring in S. The initial environment is equal to $\mathtt{reset}(S)$. The function `scy` returns all standard cycles in S.

The rules of the algorithm can be understood as follows: the depth constraints apply to acyclic variables and are used in the $(\mathtt{Imitate}_{pb})$ only. They are being reset every time a `(Project)` is called and are reduced by one for new variables introduced by $(\mathtt{Imitate}_{pb})$. The $(\mathtt{Imitate}_{*})$, $(\mathtt{Imitate}_{0})$ and `(Skip)` are used for cyclic variables. $(\mathtt{Imitate}_{*})$ is used for unfolding a regular context starting with a starred context, the `(Skip)` rule is used to skip the starred context completely and $(\mathtt{Imitate}_{0})$ is used for unfolding regular contexts starting with normal contexts.

We will demonstrate the execution of the algorithm through two examples, for acyclic and cyclic problems.

Example 3.3.4. *In the following derivation of* `CUA`*, we obtain a pre-unifier for the acyclic context unification problem:*

$$\{Xha \doteq gYb, Ya \doteq f(Za,a)\} \tag{3.23}$$

$$\begin{array}{cl}
\{d(X) \leq 16, d(Y) \leq 16, d(Z) \leq 16\} \vdash \{Xha \doteq gYb, Ya \doteq f(Za,a)\} & \\ \hline
 & \mathtt{(Imitate}_{pb}\mathtt{)} \\
\{d(X_1) \leq 15, d(Y) \leq 16, ..\} \vdash \{gX_1ha \doteq gYb, Ya \doteq f(Za,a), X \doteq gX_1([.])\} & \\ \hline
 & \mathtt{(Decomp)} \\
\{d(X_1) \leq 15, d(Y) \leq 16, ..\} \vdash \{X_1ha \doteq Yb, Ya \doteq f(Za,a), ..\} & \\ \hline
 & \mathtt{(Imitate}_{pb}\mathtt{)} \\
\{d(X_1) \leq 15, d(Y_1) \leq 15, ..\} \vdash \{X_1ha \doteq f(w_1, Y_1b), f(w_1, Y_1a) \doteq f(Za,a), Y \doteq f(w_1, Y_1([.]), ..\} & \\ \hline
 & \mathtt{(Decomp)} \\
\{d(X_1) \leq 15, d(Y_1) \leq 15, ..\} \vdash \{X_1ha \doteq f(w_1, Y_1b), Y_1(a) \doteq a, w_1 \doteq Za, ..\} & \\ \hline
 & \mathtt{(Bind)} \\
\{d(X_1) \leq 15, d(Y_1) \leq 15, ..\} \vdash \{X_1ha \doteq f(Za, Y_1b), Y_1(a) \doteq a, w_1 \doteq Za, ..\} & \\ \hline
 & \mathtt{(Project)} \\
\{d(X_1) \leq 12, d(Z) \leq 12\} \vdash \{X_1ha \doteq f(Za,b), a \doteq a, Y_1 \doteq [.], ..\} & \\ \hline
 & \mathtt{(Delete)} \\
\{d(X_1) \leq 12, ..\} \vdash \{X_1ha \doteq f(Za,b), ..\} & \\ \hline
 & \mathtt{(Imitate}_{pb}\mathtt{)} \\
\{d(X_2) \leq 11, ..\} \vdash \{f(X_2ha, w_2) \doteq f(Za,b), X_1 \doteq f(X_2([.]), w_2), ..\} & \\ \hline
 & \mathtt{(Decomp)} \\
\{d(X_2) \leq 11, ..\} \vdash \{X_2ha \doteq Za, w_2 \doteq b, ..\} & \\ \hline
 & \mathtt{(Bind)} \\
\{..\} \vdash \{X_2ha \doteq Za, Y \doteq f(Za,[.]), X \doteq gf(X_2([.]), b)\} &
\end{array}$$

Note that although it might not be possible to extend a pre-unifier to a unifier of a context unification problem, the pre-unifier $[gf(X_2([.]),b)/X, f(Za,[.])/Y]$ obtained in the above derivation can be extended to the unifier $[gf([.],b)/X, f(ha,[.])/Y]$ by using $\xi = [[.]/X_2, h([.])/Z]$ which is not as trivial as the completing substitution that was defined in Def 2.2.8.

Example 3.3.5. *In the following derivation of* `CUA`*, we obtain a unifier for the cyclic monadic context unification problem:*

$$\{XXXa \doteq fXf^7a\} \tag{3.24}$$

Note that by counting the function symbols and the variables, it is easy to deduce that $[f^4([.])/X]$ is the only unifier of the problem.

$$\frac{E \vdash S \cup \{A \doteq A\}}{E \vdash S}\ \texttt{(Delete)} \qquad \frac{E \vdash S \cup \{f(\overline{s_n}) \doteq f(\overline{t_n})\}}{E \vdash S \cup \{s_1 \doteq t_1, .., s_n \doteq t_n\}}\ \texttt{(Decomp)}$$

$$\frac{E \vdash S \cup \{x \doteq t\} \qquad x \notin \texttt{FV}(t), \sigma = [t/x]}{E\sigma \vdash S\sigma \cup \{x \doteq t\}}\ \texttt{(Bind)}$$

$$\frac{E \cup \{X \triangleleft C^*(C_r)\} \vdash S}{E \cup \{X \triangleleft C_r\} \vdash S\}}\ \texttt{(Skip)} \qquad \frac{E \vdash S \qquad X(s) \doteq t \in S, (X \triangleleft C_r) \notin E, \sigma = [[.]/X]}{\texttt{reset}(S\sigma) \vdash S\sigma \cup \{X \doteq [.]\}}\ \texttt{(Project)}$$

$$\frac{E \vdash S \qquad c \in \texttt{scy}(S), X\, in\, c, X\triangleleft \notin E, C_r \in \texttt{derail}(\texttt{size}(c), \texttt{ccon}(c))}{E \cup \{X \triangleleft C_r\} \vdash S}\ \texttt{(Rec)}$$

$$\frac{E \vdash S \qquad X(s) \doteq f(\overline{t_n}) \in S, (\texttt{d}(X) \leq 0) \notin E, (\texttt{d}(X) \leq m) \in E, X\triangleleft \notin E, u = \texttt{PB}(f,k), \sigma = [u/X]}{E\sigma \cup \{\texttt{d}(X') \leq m-1\} \vdash S\sigma \cup \{X \doteq u\}}\ \texttt{(Imitate}_{pb}\texttt{)}^1$$

$$\frac{E \vdash S \qquad X(s) \doteq f(\overline{t_n}) \in S, (X \triangleleft (C^*(C_r)) \in E, C = f(t_1, .., C'_{@k}, .., t_n)), u = f(t_1, .., X'_{@k}, .., t_n), \sigma = [u/X]}{E\sigma \cup \{X' \triangleleft C'(C^*(C_r))\} \vdash S\sigma \cup \{X \doteq u\}}\ \texttt{(Imitate}_*\texttt{)}$$

$$\frac{E \vdash S \qquad X(s) \doteq f(\overline{t_n}) \in S, (X \triangleleft (C(C_r)) \in E, C = f(t_1, .., C'_{@k}, .., t_n)), u = f(t_1, .., X'_{@k}, .., t_n), \sigma = [u/X]}{E\sigma \cup \{X' \triangleleft C'(C_r)\} \vdash S\sigma \cup \{X \doteq u\}}\ \texttt{(Imitate}_0\texttt{)}$$

Figure 3.2: CUA - Context unification Rules

1. the variable X' in (Imitate$_{pb}$) is the context variable which is introduced in the context u.

$$\begin{array}{cl}
\{d(X) \leq 8\} \vdash \{XXXa \doteq fXf^7a\} & \\ \hline
\{.., X \lhd f^*([.])\} \vdash \{XXXa \doteq fXf^7a\} & \texttt{(Rec)} \\ \hline
\{.., X_1 \lhd f^*([.])\} \vdash \{fX_1fX_1fX_1a \doteq ffX_1f^7a, X \doteq fX_1([.])\} & \texttt{(Imitate}_*\texttt{)} \\ \hline
\{.., X_1 \lhd f^*([.])\} \vdash \{X_1fX_1fX_1a \doteq fX_1f^7a, ..\} & \texttt{(Decomp)} \\ \hline
\texttt{(Imitate}_*\texttt{)} + \texttt{(Decomp)} \times 3 & \\ \hline
\{.., X_4 \lhd f^*([.])\} \vdash \{X_4f^4X_4f^4X_4a \doteq fX_4f^7a, X_3 \doteq fX_4([.]), ..\} & \\ \hline
\{.., X_4 \lhd [.]\} \vdash \{X_4f^4X_4f^4X_4a \doteq fX_4f^7a, X_3 \doteq fX_4([.]), ..\} & \texttt{(Skip)} \\ \hline
\emptyset \vdash \{f^8a \doteq ff^7a, X \doteq f^4([.]), ..\} & \texttt{(Project)} \\ \hline
\emptyset \vdash \{X \doteq f^4([.]), ..\} & \texttt{(Delete)}
\end{array}$$

let σ be the substitution obtained from the derived system, then $\sigma|_X$ is a unifier of the system 3.24.

The following lemma, which will be used in the completeness proof, confirms that cyclic variables can indeed be mapped by `CUA` to contexts based on the cycle's context and size (See Def. 3.2.11).

Lemma 3.3.6. Let X be a variable in a cycle in system S with cycle context C and cycle size n and let $C' \in \texttt{insts}(C, n)$, then we can derive using `CUA` a system S' such that $\sigma_{S'}(X) = C'$.

Proof. According to the definition of `insts` there is a regular context C_r corresponding to C'. by applying the three cyclic rules `(Skip)`, `(Imitate`$_0$`)`, `(Imitate`$_*$`)` and at the end `(Project)` we can simulate any instantiation of C_r. $\square$

3.4 Soundness and Completeness

In this section we will prove the soundness and completeness of `CUA` with regard to `PUA`$_C$ but we will consider only pre-unifiers which can be extended to unifiers. Since `PUA`$_C$ was proved to enumerate all these pre-unifiers, the correctness proofs will mean that `CUA` enumerates all of them as well.

The soundness of the algorithm trivially follows from the fact that each non-trivial rule applies only substitutions. The completeness proof is based on three things. First it is relatively easy to see that there can be at most m applications of the `(Project)` rule, where m is the number of context variables in the problem. Second, the close relationship between first-order unification and `CUA` without the `(Project)` rule and the cyclic rules makes it possible to prove that if a context variable is not and cannot be a part of a cycle in the problem, then the maximal number of `(Imitate`$_{pb}$`)` applications on it between any two applications of `(Project)` is bound by the first-order bound (`fbound`). The third and most involved claim is that given a standard cycle, then for any unifier of the problem, there is a context variable in the cycle which is mapped by the unifier to a term which is subsumed by an instance generated by `insts`. This can be proved by induction on the size of the standard cycles and by utilizing the technique of derailing [71] which was introduced in the previous section.

Theorem 3.4.1 (Soundness)**.** If S' is obtained from a unification system S using `CUA` then $\texttt{PreUnifiers}(S')|_{\texttt{FV}(S)} \subseteq \texttt{PreUnifiers}(S)$ where $\texttt{PreUnifiers}(S')|_{\texttt{FV}(S)} = \{\sigma|_{\texttt{FV}(S)} \mid \sigma \in \texttt{PreUnifiers}(S')\}$.

Proof. The rules in CUA are the same as the rules in $\mathtt{PUA}_C$ but pose more restrictions on the generated substitutions. First, we restrict the depth of terms using the depth constraints and second, we generated partial bindings which are less general than the ones generated in $\mathtt{PUA}_C$ due to the use of the binding constraints. Formally, we will show that for each pre-solved form S which is obtained using CUA, there is a pre-solved form S' which can be obtained using $\mathtt{PUA}_C$ such that $\sigma_{S'} \leq \sigma_S$. The simulation of the rules (Delete), (Decomp), (Bind) and (Project) is straightforward. The simulation of the different imitation rules is done by simply replacing them with the single (Imitate) rule in $\mathtt{PUA}_C$. Since it generates a partial binding which is equal to the one generated by $(\mathtt{Imitate}_{pb})$ and is more general than the ones generated by $(\mathtt{Imitate}_*)$ and $(\mathtt{Imitate}_0)$, the substitution is more general as well. The (Skip) and (Rec) rules can be ignored completely as they affect the environment only. □

The following lemmas state that if a context variable is mapped, in some pre-unifier, to a large context, then this variable, or a smaller one, can be related to a standard cycle.

Definition 3.4.2 (Relation on variables)**.** Given a system S, the relation $<^0_c$ for S is a relation on the variables in S if for all constraints $t \doteq s \in S$ and positions p_1 and p_2 such that $\mathtt{hd}(t|_{p_1}) = x$ and $\mathtt{hd}(s|_{p_2}) = y$ and p_1 is a proper prefix of p_2, then $y <^0_c x$. We define $=^0_c$ in a similar way but require that $p_1 = p_2$. $=_c$ is the symmetric, transitive closure of $=^0_c$. We now define the relation $<^T_c$ inductively:

- if $x <^0_c y$ then $x <^T_c y$.
- if $x =_c z$ and $z <^T_c y$ then $x <^T_c y$.
- if $<^T_c z$ and $z <^T_c y$ then $x <^T_c y$.

$<_c$ is any arbitrary extension of $<^T_c$ into a total order, such that if $<^T_c$ is acyclic, so is $<_c$ (otherwise, the relation is not an ordering).

Example 3.4.3. *Given the system* $\{f(Xa, g(w,y)) \doteq f(h(z), Xb), y \doteq Xc, y \doteq w\}$*, then* $<^0_c = \{(z,X),(w,X),(y,X)\}$ *and* $=^0_c = \{(y,X),(y,w)\}$*. The relation* $<^T_c$ *then is a superset of the set* $\{(z,X),(w,X),(y,X),(X,X)\}$*.*

Definition 3.4.4 (Repeated variables)**.** Let S be a system, then a context variable X in S is called a repeated variable in S if there exists a system S' obtained from S using $\mathtt{PUA}_C$ via any sequence of transformations not including (Project) such that $\mathtt{d}(\sigma_{S'}(X)) > \mathtt{fbound}(S)$.

Example 3.4.5. *Assume we can obtain, from the system S containing the constraint $Xa \doteq t$ and having $\mathtt{fbound} = 4$, a system with the solved constraint $X \doteq f^5([.])$, then X is called a repeated variable in S.*

We first prove that in acyclic systems the depth of contexts mapped to context variables is bound by fbound.

Lemma 3.4.6. Let S be a system such that all variables in it are acyclic according to $<_c$ and X be a context variable in S. Then, for any system S' obtainable from S by using $\mathtt{PUA}_C$ without the application of a (Project), we have $\mathtt{d}(\sigma_{S'}(X)) \leq \mathtt{fbound}(S)$.

Proof. By induction on the number m of variables, ordered by $<_c$. If $m = 1$, we can apply `(Decomp)` exhaustively until we get a constraint $t \doteq s$ in a system S' such that $\mathtt{hd}(t) = X$ and clearly, for all systems S'' obtainable from S without an application of `(Project)` we have $\mathtt{d}(\sigma_{S''}(X)) \le \mathtt{md}(S') \le \mathtt{md}(S) < \mathtt{fbound}(S)$. Otherwise, let x be a maximal (first-order or context) variable according to $\le_c$. Then, there is no other variable y in S, such that there is an equation $t \doteq s \in S$, $\mathtt{hd}(t|_{p_1}) = y$, $\mathtt{hd}(s|_{p_2}) = x$ with p_1 a proper prefix of p_2. Let S_0 be the problem after the removal of all equations containing x in rigid positions. By induction hypothesis, for any system S'_0 obtainable from S_0 without the application of `(Project)` and for all variables y in S other than x, $\mathtt{d}(\sigma_{S'_0}(y)) \le \mathtt{fbound}(S_0)$. Since $\mathtt{md}(S) \ge \mathtt{md}(S_0)$, we get that for all variables y other than x, $\mathtt{d}(\sigma_{S'_0}(y)) \le \mathtt{fbound}(S_0) \le \mathtt{fbound}(S) - \mathtt{md}(S)$. Since we can choose the order of equations to be processed in $\mathtt{PUA}_C$ freely (see Remark 2.2.18), then also $\mathtt{d}(\sigma_{S'}(y)) \le \mathtt{fbound}(S) - \mathtt{md}(S)$. Now, let $t \doteq s \in S$ be an equation containing x and $t' \doteq s'$ the equation after applying several `(Decomp)`s such that $\mathtt{hd}(t') = x$. Let V_0 be the set of all variables in s', then $\mathtt{d}(\sigma_{S'}(x)) \le \mathtt{d}(s') + \max_{y \in V_0}(\mathtt{d}(\sigma_{S'}(y)))$. Since $\mathtt{d}(s') \le \mathtt{md}(S)$ and $\mathtt{d}(\sigma_{S'}(y)) \le \mathtt{fbound}(S) - \mathtt{md}(S)$, we get that $\mathtt{d}(\sigma_{S'}(x)) \le \mathtt{fbound}(S)$. □

Definition 3.4.7 (Sub-equations)**.** Given a rigid-rigid constraint $t \doteq s$, its set of sub-equations is $\{t' \doteq s' \mid t|_p = t', s|_p = s', p \in \mathtt{rigid\text{-}pos}(t) \cap \mathtt{rigid\text{-}pos}(s)\}$. For a given constraint e, we denote its set of sub-equations by $\mathtt{sub}(e)$.

Example 3.4.8. *$sub(f(XgXa, gb) \doteq f(ga, gz)$ is $\{f(XgXa, gb) \doteq f(ga, gz), XgXa \doteq ga, gb \doteq gz, b \doteq z\}$.*

Definition 3.4.9 (Problem Restriction)**.** Let S be a system, then a restriction of S with regard to a variable set $V^0 \subseteq \mathtt{FV}(S)$ is the set of all sub-equations of equations in S such that a variable from V^0 is the head of one of the terms in the sub-equation.

Example 3.4.10. *The restriction to $V^0 = \{X\}$ of the above system is the problem $\{XgXa \doteq ga\}$.*

Lemma 3.4.11. If σ unifies a system S, then it unifies a problem restriction S' of S.

Proof. S' is just a subset of the equations generated from S after the application of `(Decomp)` transformations, which preserves the set of unifiers [76]. □

Lemma 3.4.12. If a variable X in a system S is repeated, then it is either cyclic or there is a smaller variable in S, according to $<_c$, which is cyclic.

Proof. Let V_0 be the set of all smaller variables including X and assume none of them is cyclic, then the order $<_c$ is well-founded over the set V^0 and we consider the problem restriction S' of S with regard to V^0. We know that there is a derivation φ in $\mathtt{PUA}_C$ such that we obtain S' with $\mathtt{d}(\sigma_{S'}(x)) > \mathtt{fbound}(S) \ge \mathtt{fbound}(S')$. We would like to get a contradiction to the existence of such a unifier which will imply that no such (extension of a) unifier also exists for S using Lemma 3.4.11. We choose φ such that it does not contain a `(Project)` call. We can do so as `(Project)` resets repeatability (according to Def. 3.4.4) so we can choose φ starting after the last `(Project)` before obtaining S'. Now, as all the variables in S' are acyclic, we can use Lemma 3.4.6 to get a contradiction. □

The next lemmas show that if we have a cyclic variable, a standard cycle can be obtained using `CUA`.

Lemma 3.4.13. Given a system S with environment E and assume S contains a cyclic variable and for all unsolved $X \in \mathtt{FV}(S)$, there is $v \geq \mathtt{fbound}(S)$ such that $(\mathtt{d}(X) \leq v) \in E$, then we can obtain a system S' and environment E' using the rules `(Delete)`, `(Decomp)`, `(Bind)` and (`Imitate`$_{pb}$) such that S' contains a cycle over the variables $X_1, ., , X_n$ such that $(\mathtt{d}(X_i) \leq v) \in E'$ for $0 < i \leq n$.

Proof. We can obtain such a cycle with the application of `(Decomp)` and `(Bind)` only and these two rules do not affect depth constraints.

□

Lemma 3.4.14. Given a system S with environment E and assume S contains a cycle $X_1 s_1 \doteq t_1, .., X_m s_m \doteq t_m$ and assume for all $0 < i \leq m$ there is $v_i \geq \mathtt{fbound}(S)$ such that $(\mathtt{d}(X_i) \leq v_i) \in E$, then we can obtain a system S' and environment E' using the rules `(Delete)`, `(Decomp)`, `(Bind)` and (`Imitate`$_{pb}$) such that S' contains a non-unique standard cycle over the variables $Y_1, .., Y_m$ and such that $(\mathtt{d}(Y_i) \leq u_i) \in E'$ for $0 < i \leq m$ and $u_i > \mathtt{md}(S)$.

Proof. We prove this by induction on $n = \Sigma_{i=1}^{m-1} i * m_i$ where m_i is the size of the minimal position of X_{i+1} in t_i for $0 < i \leq m-1$. If $n = 0$, then we are done as $X_1 \doteq t_1$ cannot be a flex-flex constraint (see next) and we already have a non-unique standard cycle. If $n > 0$, then we apply (`Imitate`$_{pb}$) on an equation $X_j \doteq t_j$ with $0 < j < m$ maximal such that $\mathtt{hd}(t_j) \notin \mathfrak{V}$. The result, after applying (`Imitate`$_{pb}$), is again a cycle with a new variable X'_j instead of X_j. n is decreased in the new cycle as either $j > 1$ and then we get m_j is decreased by 1 and m_{j-1} is increased by 1, or $j = 1$ and then m_1 is decreased by 1. In the second case, m_m is increased but we don't count it. Therefore, we can apply the induction hypothesis in order to obtain a non-unique standard cycle. The reason E' is as above is that we apply at most n (`Imitate`$_{pb}$) steps and each one of these steps decreases one depth constraint by 1, so at worst case one constraint will be decreased by n. As we assumed the constraints to be of the form $\mathtt{d}(X_i) \leq v_i$ before we start, we will obtain, in the worst case, one constraint of the form $\mathtt{d}(X'_i) \leq u_i$ with $u_i \geq v_i - n$. Since $m_i \leq \mathtt{md}(S)$ for $0 < i \leq m$ and $v_i \geq \mathtt{fbound}(S) = (k+1) \cdot \mathtt{md}(S)$ ($k = \mathtt{size}(\mathtt{FV}(S))$) and $m \leq k$ we obtain that $u_i > \mathtt{md}(S)$. □

Lemma 3.4.15. Given a system S with environment E and assume S contains a non-unique standard cycle $X_1 s_1 \doteq X_1 v_1, .., X_m s_m \doteq t_m$ and assume $(\mathtt{d}(X_i) \leq k) \in E'$ for $0 < i \leq n$ and $k > k_0$ where k_0 is the size of the minimal position of X_1 in t_m, then we can obtain a system S' with a standard cycle using the rules `(Delete)`, `(Decomp)`, `(Bind)` and (`Imitate`$_{pb}$).

Proof. First, if the non-unique standard cycle is also standard, then we are done. Otherwise, let p_m be the minimal position in t_m of X_1. The way to achieve a standard cycle is similar to what was done in the proof of the previous lemma. By applying $\mathtt{size}(p_m)$ times the rule (`Imitate`$_{pb}$) on the equation $X_m s_m \doteq t_m$ we will obtain a standard cycle. Applying the rule $\mathtt{size}(p_m)$ times is possible according to the depth constraints. □

Lemma 3.4.16. Let S_0 be a system with a cyclic variable, then there is a system S with a cyclic variable and with environment E such that S_0 is obtainable from S using no application of the rule `(Project)` and for all unsolved variables x in S, $(\mathtt{d}(x) \leq v) \in E$ where $v \geq \mathtt{fbound}(S)$.

Proof. Let φ be the derivation of S_0 and let S_1 be the last system in the derivation which is either an initial system or immediately after the application of `(Project)`. If there is a cyclic variable in S_1, then we choose $S = S_1$ and have $(\mathtt{d}(x) \leq 2 \cdot \mathtt{fbound}(S)) \in E$ for all unsolved variables x in S_1 and we are done. Otherwise, since S_1 is acyclic, let $S = S_0$ and we can use Lemma 3.4.6 in order to obtain S such that $(\mathtt{d}(x) \leq v) \in E$ for all unsolved variables x in S where $v \geq \mathtt{fbound}(S)$. □

Lemma 3.4.17. Given a system S with environment E and assume it contains a cyclic variable, then we can obtain a standard cycle using `CUA` while applying only the rules `(Delete)`, `(Decomp)`, `(Bind)` and $(\mathtt{Imitate}_{pb})$.

Proof. We first use Lemma 3.4.16 in order to obtain a system with a cyclic variable such that for all $X \in \mathtt{FV}(S)$ there is $v \geq \mathtt{fbound}(S)$ such that $(\mathtt{d}(X) \leq v) \in E$. We use now Lemma 3.4.13 in order to obtain a cycle without having the depth constraints changed. Next we obtain a non-unique standard cycle over the variables $X_1, .., X_m$ such that $(\mathtt{d}(X_i) \leq v) \in E'$ for $0 < i \leq n$ and $v > \mathtt{md}(S)$ using Lemma 3.4.14. The last step is to obtain a standard cycle using Lemma 3.4.15 and here we note that the size of the minimal position of X_1 in t_m must be smaller than $\mathtt{md}(S)$. This is because the rigid positions of context variables cannot become deeper by applying the $(\mathtt{Imitate}_{pb})$ rule. □

In the rest of the chapter we obtain weaker results than completeness of the algorithm with regard to all pre-unifiers. Since the existence of a pre-unifier for a context unification problem does not imply that the problem is unifiable, we will consider only those pre-unifiers which can be completed into unifiers.

In the following lemma we show that for any such pre-unifier σ of a problem containing a standard cycle, we can derive a problem with one less context variable which is pre-unifiable by σ.

Lemma 3.4.18. Given a system S with a standard cycle, then for any ground σ of S, there is a derivation to a system S' using `CUA` such that S' is unifiable by σ and S' contains less unsolved context variables.

Proof. Since σ is a ground unifier of S we can use Lemma 3.2.14 in order to obtain $\sigma(X) \in_s \mathtt{insts}(C', n)$ for C' the standard cycle's context and n its size. Assume there is a context $D \in \mathtt{insts}(C', n)$ and a substitution θ such that $\sigma(X) = D\theta$. We now use Lemma 3.3.6 in order to obtain a system S' such that $\sigma_{S'}(X) = D$ and therefore $\sigma_{S'} \leq \sigma$ and we have one less unsolved context variable. □

Theorem 3.4.19 (Completeness)**.** If θ is a ground unifier of a unification system S, then there exists a pre-solved system S', which is obtainable from S using `CUA` such that $\sigma_{S'}|_{\mathtt{FV}(S)} \leq \theta$.

Proof. We will prove by induction over the number of unsolved context variables, that each unifiable pre-solved form obtainable using $\mathtt{PUA}_C$ can be also obtained by CUA. The induction hypothesis is therefore: for a given system S having at most n unsolved context variables, if it is possible to obtain a ground unifier of S using $\mathtt{PUA}_C$, then it is possible to obtain the same unifier using CUA. Induction base: CUA runs the same as the complete $\mathtt{PUA}_C$ on purely first-order problems. Induction step: Once we apply (Project) in $\mathtt{PUA}_C$, we can use the induction hypothesis so we assume we need to simulate, using CUA, the remaining rules only. We notice that all rules except (Imitate) are the same. (Imitate) differs from $(\mathtt{Imitate}_{pb})$ in CUA with regard to cyclic variables only. We consider the following two cases. If (Imitate) is applied on a variable which is not repeated and is not cyclic, then we can use Lemma 3.4.6 and obtain the same system using the rule $(\mathtt{Imitate}_{pb})$ of CUA. Now, assume we apply (Imitate) in $\mathtt{PUA}_C$ on a variable that is either cyclic or repeated. Using Lemma 3.4.12 we know that if the variable is repeated, then there is a cyclic variable in the system. We now show that without losing any unifier, we can eliminate one context variable and therefore apply the induction hypothesis. Since the only two non-deterministic rules to apply are $(\mathtt{Imitate}_{pb})$ and (Project) and an application of (Project) will allow us to use the induction hypothesis we can use Lemma 3.4.17, without losing any unifier, in order to obtain a standard cycle. Lemma 3.4.18 tells us that we can derive a system using CUA which is unifiable by θ and which contains one less unsolved context variable. □

3.5 Termination and Minimal Unifiers

In this section we will show that in practice, the number of recursive calls to $(\mathtt{Imitate}_*)$ can be bounded due to the following result [72]:

Definition 3.5.1 (Minimal unifiers)**.** Given a system S, a unifier σ of S is called minimal if there is no other unifier σ' of S with $\Sigma_{x\in\mathtt{FV}(S)}\mathtt{size}(\sigma'(x)) < \Sigma_{x\in\mathtt{FV}(S)}\mathtt{size}(\sigma(x))$.

Definition 3.5.2 (Exponent of periodicity)**.** A ground unifier σ has an exponent of periodicity n iff n is the maximal number such that there is some context variable x and ground contexts A, B and C q such that $\sigma(x) = AB^nC$.

Lemma 3.5.3 ([72], [71])**.** There are constants c and d, such that for every unifiable system S and for every minimal unifier σ of s, its exponent of periodicity is less than $c * 2^{2.14*d*\mathtt{size}(S)}$.

The Restricted CUA

The exponent computed in the lemma above allows us to replace the Kleene stars in the regular terms with concrete values.

Definition 3.5.4 (Restricted derail)**.** Let e be the exponent of periodicity of the initial system S, then the restricted derail function for S ($\mathtt{derail}_S$) is defined as derail but instead of introducing the Kleene star, the function introduces the number e. The produced contexts are called restricted regular contexts or just regular contexts.

Definition 3.5.5 (Restricted instantiations)**.** Similarly to Def 3.2.11, we define a restricted instantiation of a restricted regular context C as the context obtained by replacing the n occurrences of the exponents e from above by values $k_1, .., k_n$ such that $k_i \leq e$ for $0 < i \leq n$. In addition we define the set of all instantiations of a restricted regular context C as the finite set containing all possible restricted instantiations. In the remaining of this section `insts` will refer to its restricted versions.

Example 3.5.6. *The context* $f^4([.], a)(g^6([.])(b))$ *is a restricted instantiation of the restricted regular context* $f^e([.], a)(g^e([.])(b))$ *for* $e = 8$.

Definition 3.5.7 (Environments and constraints)**.** The notions of environments and of binding and depth constraints are the same as for `CUA`. The only difference is that we identify each binding constraint over a restricted regular context C with a natural number such that this number is the maximal size of `mpath` of all terms contained in $\texttt{insts}(C)$. Since the language is (now) finite, it is possible to compute this value. We call this number the value of the constraint.

Example 3.5.8. *Assume we have a constraint* $X \lhd C$ *where* C *is the restricted regular context from the previous example, then the value of this constraint is* 16.

Definition 3.5.9 (Restricted `CUA` (`RCUA`))**.** Let S_0 be the initial system, then the restricted `CUA` (`RCUA`) consists of the rules `(Delete)`, `(Decomp)`, `(Bind)`, $\texttt{(Imitate}_{pb}\texttt{)}$, `(Project)` and $\texttt{(Imitate}_0\texttt{)}$ from Fig. 3.2 together with the three rules in Fig. 3.3.

Lemma 3.5.10 (Soundness)**.** If S' is obtained from a unification system S using `RCUA` then $\texttt{PreUnifiers}(S')|_{\texttt{FV}(S)} \subseteq \texttt{PreUnifiers}(S)$ where $\texttt{PreUnifiers}(S')|_{\texttt{FV}(S)} = \{\sigma|_{\texttt{FV}(S)} \mid \sigma \in \texttt{PreUnifiers}(S')\}$.

Proof. Follows from Thm. 3.4.1. □

Lemma 3.5.11. Let S be a unification system and θ a minimal unifier of S, then we can obtain a pre-solved system S' using `RCUA` such that $\sigma_{S'} \leq^{|\texttt{FV}(S)} \theta$.

Proof. From the completeness of `CUA` we know that we can obtain such a pre-unifier for each unifier of S. By using Lemma 3.5.3 we can show that we do not need to seek pre-unifiers with term depth bigger than the exponent of periodicity, which is exactly the bound we use in the algorithm. □

Definition 3.5.12 (Regular measure)**.** Let E be an environment and let $d_1, .., d_n$ be the values of the binding constraints in E for all unsolved variables in S, then the regular measure of E is the sum $\Sigma_{0<i\leq n} d_i$.

Definition 3.5.13 (Depth measure)**.** Let E be an environment and let $m_1, .., m_n$ be all the numbers occurring in depth constraints in E for all unsolved variables in S, then the depth measure of E is the sum $\Sigma_{0<i\leq n} m_i$.

Theorem 3.5.14. Given a system S, `RCUA` terminates on S.

$$\frac{E \cup \{X \triangleleft C^n(C_r)\} \vdash S}{E \cup \{X \triangleleft C_r\} \vdash S\}}\ \texttt{(Skip)}$$

$$\frac{E \vdash S \qquad c \in \texttt{scy}(S), X \in c, (X \triangleleft C'_r) \notin E, C_r \in \texttt{derail}_{S_0}(\texttt{size}(c), \texttt{ccon}(c))}{E \cup \{X \triangleleft C_r\} \vdash S}\ \texttt{(Rec)}$$

$$\frac{E \vdash S \qquad x(s) \doteq f(\overline{t_n}) \in S, (X \triangleleft (C^m(C_r)) \in E, C = f(t_1, .., C'_{@k}, .., t_n)), u = f(t_1, .., X'_{@k}, .., t_n), \sigma = [u/X]}{E\sigma \cup \{X' \triangleleft C'(C^{m-1}(C_r))\} \vdash S\sigma \cup \{X \doteq u\}}\ \texttt{(Imitate}_*\texttt{)}$$

Figure 3.3: RCUA - Restricted CUA

Proof. The algorithm is finitely branching. We will show termination of a specific run by taking the lexicographic ordering of the following measure $\mu =< m_1, m_2, m_3, m_4, m_5, m_6 >$ where

- m_1 is the number of unsolved context variables,
- m_2 is the number of unsolved context variables that do not occur also in a binding constraint in the environment E,
- m_3 is the regular measure,
- m_4 is the depth measure,
- m_5 is the number of unsolved first order variables and
- m_6 is the number of symbols other than $\doteq$ in the problem.

We prove that the measure μ is decreased after any application of `RCUA`.

- An application of `(Delete)` or `(Decomp)` decreases m_6 and does not increase any other measure.
- An application of `(Bind)` decreases m_5 and does not increase m_1, m_2 or m_4. It also does not increase m_3 since the size of the `mpath` of regular contexts is not affected by `(Bind)`.
- An application of `(Imitate`$_{pb}$`)` decreases m_4 and does not increase m_1 or m_2. It does not increase m_3 following the previous argument.
- An application of `(Skip)`, `(Imitate`$_0$`)` and `(Imitate`$_*$`)` decreases m_3 as we decrease the size of the maximal `mpath` of a regular term in the new constraint by at least 1. it does not increase m_1 or m_2.
- An application of a `(Project)` decreases m_1.
- An application of a `(Rec)` decreases m_2 as it introduces a binding constraint for a context variable and is applicable only if one did not exists. It does not increase m_1.

Therefore the measure μ decreases after each application of `RCUA`. □

Decidability Result: Stratified Context Unification

In this section we will give an algorithm which decides the unifiability problem of a subclass of the context unification systems. The algorithm will take as an input a system in pre-solved form and will decide if the system is unifiable.

The stratified context unification problem was shown to be decidable in [70]. This fragment plays a role, among other fields, also in computational linguistics [27]. In this section we give another algorithm to decide the problem, which is based on `RCUA`. All the definitions in this section and the treatment of flat equations are taken from [70]. The stratified context unification algorithms given in [70] is a very specialized algorithm, which contains many rules for the different cases possible and therefore has a relatively complex correctness proof. Our version, which follows the very general pre-unification algorithm [49], benefits from its correctness proof and this allows us to give, we hope, a simpler proof for the decidability of this problem.

Definition 3.5.15 (Second-order prefixes (SO-prefixes))**.** SO-prefixes are words over $(\mathfrak{V}_2)^*$. Let P be a system, the SO-prefix of a term t is defined inductively:

- if t occurs in an equation $t \doteq s$, then t has an empty SO-prefix.
- if $t = f(\overline{s_n})$ has an SO-prefix w, then s_i, for $0 < i \leq n$, has SO-prefix w.
- if $t = X(s)$ has an SO-prefix w, then s has SO-prefix $w \cdot X$.

Definition 3.5.16 (Stratified unification systems)**.** A system S is called stratified if for every context and first-order variable in S, all its occurrences have the same SO-prefix..

Example 3.5.17. *$\{Xa \doteq XYa, fXa \doteq Za\}$ is stratified while $\{Xa \doteq XYa, fXa \doteq Ya\}$ is not.*

From now on in this section we will refer to stratified systems only.

Definition 3.5.18 (Clusters)**.** Let S be a system in pre-solved form, a cluster c of S is a minimum subset of constraints of S such that if a constraint $Xt \doteq Ys$ is in c and there is a constraint $Xv \doteq u$ in S, then it is also in c.

Example 3.5.19. *the clusters of $\{X_1a \doteq X_2b, X_1c \doteq X_3d, Y_1e \doteq Y_2f\}$ are $\{X_1a \doteq X_2b, X_1c \doteq X_3d\}$ and $\{Y_1e \doteq Y_2f\}$.*

Definition 3.5.20 (Clusters instantiations)**.** Given a system S in pre-solved form and a cluster c in S with variables $X_1, .., X_n$, `INST`(S, c) is the application of either:

- a `(Project)` on some equation in c or
- if $n > 1$ and there is a function symbol f in the signature with $m = \mathtt{ar}(f) > 1$ then:
  - choose an index $0 < k_i \leq m$ for each variable X_i such that at least two indices are different.
  - replace each X_i with $f(y_{(i,1)}, .., X_{(i,k_i)}([\cdot]), .., y_{(i,m)})$ containing new variables.
  - apply `(Decomp)` in order to eliminate the f symbol.
  - apply `(Bind)`s in order to solve the new equations having first order variables as at least one head.

Lemma 3.5.21. Given a system S in pre-solved form with a minimal (with regard to the number of equations in it) cluster c, `INST`(S,c) results in a system with either less context variables or a smaller minimal cluster.

Proof. If we apply a `(Project)` on a variable in the cycle, the number of context variables decreases. Therefore, we assume we choose the second option. After the application of the `(Decomp)` and `(Bind)` rules and as at least one variable was mapped to a term with a different index for the first character of the position to the hole, the resulted system will have a cluster smaller by at least one equation. Since we took a minimal cluster, the size of the minimal cluster in the resulted system will be smaller. In addition, since the head context variables in the clusters are all disjoint, the process does not affect other clusters. □

Example 3.5.22. *The cluster $\{Xt_1 \doteq Ys_1, Yt_2 \doteq Zs_2\}$ can be instantiated using the substitution $X \mapsto f(X'[.], w_1)$, $Y \mapsto f(Y'[.], w_2)$ and $Z \mapsto f(w_3, Z'[.])$ and after the applications of `(Decomp)` and `(Bind)`, it is reduced to the cluster $\{X't_1 \doteq Y's_1\}$.*

Lemma 3.5.23. (Lemmas 6.6 and 6.7 in [70]) Given a system S in pre-solved form and a minimal cluster c in S, then applying `INST`(S, c) is sound and complete.

Definition 3.5.24 (Stratified context unification algorithm (`SCUA`))**.** Given a system S, we apply one of the following transformations:

- if S is not in pre-solved form, we apply `RCUA`.
- if S is in pre-solved form, we apply `INST`(S, c) on a minimal cluster c in S.

Lemma 3.5.25. `SCUA` preserves the stratifiability of a problem.

Proof. The transformations of `RCUA` clearly preserve stratifiability as all manipulations on SO-prefixes are done using substitutions, which apply to the whole problem. For `INST`, the proof is exactly as in Lemma 6.5 in [70]. □

Lemma 3.5.26 (Soundness)**.** If S' is obtained from a unification system S using `SCUA` then `PreUnifiers`(S') $\subseteq$ `PreUnifiers`(S).

Proof. Following lemmas 3.5.10 and 3.5.23. □

Lemma 3.5.27 (Completeness)**.** Let S be a unification system and θ a minimal unifier of S, then we can obtain a pre-solved system S' using `SCUA` such that $\sigma_{S'} <^{|_{\text{FV}(S)}} \theta$.

Proof. Using lemmas 3.5.11 and 3.5.23. □

Lemma 3.5.28 (Termination)**.** `SCUA` terminates.

Proof. Using the lexicographic ordering on the measure $< m_1, m_2, m_3 >$ where m_1 is the number of context variables, m_2 is 1 if the problem is not pre-solved and 0 otherwise and m_3 is the size of a minimal cluster. First, it is clear that m_1 is never increased. If the problem is not in pre-solved form, then we apply `RCUA` and obtain (following Theorem 3.5.14) either a pre-solved form or an error, resulting in a decrease in m_2. If the problem is in pre-solved form, then we apply `INST` resulting, according to Lemma 3.5.21, either with less context variables and the reduction of m_1 or with a smaller minimal cluster and the reduction of m_3. It is easy to see that the application of `INST` results in a pre-solved form so m_2 does not increase. □

We can now prove the following result from [70].

Theorem 3.5.29. The stratified context unification problem is decidable.

Proof. Using lemmas 3.5.25, 3.5.26, 3.5.27 and 3.5.28. □

3.6 Open Problems

Context unification and its subclasses form an important field within higher-order unification. Despite that, there are several questions whose answer is still unknown. In this section we discuss two of them in the hope that the results obtained in this thesis can help and maybe even lead to answers to these questions.

The decidability of the context unification problem

The question whether the context unification problem is decidable is an open problem since the introduction of this class in the early 90s [20]. An interesting attempt can be found in [57]. An interesting characteristic of context unification is that pre-unifiers fail to capture the unifiability of problems. Despite that, since if a problem is unifiable then there exist pre-unifiers of the problem, we can try to find a method to decide if a certain pre-unifier is also a unifier. By restricting the possible pre-unifiers to check to a finite set, we can effectively decide the unifiability of the problem. This method was successfully used in order to show the decidability of the stratified context unification problem [70] (see Sec. 3.5).

On the other hand, there is a standard approach for looking for unifiers in higher-order problems which extends the general higher-order pre-unification algorithm with rules for guessing terms for higher-order variables (see for example [55]). These algorithms fails to terminate due to the blind guess for terms.

It might be useful to extend the pre-unification algorithm given in this chapter with the same rules and to try and show termination. This algorithm enumerates a subset of the pre-unifiers found by the general algorithm and therefore is better suited for the task. In addition, the use of regular terms in the process of unifying cycles might be extended to terms with flexible heads and therefore, to enable the use of the periodicity lemma (see Sec. 3.5) in order to show decidability.

Parametrizing complete sets of unifiers

Another important question is whether there is a way to give a finite representation of the whole set of pre-unifiers using parameters. For very restricted class, such as word unification with up to three variables [1] and context unification with up to one variable [34], it was shown that such a method exists. On the other hand, when one considers word unification problems with four variables, the problem is already unsolvable [45].

The use of regular terms in the algorithm presented in this chapter strongly implies that more subclasses exist. In the algorithm, the regular terms are used in order to approximate possible pre-unifiers by specifying the form of their supersets. The regular terms do give a precise mapping to variables until we project the arguments, in which case the question whether the mapping is indeed part of a pre-unifier depends on the unifiability of the arguments as well. Since the regular terms describe an infinite set of possible mappings, each with a different affect on the projection of the arguments, we will be forced to check an infinite number of argument projections and filter out those which do not lead to a pre-unifier.

Although this task is impossible in general, it should be made possible when dealing with projections of ground arguments. In this case it should be possible to find a finite restriction to the regular terms which may satisfy the argument projections. A trivial class which adheres to this restriction is the class of context unification problems having at most one first-order or context variable in each side of each equations.

CHAPTER 4

Bounded Higher-order Unification

The unification principle has many uses in computer sciences. Due to the undecidability of the higher-order unification problem, many applications find it necessary to restrict the use of unification to decidable classes only. This can either be done by applying unification on fragments of higher-order logic problems, whose unifiability is known to be decidable or by restricting unification algorithms to search for an incomplete set of unifiers. Among the fragments of the first we can find higher-order pattern unification [61], [67] and decidable sub-classes of context unification [70], [21], [55], [73]. When we need to consider arbitary higher-order unification problems, as is the case in higher-order resolution, we must search for an incomplete set of unifiers.

Most higher-order theorem provers, such as Isabelle [66], TPS [2] and LEO II [9], rely on Huet's pre-unification algorithm [49] (see Sec. 2.2) for the unification of higher-order terms.

Since the algorithm does not terminate, these theorem provers must search for incomplete finite sets of unifiers only. The most common way to obtain such a set is by bounding the depth of the terms in the co-domain of the unifiers. The next example, which was already given in Chap. 1, gives a family of simple unification problems where the depth of terms in the co-domain of unifiers grows exponentially while the size of the problems grows linearly.

Example 4.0.1 (see Ex. 1.3.2 for more information)**.** *The following monadic second-order equation has a unique unifier σ such that $\sigma(X_1) = a^{3^n}$. The depth of terms we need to search for cannot be therefore smaller than 3^n but the size of the problem is only $6n + 6$.*

$$\{X_1abX_1bX_2 \dots bX_nc \doteq aX_1bX_2X_2X_2 \dots bX_nX_nX_nbaaac\} \tag{4.1}$$

Another approach for obtaining incomplete sets of unifiers is by bounding the number of occurrences of bound variables in the co-domains of unifiers. This approach, called bounded higher-order unification [74], gives a more refined incompleteness as can be shown with regard to the example above, which has only 2 bound variable occurrences per variable (one in the binder and one in the term). The drawback of this method is that the computed (incomplete) set of pre-unifiers is now infinite although it was shown that the unifiability problem is decidable [74].

Despite the years that have passed since its introduction, we could not find any application in the literature for the bounded higher-order unification algorithm. It might be that despite its advantage - a much more refined bound - its disadvantage, in the form of infinite sets of unifiers, made it unusable in practice. More specifically, the use of an infinite set of most general unifiers is not practical in the context of resolution.

Our main aim in this thesis is to improve the unification procedure taking place in higher-order resolution. We believe that using the bounded unification algorithm within a resolution calculus will improve the search for refutations due to the algorithm refined bounds. For this aim we need both to be able to enumerate all the infinite unifiers in the set and to be able to decide unifiability.

In this chapter we will introduce an extension of the bounded higher-order unification algorithm, which will be based on the algorithm for context pre-unification from Chap. 3. In Chap. 5 we will show how this algorithm can be integrated into the constrained resolution calculus in order to give a refutation method which is complete with regard to the refined notion of bounded unifiers.

4.1 Bounded Higher-Order Unification Problems

In this section we will describe the class of bounded higher-order unification problems and describe a variant of the pre-unification algorithm, which is applicable to these problems. Many definitions and notions in the section are based on those in [74].

The next definition will measure terms according to the number of occurrences of bound variables in them.

Definition 4.1.1 ($\lambda\texttt{size}$)**.** The $\lambda\texttt{size}$ of a term t is defined as the number of occurrences of λ-binders and bound variables in t.

Example 4.1.2. *The $\lambda\texttt{size}$ of the term $\lambda z.g(\lambda x.z(y,x),x)$ is* 5.

Definition 4.1.3 (Bounded higher-order unification systems)**.** Let S be a unification system and $\hat{b} : \mathfrak{V} \to \mathbb{N}$ a mapping from free variables to natural numbers, then the pair $(S, \hat{b})$ is called a bounded higher-order unification system.

From now on systems will refer to bounded higher-order unification systems. The word system will also refer to the higher-order unification system within the bounded one when it will not result in any confusion.

Definition 4.1.4 (Bounded unifiers)**.** Let $(S, \hat{b})$ be a system and σ a unifier of S, then σ is a bounded unifier of $(S, \hat{b})$ if for all variables $x \in \texttt{FV}(S)$, $\lambda\texttt{size}(\sigma(x)) \leq \hat{b}(x)$. We will sometime denote the value $\hat{b}$ for a variable x also as $x^{\hat{b}=\hat{b}(x)}$.

We would like to define next the notion of bounded pre-unifiers but consider the following example:

Example 4.1.5. *The substitution $\lambda z.f(y_1(z), y_2(z))/x$ is in the set* `PreUnifiers` *of the system:*

$$x(a) \doteq f(x_1(a), x_2(b)) \tag{4.2}$$

but is not in the set of the bounded system:

$$x^{\hat{b}=2}(a) \doteq f(x_1(a), x_2(b)) \tag{4.3}$$

since the bound variable z occurs three times in the substitution. Still, we can extend this substitution to a bounded unifier of system 4.3 by applying the completing substitution $[\lambda z.y_0/y_1, \lambda z.y_0/y_2, \lambda z.y_0/x_1, \lambda z.y_0/x_2]$.

The previous example leads us to the following definition of bounded pre-unifiers.

Definition 4.1.6 ($\lambda\mathtt{size}_r$)**.** The $\lambda\mathtt{size}_r$ of a term t is defined as the number of all λ-binders and occurrences of bound variables in t in rigid positions.

Example 4.1.7. *The $\lambda\mathtt{size}_r$ of the term $\lambda z.f(X(z), z)$ is 2 while the $\lambda\mathtt{size}$ of the term is 3.*

Definition 4.1.8 (Bounded pre-unifiers)**.** Let $(S, \hat{b})$ be a system and σ a pre-unifier of S, then σ is a bounded pre-unifier of $(S, \hat{b})$ if for all variables $x \in \mathtt{FV}(S)$, $\lambda\mathtt{size}_r(\sigma(x)) \leq \hat{b}(x)$.

As was done for the context unification problems in Chap. 3, we will also restrict the partial bindings for variables in bounded systems. In the following definition, we will refer to the function $\hat{b}$ and we will extend it to map new variables as well.

Definition 4.1.9 (Bounded partial bindings)**.** The set of bounded partial bindings for a variable x of type α and for an atom a is the same set containing the partial bindings $u = \lambda\overline{y_n}.a(\lambda\overline{z^1_{p_1}}.x_1(\overline{y_n}, \overline{z^1_{p_1}}), .., \lambda\overline{z^m_{p_m}}.x_m(\overline{y_n}, \overline{z^m_{p_m}}))$ from Def. 2.2.13 where in addition we update the $\hat{b}$ function to have the following values for the new variables:

- for all $0 < i \leq m$, $n \leq \hat{b}(x_i) \leq \hat{b}(x)$.
- if a is a function symbol or a free variable (an imitation binding), then $\Sigma_{0<i\leq m}(\hat{b}(x_i) - n) \leq \hat{b}(x) - n$.
- if $a = y_i$ for $0 < i \leq n$ is a bound variable (a projection binding), then $\Sigma_{0<i\leq m}(\hat{b}(x_i) - n) \leq \hat{b}(x) - n - 1$.

We denote the set of bounded partial bindings for type α and atom a by $\mathtt{PB}^b(a, \alpha)$ for imitation bindings and $\mathtt{PB}^b(i, \alpha)$ for projection bindings with index $0 < i \leq n$.

Example 4.1.10. *We will compute the set of partial bindings for the context unification constraint: $x^{\hat{b}=2}(y) \doteq f(g, a)$ where x_1 and x_2 are fresh free variables*

- *the set of imitation bindings for atom f is $\mathtt{PB}^b(f, \mathrm{i} \to \mathrm{i}) = \{\lambda z_1.f(\lambda z_2.x_1(z_1, z_2), x_2(z_1))\}$ where $\hat{b}(x_1) = 2$ and $\hat{b}(x_2) = 1$ for example.*

- *the set of projection bindings for index* 1 *is* $\mathtt{PB}^{b}(1, \mathrm{i} \rightarrow \mathrm{i}) = \{\lambda z_1.z_1\}$ *where* $\hat{b}$ *does not change so we have only one element in the set.*

The intuition behind this definition of partial bindings is two folded. First we would like to make the sum of the bound variable occurrences in the terms mapped to the new variables no greater than their number in the term mapped to the original variable. On the other hand, the new variables might be of a more complex type than the original variable and we would like to factor that out.

The first issue is handled by factoring out the number of λ-binders in terms mapped to the original variable while the second thing is handled as follows. Since all terms are in η-expanded form, a variable of a more complex type implies more λ-binders in the partial binding but also a bigger arity for the fresh variable. Since we count also λ-binders in the definition of $\lambda\mathtt{size}_r$, these two values are factored out so the actual number of bound variables in a term which is mapped to a variable of a more complex type must be smaller than the number in the original variable by exactly the same degree.

We demonstrate it in the following example.

Example 4.1.11. *Assume we have the constraint:*

$$xa \doteq f(y, yb) \tag{4.4}$$

where a, b *are of type* i, x, y *are of type* $\mathrm{i} \rightarrow \mathrm{i}$ *and* f *is of type* $(\mathrm{i} \rightarrow \mathrm{i}) \rightarrow \mathrm{i} \rightarrow \mathrm{i}$. *A partial binding for* x *will be of the form*

$$\lambda z_1.f(\lambda z_2.x_1(z_1, z_2), x_2(z_1)) \tag{4.5}$$

such that $\hat{b}(x) \geq \hat{b}(x_1), \hat{b}(x_2)$ *and* $(\hat{b}(x_1) - 1) + (\hat{b}(x_2) - 1) \leq \hat{b}(x) - 1$. *We might first wonder where do we count the binder* z_2. *It seems that the equations above ignore completely the fact that one binder occurs only in a mapping for* x. *A closer examination of the possible mappings for* x_1, *shows us that this binder occurs in all these mappings as well, as the arity of* x_1 *is greater than the arity of* x.

Remark 4.1.12. In the rest of this section, we will refer to bounded systems, unifiers, pre-unifiers and partial bindings as systems, unifiers, pre-unifiers and partial bindings respectively.

Definition 4.1.13 (The set of rules $\mathtt{PUA}_B$ (pre-unification algorithm for bounded higher-order unification))**.** Let $(S, \hat{b})$ be a unification system, then the set of rules $\mathtt{PUA}_B$ is defined in Fig. 4.1.

Definition 4.1.14 (Pre-solved forms)**.** A unification constraint $t \doteq s$ is in pre-solved form if it is either in solved form where, for some variable x, $\mathtt{hd}(t) = x$ and $\lambda\mathtt{size}_r(s) \leq \hat{b}(x)$ or t and s are flexible terms. A system is in pre-solved form if all its constraints are in pre-solved form.

The notion of complete sets of pre-unifiers can now be carried over from the unbounded case.

Definition 4.1.15 (Complete sets of pre-unifiers)**.** Given a unification system S, its complete set of pre-unifiers is the set $\mathtt{PreUnifiers}(S)$ of (normalized) substitutions such that:

$$\frac{S \cup \{A \doteq A\}}{S}\ \texttt{(Delete)} \qquad \frac{S \cup \{\lambda\overline{z_k}.f(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_n})\}}{S \cup \{\lambda\overline{z_k}.s_1 \doteq \lambda\overline{z_k}.t_1, .., \lambda\overline{z_k}.s_n \doteq \lambda\overline{z_k}.t_n\}}\ \texttt{(Decomp)}$$

$$\frac{S \cup \{\lambda\overline{z_k}.x(\overline{z_k}) \doteq \lambda\overline{z_k}.t\} \qquad x \notin \texttt{FV}(t), \sigma = [\lambda\overline{z_k}.t/x]}{S\sigma \cup \{x \doteq \lambda\overline{z_k}.t\}}\ \texttt{(Bind)}$$

$$\frac{S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m}) \in S, u \in \texttt{PB}^b(f,\alpha), \sigma = [u/x]}{S\sigma \cup \{x \doteq u\}}\ \texttt{(Imitate)}$$

$$\frac{S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m}) \in S, 0 < i \leq k, u = \texttt{PB}^b(i,\alpha), \sigma = [u/x]}{S\sigma \cup \{x \doteq u\}}\ \texttt{(Project)}$$

Figure 4.1: $\texttt{PUA}_B$- Huet's pre-unification rules for bounded unification

- $\{\sigma \circ \xi_S \mid \sigma \in \texttt{PreUnifiers}(S)\} \subseteq \texttt{Unifiers}(S)$ where ξ is the completing substitution.
- for every normalized $\theta \in \texttt{Unifiers}(S)$ there exists $\sigma \in \texttt{PreUnifiers}(S)$ such that $\sigma|_{dom(\theta)} \leq \theta$.

The soundness and completeness proofs are given next.

Theorem 4.1.16 (Soundness). If S' is obtained from a unification system S using $\texttt{PUA}_B$ and is in pre-solved form, then $\sigma_{S'}|_{\texttt{FV}(S)} \in \texttt{PreUnifiers}(S)$.

Proof. Follows directly from Thm. 2.2.16. □

Theorem 4.1.17 (Completeness). If θ is a pre-unifier of a unification system S, then there exists a pre-solved system S', which is obtainable from S using $\texttt{PUA}_B$ such that $\sigma_{S'}|_{\texttt{FV}(S)} \leq \theta$.

Proof. The only restriction we pose is with regard to the values of $\hat{b}$ over the new variables in `(Imitate)` and `(Project)`. We consider then the set computed by $\texttt{PB}^b$ for a variable x and assume $\theta(x) = t$. Let $t = \lambda\overline{z_n}.f(t_1, .., t_m)$, then $\hat{b}(x) \geq n + \Sigma_{0<i\leq m}\lambda\texttt{size}_r(t_i)$. Let $s = \lambda\overline{z_n}.f(\lambda\overline{y_{n_1}}x_1(\overline{z_n}, \overline{y_{n_1}}), .., \lambda\overline{y_{n_m}}x_m(\overline{z_n}, \overline{y_{n_m}}))$ such that $\hat{b}(x_i) = \lambda\texttt{size}_r(t_i) + n$ for $0 < i \leq m$, then we can find a substitution σ such that $\sigma(x) = t$. We also notice that $n \leq \hat{b}(x_i) \leq \hat{b}(x)$ for all $0 < i \leq m$ on the one hand and $\Sigma_{0<i\leq m}\hat{b}(x_i) - n = \Sigma_{0<i\leq m}\lambda\texttt{size}_r(t_i) \leq \hat{b}(x) - n$ on the other, as required in Def. 4.1.9. We can similarly prove the projection case. The rest follows directly from Thm. 2.2.17. □

4.2 Regular Terms and Bounds

In the previous chapter we have found out that a major difference between the context pre-unification algorithm $\texttt{PUA}_C$ and Huet's algorithm `PUA` is the bound on how many applications of the `(Project)` rule we can use in each derivation.

We argue that the same bound also holds for bounded higher-order unification by considering the following measure.

Definition 4.2.1 (Bounding measure [74])**.** Let $(S, \hat{b})$ be a system. The bounding measure of $(S, \hat{b})$, which is denoted by `b-measure`, is the multiset $\{\hat{b}(x) - \mathtt{ar}(x) | x \in V\}$ where V is the set of unsolved variables in S. Multi-sets are ordered according to multi-sets ordering [24].

By considering the function $\mathtt{PB}^b$ we can see that any choice of a projection partial binding strictly decreases the bounding measure while the application of no other rule increases it. It is a bit less trivial to see that on an application of an `(Imitate)` but a closer examination of the values computed in the function $\mathtt{PB}^b$ shows that the value never increases.

Having a pre-defined bound on the number of possible projections means we can try to use the same bounds for acyclic and cyclic systems as we have used for the context unification systems in Chap. 3. The first-order bound used for acyclic problems does indeed carry over to bounded higher-order unification problems. Unfortunately, the regular bound becomes more complex when dealing with bounded higher-order systems.

Cycles with complex contexts

In Def. 2.1.46 in Chap. 2 we have distinguished between two types of contexts, simple and complex. In Chap. 3 we have argued that given a cycle, the cycle context must be simple. In this section we will argue the same for arbitrary higher-order systems.

When considering cycles of one constraint only, it is clear that when presented in η-expanded form, the contexts are simple as can be seen from the following example.

Example 4.2.2. *The cycle context of the standard cycle:*

$$\lambda z_0.x(t, z_0) \doteq \lambda z_0.f(g(\lambda z.x(z_0, z)), y) \tag{4.6}$$

is $f(g(\lambda z.([.])), y)$ which is simple and the cycle can be represented as:

$$\lambda z_0.x(t, z_0) \doteq \lambda z_0.f(g(\lambda z.([.])), y)(x(z_0, z)) \tag{4.7}$$

if we allow for variable capture in applications of contexts.

We will introduce the following definitions:

Definition 4.2.3 (Impure terms)**.** A term t is called impure if there is a position $p \in \mathtt{rigid\text{-}pos}(t)$ such that $t|_p = \lambda z.s$ for some term s. It is called pure otherwise.

Definition 4.2.4 (Impure cycles)**.** A cycle is called impure if there is a constraint $\lambda\overline{z_n}.x(\overline{t_m}) \doteq \lambda\overline{z_n}.t$ in the cycle and t is impure. It is called pure otherwise.

The added complexity when dealing with impure cycles is that they introduce new binders and therefore may increase the number of bound variable occurrences. In this section we will argue informally that such cases need not be considered. A formal proof will be given in Sec. 4.4.

Consider the impure cycle:

$$\{\lambda z_0.x(t_1, t_2) \doteq \lambda z_0.g(\lambda z.(x(z, s_1)))\} \tag{4.8}$$

We can have the following two possible derivations (according to the definition of $\mathtt{PB}^b$):

- we can project one of the arguments of x such as by the substitution $[\lambda z_1, z_2.z_1/x]$ or
- we can imitate the symbol g by the substitution $[\lambda z_1, z_2.g(\lambda z.x_1(z_1, z_2, z), x_2(z_1, z_2))/x]$.

In the first case we decrease the bounding measure since we eliminate the element $\hat{b}(x)$ and in the second case we decrease the bounding measure as we have that $\hat{b}(x_1) + 1 + \hat{b}(x_2) \leq \hat{b}(x)$ according to the definition of $\mathtt{PB}^b$, which implies that $\hat{b}(x_1), \hat{b}(x_2) < \hat{b}(x)$.

As was mentioned, in Sec. 4.4 we will formalize the argument that we can concentrate on pure cycles only.

Remark 4.2.5. In the remaining part of the thesis, we will generalize over the form of standard cycles and will write them:

$$\lambda\overline{z_{m_1}}.x_1(\overline{t^1_{n_1}}) \doteq \lambda\overline{z_{m_1}}.x_2(\overline{s^1_{n_2}}), .., \lambda\overline{z_{m_k}}.x_k(\overline{t^k_{n_k}}) \doteq \lambda\overline{z_{m_k}}.C[x_1(\overline{s^k_{n_1}})] \tag{4.9}$$

where:

- the terms t^i_j for $0 < i \leq k$ and $0 < j \leq n_i$ can contain bound variables z_l for $0 < l \leq m_i$.
- the terms s^i_j for $0 < i \leq k-1$ and $0 < j \leq n_{i+1}$ can contain bound variables z_l for $0 < l \leq m_i$.
- the terms s^k_j for $0 < j \leq n_1$ can contain bound variables z_l for $0 < l \leq m_k$.

Please note that in order to simplify the presentation we are making an exception and are allowing variables capture for the variables mentioned above, i.e. context applications of the form $\lambda z.([.])z$ are allowed.

Regular terms

In order to justify the added complexity of the regular bounds when dealing with bounded higher-order systems, we will give an informal discussion of how it is constructed.

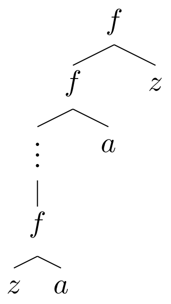
In the previous chapter the statement which said that cyclic context variables must be mapped to contexts was true by the definition of context systems. When considering bounded higher-order systems, variables can be mapped to any term.

We have also proved that for every cycle in a context unification problem, we can generate a finite set of regular contexts such that for every pre-unifier of the system, there is a variable in the cycle which can be described by one of them. Can we do the same for bounded higher-order systems.

The naive answer is clearly no - regular contexts cannot describe arbitrary higher-order terms. But we can use them in somewhat a different way. Consider the following standard cycle:

$$x^{\hat{b}=3}y \doteq f(xa, w) \tag{4.10}$$

which is unifiable by $[\lambda z.f(z, z)/x, f(a, a)/y, f(a, a)/w)]$ but also by $[\lambda z.f(f(z, a), z)/x, f(a, a)/y, f(a, a)/w)]$. It is easily seen that system 4.10 is unifiable by any substitution of the form $[\lambda z.t/x, f(a, a)/y, f(a, a)/w)]$ where t can be described by:



How can we use regular contexts in order to describe the infinitely-many unifiers above? Consider the term $\lambda z.f(C(z), z)$ where C is the regular context $(f([.], a))^*$ which can be instantiated into any of the terms defined above. It might seem that our choice of $\lambda z.f(C(z), z)$ was arbitrary and indeed it turns out it is only one set of terms described by

$$\lambda z.(f([.], w))^*(f([.], z))(f([.], a))^*(f(z, a)) \tag{4.11}$$

which, together with the fixed values for y and w, covers all possible unifiers with $\lambda\texttt{size}_r = 3$. It is pretty obvious that if their $\lambda\texttt{size}_r$ are big, the terms can become very complex. But it turns out we do not need to compute complex terms on the spot. If we never increase the bounding measure (see Def. 4.2.1), then

$$x \mapsto \lambda z.(f([.], w))^* f(x_1(z), x_2(z)) \tag{4.12}$$

where $\hat{b}(x_1), \hat{b}(x_2) < \hat{b}(x)$ is enough to describe all possible mappings of $\lambda\texttt{size}_r > 2$ and also decreases the measure. Since all possible mappings of $\lambda\texttt{size}_r = 2$ can be described by regular contexts and since mappings of $\lambda\texttt{size}_r = 1$ cannot unify cycles (can easily seen by checking the depths of the two terms), we get a complete description of all possible pre-unifiers of system 4.10.

The last argument might require some additional explanation. Why can we assume that both $\hat{b}(x_1)$ and $\hat{b}(x_2)$ are smaller than $\hat{b}(x)$? Assume $\hat{b}(x_1) = \hat{b}(x)$, then $\hat{b}(x_2) = 1$ which means, in our case, that it must be mapped to w. But as we already described this case, we can assume that $\hat{b}(x_2) > 1$. Similarly, $\hat{b}(x_1) > 1$ as otherwise the problem is not unifiable. Therefore both are smaller than $\hat{b}(x)$.

The general idea would be to describe all possible regular contexts up to a non-unary symbol which must contain at least two arguments with $\hat{b} > \texttt{ar}$.

Definition 4.2.6 (Regular terms)**.** The set $\texttt{Term}_r^n$ for a given arity n is defined as $\lambda\overline{z_n}.C_r(t)$ where:

- C_r is a regular context.
- the bound variables z_i for $0 < i \leq n$ cannot occur in the scope of starred sub-contexts of C_r.
- t is a term which may contain bound variables.

Example 4.2.7. *The computed term from the discussion above:*

$$\lambda z.(f([.], w))^*(f([.], z))(f([.], a))^*(f(z, a)) \tag{4.13}$$

is a regular term.

The following definition will capture the form of mappings of cyclic variables when the standard cycle is of one constraint only.

Definition 4.2.8 (reg)**.** Given a variable x and a pure context C, the finite set of regular terms is defined as $\mathtt{reg}(x, C) = \{\lambda \overline{z_m}.C^* C'(t(\overline{z_m}))\}$ such that:

1. $C = C' f(v_1, .., C''_{@k}, .., v_n)$ for some $C'', f, v_1, .., v_n$.
2. we have one of the following:
   a) $t \in \mathtt{PB}^b(f, \tau(x))$ and $\hat{b}(x_i) < \hat{b}(x)$ for all new variables x_i which were introduced by the function $\mathtt{PB}^b$.
   b) there is $0 < i \leq m$ such that $t \in \mathtt{PB}^b(i, \tau(x))$.

For a given context and a variable, the following infinite set contains all the instantiations of their regular terms.

Definition 4.2.9 (Instantiations of regular terms ($\mathtt{insts_1}$))**.** Given a variable x and a context C, then we define the infinite set $\mathtt{insts_1}(x, C) = \{t' | t \in \mathtt{reg}(x, C)\}$ such that:

- t' is obtained from t by replacing each Kleene star $*$ with some $k \geq 0$.

An important proof idea which is used in all the following proofs and is taken from [74] is that of a maximal context with no bound variables occurrences.

Definition 4.2.10 (Maximal contexts)**.** Let x be a variable and σ a substitution, then D is called a maximal context of $\sigma(x)$ if D is a maximal prefix of $\sigma(x)$ such that it does not contain bound variable occurrences. If there is a context C such that D is of the form $(C\sigma)^l C'$ for some $l \geq 0$ and C' a prefix of $C\sigma$ then D is called a maximal context of $\sigma(x)$ for C.

Example 4.2.11. *Let* $\sigma = [\lambda z.f(g(f(z, f(z, a))), f(a, a))/x_1,\ f(a, a)/y]$ *then the maximal context for* x *and* σ *is* $f(g([.]), f(a, a))$*. This is a maximal context for the context* $f(g([.]), y)$ *but not for* $f([.], f(a, a))$ *or* $f(g(f([.], a)), f(a, a))$.

The above definition will allow us to apply the same proof techniques which we used in Chap. 3 for context unification problems also for bounded unification problems. This is done by separating a term into a context component and an arbitrary term and proving the results over the context component which, being maximal, contains certain properties.

The first property we will prove for the case $\sigma(x)$ has a maximal context for a context C is that $\sigma(x)$ is subsumed by $\mathtt{insts_1}(x, C)$ (see Def. 2.1.18).

Lemma 4.2.12. Let $\lambda\overline{z_{m_1}}.x_1(\overline{t^1_{n_1}}) \doteq \lambda\overline{z_{m_1}}.x_2(\overline{s^1_{n_2}}), .., \lambda\overline{z_{m_k}}.x_k(\overline{t^k_{n_k}}) \doteq \lambda\overline{z_{m_k}}.C[x_1(\overline{s^k_{n_1}})]$, be a pure cycle in system S with cycle context C and σ a pre-unifier for S. Let $\sigma(x_i) = \lambda\overline{z_n}.Dt$ for some index $0 < i \leq k$, a context D and a term t. If D is a maximal context for C then $\sigma(x_i) \in_s \texttt{instsı}(x_i, C)$.

Proof. Since D is a maximal context for C, we have $D = (C^l C')\sigma$ for $l \geq 0$. We need now to show that the requirements of Def. 4.2.8 hold. The first requirement holds by assumption. We now consider the following three cases:

- $t = f(v_1, .., v_r)$ and there are at least two indices $0 < i_1, i_2 \leq r$ such that $\lambda\texttt{size}_r(v_{i_1})$, $\lambda\texttt{size}_r(v_{i_2}) > 0$. Let $t' = \texttt{PB}^b(f, \tau(x_i))$ such that $t'\theta = \lambda\overline{z_n}.t$ for some θ, then t' must contain at least two new variables, y_{i_1}, y_{i_2}, such that $\hat{b}(y_{i_1}), \hat{b}(y_{i_2}) > n$ and we satisfy requirement 2a. Therefore, $\lambda\overline{z_n}.Dt'(\overline{z_n}) \in \texttt{instsı}(x_i, C)$ and $\sigma(x_i) \in_s \texttt{instsı}(x_i, C)$.
- $\texttt{hd}(t) = z_r$ for $0 < r \leq k$ and let $t' \in \texttt{PB}^b(r, \tau(t))$ such that $t'\theta = \lambda\overline{z_n}.t$ for some θ and we satisfy requirement 2b and have $\sigma(x_i) \in_s \texttt{instsı}(x_i, C)$.
- otherwise $t = f(v_1, .., v_r)$ where we have only one index $0 < j \leq r$ such that $\lambda\texttt{size}_r(v_j) > 0$. In this case D is not maximal as it should include f as well and we get a contradiction.

Note that we did not consider the case $\lambda\texttt{size}_r(t_i) = 0$ for all $0 < j \leq \texttt{ar}(f^0)$. A simple counting of the symbols occurring on each side of the unification constraint will give us that σ is not a pre-unifier in this case. $\square$

We can now prove the corresponding lemma to Lemma 3.2.6.

Lemma 4.2.13. Given a pure standard cycle $\{\lambda\overline{y_m}.x(\overline{t_n}) \doteq \lambda\overline{y_m}.C[x(\overline{s_n})]\}$ in system S, then for every ground unifier σ of S, $\sigma(x) \in_s \texttt{instsı}(x, C)$.

Proof. Let $\sigma(x) = \lambda\overline{z_n}.C^0[t]$ such that C^0 is the maximal context. Let k be the depth of $\sigma(x)$ and let A be the greatest common prefix of $(C^{k+1})\sigma$ and C^0. Then $A = (C\sigma)^l(C')$ for some $l \leq k$ where C' is a proper prefix of $C\sigma$ and let C'' be a context such that $C'(C'') = C\sigma$. We will first prove, in a similar way we did in the proof of Lemma 3.2.6, that $A = C^0$. Assume otherwise, then A must be a proper prefix of C^0 and therefore, $C^0 = A(f(t_1, .., D_{@k}, .., t_n))$ for some context D where $A(f(t_1, .., [.]_{@k}, .., t_n))$ is not a prefix of $(C^{k+1})\sigma$. Applying σ to the unification constraint, we get:

$$\lambda\overline{y_m}.A(f(t_1, .., t'_k, .., t_n)) = \lambda\overline{y_m}.(C\sigma)(A(f(t_1, .., t''_k, .., t_n))),$$

where t'_k, t''_k are terms. Applying `(Decomp)`s we get:

$$\lambda\overline{y_m}.f(t_1, .., t'_k, .., t_n) \doteq \lambda\overline{y_m}.C''(C'(f(t_1, .., t''_k, .., t_n)))$$

Now we consider the head component of the main path of C''. If it is k, then we get a contradiction to the maximality of A as σ is a ground unifier and A should include f as well. Otherwise, it is $l \neq k$ and let $C''[.] = f(s_1, .., D'_{@l}, .., s_n)$, then we get from the constraint, after one `(Decomp)`, that

$$t_l \doteq D'(C'(f(t_1,..,t_l,..,t_n)))$$

which is a contradiction to the unifiability of the pair as the positions of the holes in D' and C' are rigid (according to the definition of standard cycles). We therefore assume that $A = C^0$ and by Lem. 4.2.12, we get that $\sigma(x) \in_s \mathtt{insts}_1(x, C)$ $\square$

When we consider cycles containing more than one constraint, our first consideration will be to check if the more general description of unifiers given in Def. 3.2.11 is enough to describe the mappings of variables in these cycles as well.

Consider the system :

$$\{x_1^{\hat{b}=3} f(a,a) \doteq x_2^{\hat{b}=3} a, x_2 b \doteq f(x_1 a, f(b,b))\} \tag{4.14}$$

and its unifier σ

$$[\lambda z.f(f(z,z), f(a,a))/x_1, \lambda z.f(f(f(a,a), f(a,a)), f(z,z))/x_2] \tag{4.15}$$

Is it true that $\sigma(x_i) \in_s \mathtt{insts}_1(x, f([.], f(b,b)))$ for some $0 < i \leq 2$? Let $f([.], f(b,b)) = C$, then in order for it to be true $\sigma(x_i)$ must be of the form $\lambda z.Dt$ for some context D and a term t such that (According to Def. 4.2.8):

- D is of the form $C^k C'$ for C' a prefix of C.
- D does not contain bound variables.
- t is either a flex term or it is a rigid term such that at least two immediate subterms contain bound variables.

Clearly both $\sigma(x_1)$ and $\sigma(x_2)$ fail to satisfy all these requirements. Since σ is a bound unifier, we must extend our definition on $\mathtt{insts}$ in order to maintain completeness.

The reason for the fact that σ is a bounded unifier but none of its mappings is described by $\mathtt{insts}_1$ can be traced back to the previous chapter. When considering the positions of the bound variables in $\sigma(x_1)$ and $\sigma(x_2)$ and the position of the hole in the cycle context, we notice that the position of the hole in the cycle's context is at position 1 but there is a bound variable at position 1.1.2 in $\sigma(x_1)$ and at positions 1.2.1 and 1.2.2 in $\sigma(x_2)$. This is the same derailing we had in the previous chapter and the treatment will be the same as well.

Example 4.2.14. *Let (S, b) be system 4.14 and let $f([.], y) \in \mathtt{one\text{-}derail}(f([.], f(b,b)))$. We see now that $\sigma(x_1)$ is indeed subsumed by $\mathtt{insts}_1(x_1, f([.], y))$ (consider the substitution $[f(a,a)/y]$).*

We can now change the definition of the iterated derailing.

Definition 4.2.15 (Iterated derailing)**.** Let $\lambda\overline{z_{m_1}}.x_1(\overline{t^1_{n_1}}) \doteq \lambda\overline{z_{m_1}}.x_2(\overline{s^1_{n_2}}), .., \lambda\overline{z_{m_k}}.x_k(\overline{t^k_{n_k}}) \doteq \lambda\overline{z_{m_k}}.C[x_1(\overline{s^k_{n_1}})]$ be a standard cycle with a context C, then the set $\mathtt{derail}(x_i, m, C)$ of the iterated derailed regular terms for the cycle context C and an index $0 < i \leq k$ is defined as follows:

- if $m = 1$ then $\texttt{derail}(x_1, 1, C) = \texttt{reg}(x_1, C)$.
- if $m > 1$ then $\texttt{derail}(x_i, m, C) = \{\lambda\overline{z_{m_i}}.C^*(D_{pre}(C^r)) \mid$
  $D \in \texttt{one-derail}(C), \lambda\overline{z_{m_i}}.C^r \in \texttt{derail}(m-1, D_{post}(D_{pre}))\}$

Definition 4.2.16 (Instantiations of regular terms)**.** Let C be a context and x a variable, then the infinite set $\texttt{insts}(x, C, m) = \{t' | t \in \texttt{derail}(x, m, C)\}$ such that t' is obtained from t by replacing each of the n occurrences of the Kleene stars in t with the natural numbers $k_1, .., k_n$ respectively.

Next, we will prove some properties of instantiations of regular terms.

The first property is that if a term is subsumed by an instantiation using n iterations, then it also does so using $m > n$ iterations as well.

Lemma 4.2.17. Given a system S containing a pure standard cycle with a cycle context C and variable x and assume that for some pre-unifier σ of S, $\sigma(x) \in_s \texttt{insts}(x, C, l)$ for $l > 0$ then $\sigma(x) \in_s \texttt{insts}(x, C, k)$ for all $k > l$.

Proof. For the corresponding iterations we replace the Kleene star with 0 and choose an empty prefix. □

The next property asserts that any term subsumed by $\texttt{insts}_1$ is also subsumed by $\texttt{insts}$ after one iteration.

Lemma 4.2.18. Given a system S containing a pure standard cycle with a cycle context C and let σ be a pre-unifier of S, if $\sigma(x) \in_s \texttt{insts}_1(x, C)$ then $\sigma(x) \in_s \texttt{insts}(x, C, 1)$.

Proof. Clear from the definitions of $\texttt{insts}$ and $\texttt{insts}_1$. □

The following lemma unfolds one iteration and relates terms obtained using different iterations of $\texttt{insts}$.

Lemma 4.2.19. Let C be a context, C' and C'' contexts such that $C'C'' = C$ and $C'' = f(v_1, .., D'_{@k}, .., v_n)$. Assume that:

- $\lambda\overline{z_m}.t_0 \in_s \texttt{insts}(x, D'C'f(y_1, .., [.], .., y_n), m)$ for some variable x and
- $\lambda\overline{z_m}.t_1 \geq_s \lambda\overline{z_m}.(C^lC'f(y_1, .., t_0, .., y_n))$ such that C^lC' does not contain any bound variable.

Then, $\lambda\overline{z_m}.t_1 \in_s \texttt{insts}(x, C, m+1)$.

Proof. From the assumptions we know that there are substitutions θ_1 and θ_2 and a term t' such that:

- $t' \in \texttt{insts}(x, D'C'f(y_1, .., [.], .., y_n), m)$.
- $\lambda\overline{z_m}.t_0 = t'\theta_1$.

- $\lambda\overline{z_m}.t_1 = (\lambda\overline{z_m}.(C^l C' f(y_1, .., t_0, .., y_n)))\theta_2$.

Furthermore, since $t' \in \texttt{insts}(x, D'C'f(y_1, .., [.], .., y_n), m)$, we know that there is a regular term $\lambda\overline{z_m}.t_0^1 \in \texttt{derail}(x, m, D'C'f(y_1, .., [.], .., y_n))$. Let $D_{pre} = C'f(y_1, .., [.], .., y_n)$ and $D_{post} = D'$, then $D_{pre}D_{post} \in \texttt{one-derail}(C)$ and $\lambda\overline{z_m}.C^*C'f(y_1, .., t_0^1, .., y_n) \in \texttt{derail}(x, m+1, C)$. Let $t'' = \lambda\overline{z_m}.(C^l C' f(y_1, .., t', .., y_n))$, then $t'' \in \texttt{insts}(x, C, m+1)$. We now have that $t_1 = t''\theta_2$ and therefore, that $t_1 \in_s \texttt{insts}(x, C, m+1)$. $\square$

We can now prove the main lemma in this section. The next lemma is the corresponding one to Lemma 3.2.14 and formalize the argument that we can describe all possible mappings of cyclic variables. Like in Lemma 3.2.14, we prove the result for one variable in the cycle but unlike Lemma 3.2.14, we do not give a precise description but approximate it enough, such that we can show the approximation has a lower bounding measure. We would like to note first that an important proof technique which will be used in the following proof is to replace terms by variables and then apply rules from the algorithm $\texttt{PUA}_B$. These two steps are possible since:

- if a system S is unifiable by σ and we replace terms $t_1, .., t_n$ in S by fresh variables $y_1, .., y_n$ in order to obtain the system S' then S' is unifiable by $\sigma \circ [t_1/y_1, .., t_n/y_n]$.
- if a system S is unifiable by σ, then we can apply (Delete), (Decomp) or (Bind) in order to obtain a system S' such that S' is unifiable by σ as these three rules do not affect the set of unifiers.

Lemma 4.2.20. Given a system S containing a pure standard cycle $\overline{\lambda\overline{z_{m_1}}.x_1(\overline{t^1_{n_1}}) \doteq \lambda\overline{z_{m_1}}.x_2(\overline{s^1_{n_2}}), .., \lambda\overline{z_{m_k}}.x_k(\overline{t^k_{n_k}}) \doteq \lambda\overline{z_{m_k}}.C[x_1(\overline{s^k_{n_1}})]}$, then for any ground unifier σ of S there is an index $0 < i \leq n$ such that $\sigma(x_i) \in_s \texttt{insts}(x_i, C, k)$.

Proof. First we compute the maximal contexts D_i of $\sigma(x_i)$ for $0 < i \leq k$. Let A be the greatest common prefix of D_i and $(\sigma(C))^h$ for $0 < i \leq k$ where h is the minimal depth of all D_i. Then, $A = \sigma(C)^q(C')$ for $q \leq h$ where C' is a proper prefix of $\sigma(C)$ and let C'' be a context such that $C'(C'') = \sigma(C)$. By induction on the number of constraints in the cycle.

- for $k = 1$ we first use Lemma 4.2.13 in order to obtain that $\sigma(x_1) \in_s \texttt{insts}_1(x_1, C)$ and then Lemma 4.2.18 in order to show that $\sigma(x_1) \in_s \texttt{insts}(x_1, C, 1)$.
- for $k > 0$, if there is $0 < i \leq k$ such that $D_i = A$, then we can use lemmas 4.2.12 and 4.2.18 in order to prove that $\sigma(x_i) \in_s \texttt{insts}(x_i, C, 1)$ and by using Lemma 4.2.17 we have $\sigma(x_i) \in_s \texttt{insts}(x_i, C, k)$. We now assume that $D_i \neq A$ for all $0 < i \leq k$, i.e. that $D_i = Af(v_1^i, .., v_n^i)$ for all $0 < i \leq k$. Clearly, there are at least two terms v_i^q and v_j^r for $0 < i, j \leq n$, $0 < q, r \leq k$ and $i \neq j$ such that both contain bound variables as otherwise f will be common to all D_i for $0 < i \leq k$ (not that σ is a ground unifier). Now consider two cases:
  - there is an index $0 < p \leq k$ such that v_i^p and v_j^p contain bound variables for $i \neq j$. In this case $\sigma(x_p)$ can be written as $\lambda\overline{z_{n_p}}.Af(\theta(x'_1)(\overline{z_{n_p}}), .., \theta(x'_n)(\overline{z_{n_p}}))$ for $\theta = [v_1^p/x'_1, .., v_n^p/x'_n]$ such that $\hat{b}(x'_i), \hat{b}(x'_j) < \hat{b}(x_p)$ and therefore we satisfy

requirement 2a in Def. 4.2.8 and have that $\sigma(x_p) \in_s \texttt{insts}_1(x_p, C)$. We can now use lemmas 4.2.18 and 4.2.17 to have $\sigma(x_p) \in_s \texttt{insts}(x_p, C, k)$.

– in this case for each $0 < i \leq k$, there is only one index $0 < l_i \leq n$ such that $v^i_{l_i}$ contains bound variables and v^i_j does not contain any bound variable for $0 < j \leq n$ and $j \neq l_i$. The standard cycle, after the application of σ, can now be represented as

$$\lambda\overline{z_{m_1}}.Af(v^1_1, .., D^1_{@l_1}(\overline{t^1_{n_1}}), .., v^1_n) \doteq \lambda\overline{z_{m_1}}.Af(v^2_1, .., D^2_{@l_2}(\overline{s^1_{n_2}}), .., t^2_n), ..,$$
$$\lambda\overline{z_{m_k}}.Af(v^k_1, .., D^k_{@l_k}(\overline{t^k_{n_k}}), .., v^k_n) \doteq \lambda\overline{z_{m_k}}.C(Af(v^1_1, .., D^1_{@l_1}(\overline{s^k_{n_1}}), .., v^1_n))$$

where $v^i_{l_i} = \lambda\overline{z_{n_i}}.D^i(\overline{z_{n_i}})$ are new terms and are the only terms containing bound variables. We can now follow the proof for Lemma 3.2.14.
Let $C'' = f(t_1, .., D'_{@l}, .., t_n)$ and let $I = \{i_1, .., i_p\}$ contain all the indices $0 < i \leq k$ such that $l_i = l$. (i.e. the head component of the position of the hole in f is l). Consider now the cycle after the applications of the `(Decomp)` rule only:

$$\lambda\overline{z_{m_1}}.u_1 \doteq \lambda\overline{z_{m_1}}.u'_1, .., \lambda\overline{z_{m_k}}.u_k \doteq \lambda\overline{z_{m_k}}.u'_k \tag{4.16}$$

where:

* for every $0 < i \leq k$, $u_i = D^i(\overline{t^i_{n_i}})$ if $i \in I$ and $u_i = v^i_l$ otherwise.
* for every $0 < i < k$, $u'_i = D^{i+1}(\overline{s^i_{n_{i+1}}})$ if $i + 1 \in I$ and $u'_i = v^{i+1}_l$ otherwise.
* $u'_k = D'C'f(v^1_1, .., D^1_{@l_1}(\overline{s^k_{n_1}}), .., v^1_n)$.

Now let us consider the substitution δ such that $\delta(y_i) = v^i_l$ for $0 < i \leq k$ where y_i are fresh variables. and consider the equations

$$\lambda\overline{z_{m_1}}.r_1 \doteq \lambda\overline{z_{m_1}}.r'_1, .., \lambda\overline{z_{m_k}}.r_k \doteq \lambda\overline{z_{m_k}}.r'_k \tag{4.17}$$

where:

* for every $0 < i \leq k$, $r_i = D^i(\overline{t^i_{n_i}})$ if $i \in I$ and $r_i = y_i$ otherwise.
* for every $0 < i < k$, $r'_i = D^{i+1}(\overline{s^i_{n_{i+1}}})$ if $i + 1 \in I$ and $r'_i = y_{i+1}$ otherwise.
* $r'_k = D'C'f(v^1_1, .., D^1_{@l_1}(\overline{s^k_{n_1}}), .., v^1_n)$.

Clearly, this cycle is pre-unifiable by $\sigma \circ \delta$. After applying `(Bind)` on all equations containing y_i, we get:

$$\lambda\overline{z_{m_{i_1}}}.D^{i_1}(\overline{t^{i_1}_{n_{i_1}}}) \doteq \lambda\overline{z_{m_{i_1}}}.D^{i_2}(\overline{s^{i_1}_{n_{i_2}}}), ..,$$
$$\lambda\overline{z_{m_{i_p}}}.D^{i_p}(\overline{t^{i_p}_{n_{i_p}}}) \doteq \lambda\overline{z_{m_{i_p}}}.D'C'f(v^1_1, .., D^{i_1}_{@l}(\overline{s^{i_p}_{n_{i_1}}}), .., v^1_n) \tag{4.18}$$

using the permutations of the arguments as was done in the proof of Lemma 3.2.14. Let us take now the substitution θ such that $\theta(w^{i_j}) = \lambda\overline{z_{n_{i_j}}}.D^{i_j}(\overline{z_{n_{i_j}}})$ for $0 < j \leq p$ where w^{i_j} are fresh variables and consider the standard cycle:

$$\lambda\overline{z_{m_{i_1}}}.w^{i_1}(\overline{t^{i_1}_{n_{i_1}}}) \doteq \lambda\overline{z_{m_{i_1}}}.w^{i_2}(\overline{s^{i_1}_{n_{i_2}}}), ..,$$
$$\lambda\overline{z_{m_{i_p}}}.w^{i_p}(\overline{t^{i_p}_{n_{i_p}}}) \doteq \lambda\overline{z_{m_{i_p}}}.D'C'f(v^1_1, .., w^{i_1}_{@l}(\overline{s^{i_p}_{n_{i_1}}}), .., v^1_n) \tag{4.19}$$

which is clearly pre-unifiable by $\sigma \circ \theta$. Assume further that we have the substitution η such that $\eta(z_i) = v_i^1$ for $0 < i \leq n$, then the standard cycle:

$$\lambda\overline{z_{m_{i_1}}}.w^{i_1}(\overline{t^{i_1}_{n_{i_1}}}) \doteq \lambda\overline{z_{m_{i_1}}}.w^{i_2}(\overline{s^{i_1}_{n_{i_2}}}), .., \\ \lambda\overline{z_{m_{i_p}}}.w^{i_p}(\overline{t^{i_p}_{n_{i_p}}}) \doteq \lambda\overline{z_{m_{i_p}}}.D'C'f(z_1, .., w^{i_1}_{@l}(\overline{s^{i_p}_{n_{i_1}}}), .., z_n) \tag{4.20}$$

is pre-unifiable by $\sigma \circ \theta \circ \eta$. Since $p < k$, we can apply the induction hypothesis in order to obtain that there is and index $0 < j \leq p$ such that $\theta(w^{i_j}) = \lambda\overline{z_{n_{i_j}}}.D^{i_j}(\overline{z_{n_{i_j}}}) \in_s \mathtt{insts}(w^{i_j}, D'C'f(z_i, .., [.]_{@l}, .., z_n)), p)$. Since $\sigma(x_{i_j}) = \lambda\overline{z_{n_{i_j}}}.Af(v_1^{i_j}, .., D^{i_j}(\overline{z_{n_{i_j}}}), .., v_n^{i_j}) = \lambda\overline{z_{n_{i_j}}}.(C\sigma)^q C'f(v_1^{i_j}, .., D^{i_j}(\overline{z_{n_{i_j}}}), .., v_n^{i_j})$, we can use Lemma 4.2.19 in order to obtain that $\sigma(x_{i_j}) \in_s \mathtt{insts}(x_{i_j}, C, p+1)$ and therefore, using Lemma 4.2.17, that $\sigma(x_{i_j}) \in_s \mathtt{insts}(x_{i_j}, C, k)$.

$\square$

4.3 Pre-unification Using Regular Terms

In this section we will give an algorithm for the pre-unification of bounded higher-order problems, which will be based on $\mathtt{PUA}_B$ in the same way that $\mathtt{CUA}$ was based on $\mathtt{PUA}_C$. The definitions of bindings and depth constraints as well as the definition of environments are similar to that in Def. 4.3.1.

Since we now have also the $\hat{b}$ values in the environment, we will use a simpler notation to define, manipulate and obtain values in environments.

Definition 4.3.1 (Environments and bounding constraints)**.** A bounding constraint is an equation of the form $\hat{b}(x) = n$ and allows us to represent and monitor the $\hat{b}$ values of the unification problem using the environments. Given an environment E, we will use the following notation for denoting constraints in E:

- $E[d(x) = n]$ stands for $(\mathtt{d}(x) \leq n) \in E$.
- $E[r(x) = t]$ stands for $(x \lhd t) \in E$.
- $E[\hat{b}(x) = n]$ stands for $\hat{b}(x) = n$.
- let $\rho \in \{d, r, \hat{b}\}$ then
  - $E[\rho(x) = \epsilon]$ means there is no such constraint for x in E.
  - when we write $E[\rho(x) = u]$ we also mean the environment obtained from E by replacing the constraint ρ for x with the new one. If there was no old constraint, we just insert a new constraint into the new environment.
  - $E[\rho_1(x_1) = v_1, .., \rho_n(x_n) = v_n]$ stands for $E[\rho_1(x_1) = v_1]..[\rho_n(x_n) = v_n]$.

Since we have at most one constraint of each type for each variable in the environment, the new notation is well defined. If the environment contains a value $\hat{b}(x)$ for all higher-order variables in the clause, the environment is called a valid one.

Example 4.3.2. *By writing $E[d(x) = 4, \hat{b}(x) = 6, r(y) = \lambda z.z]$ we mean either:*

- *that E contains a depth and a bounding constraint for x and a binding constraint for y or*
- *that $E[d(x) = 4, \hat{b}(x) = 6, r(y) = \lambda z.z]$ is obtained from E by adding or replacing these constraints for x and y.*

Definition 4.3.3 (The set of rules BUA (Bounded unification algorithm))**.** Let $(S, \hat{b})$ be a unification system and E a valid environment, then the set of rules BUA are defined in Fig. 4.2. The reset(E, S) function used in the algorithm returns an environment after adding to the environment E a depth constraint containing the value $2 \cdot \texttt{fbound}(S)$ for each unsolved variable in S. The initial environment is equal to reset$(\hat{b}, S)$. The function scy returns all pure standard cycles in S.

The following lemma, which will be used in the completeness proof, confirms that cyclic variables can indeed be mapped by BUA to terms subsumed by sets obtained from the cycles' contexts (see Def. 4.2.16).

Lemma 4.3.4. Let x be a variable in a cycle in system S with cycle context C and cycle size n and let $t \in \texttt{insts}(x, C, n)$, then we can derive using BUA a system S' such that $\sigma_{S'}(x) = t$.

Proof. According to the definition of insts there is a regular term t_r corresponding to t. by applying the four cyclic rules (Skip), (Imitate$_0$), (Imitate$_*$) and at the end (Project) and by choosing the right partial bindings, we can simulate any instantiation of t_r (by insts). □

The following example shows how the unification algorithm unfolds cyclic problems:

Example 4.3.5. *Let S be the following monadic problem based on Ex. 1.3.1 (see Remark 2.1.53 about right-associativity):*

$$\{X_1abX_1bX_2bX_3bX_4c \doteq aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac\} \tag{4.21}$$

Fig. 4.3 shows how we can obtain the unifier:

$$[\lambda z.a^{81}z/X_1, \lambda z.a^{27}z/X_2, \lambda z.a^9z/X_3, \lambda z.a^3z/X_4] \tag{4.22}$$

Please note that some constraints are removed in order to simplify presentation. We also remove the bounding constraints from the environment as clearly they will be satisfied, in monadic problems, for $\hat{b}(X) > 1$ for all variables X in the problem.

$$\frac{E \vdash S \cup \{A \doteq A\}}{E \vdash S}\ \texttt{(Delete)} \qquad \frac{E \vdash S \cup \{\lambda\overline{z_k}.f(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_n})\}}{E \vdash S \cup \{\lambda\overline{z_k}.s_1 \doteq \lambda\overline{z_k}.t_1, .., \lambda\overline{z_k}.s_n \doteq \lambda\overline{z_k}.t_n\}}\ \texttt{(Decomp)}$$

$$\frac{E[r(x) = \lambda\overline{z_k}.C^*(C_r)] \vdash S}{E[r(x) = \lambda\overline{z_k}.C_r] \vdash S\}}\ \texttt{(Skip)} \qquad \frac{E \vdash S \cup \{\lambda\overline{z_k}.x(\overline{z_k}) \doteq \lambda\overline{z_k}.t\} \qquad x \notin \texttt{FV}(t), \sigma = [\lambda\overline{z_k}.t/x]}{E\sigma \vdash S\sigma \cup \{x \doteq \lambda\overline{z_k}.t\}}\ \texttt{(Bind)}$$

$$\frac{E \vdash S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m}) \in S, 0 < i \leq k, u = \texttt{PB}^b(i,\alpha), \sigma = [u/x]}{\texttt{reset}(E, S\sigma) \vdash S\sigma \cup \{x \doteq u\}}\ \texttt{(Project)}^2$$

$$\frac{E[r(x) = \epsilon] \vdash S \qquad c \in \texttt{scy}(S,E), x \in c, t \in \texttt{derail}(x, \texttt{size}(c), \texttt{ccon}(c))}{E[r(x) = t] \vdash S}\ \texttt{(Rec)}$$

$$\frac{E[d(x) = m > 0, r(x) = \epsilon] \vdash S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_p}) \in S, u = \texttt{PB}^b(f,\alpha), \sigma = [u/x]}{E[d(x_1) = m-1, .., d(x_p) = m-1]\sigma \vdash S\sigma \cup \{x \doteq u\}}\ \texttt{(Imitate}_{pb}\texttt{)}^1$$

$$\frac{E[r(x) = \lambda\overline{z_n}.t^*(tr)] \vdash S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{v_p}) \in S, t = f(t_1, .., t'([.])_{@k}, .., t_p), u = \texttt{PB}^b(f,\alpha), \sigma = [u/x]}{E[r(x_1) = t_1, .., r(x_k) = t'(t^*(tr)), .., r(x_p) = t_p]\sigma \vdash S\sigma \cup \{x \doteq u\}}\ \texttt{(Imitate}_*\texttt{)}^1$$

$$\frac{E[r(x) = \lambda\overline{z_n}.t] \vdash S \qquad \lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{v_p}) \in S, t = f(t_1, .., t_p), u = \texttt{PB}^b(f,\alpha), \sigma = [u/x]}{E[r(x_1) = t_1, .., r(x_p) = t_p]\sigma \vdash S\sigma \cup \{x \doteq u\}}\ \texttt{(Imitate}_0\texttt{)}^1$$

1. $x_1, .., x_p$ are all fresh variables introduced by $\texttt{PB}^b$.

2. a is either a function symbol or a bound variable.

Figure 4.2: BUA - Bounded unification rules

$$\begin{array}{cl}
\{d(X_{1,2,3,4}=20\} \vdash \{X_1abX_1bX_2bX_3bX_4c \doteq aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac\} & \\ \hline
\{d..,r(X_1)=\lambda z.a^*z\} \vdash \{X_1abX_1bX_2bX_3bX_4c \doteq aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac\} & \texttt{rec} \\ \hline
\{d(X_1^1)=20, d(X_{2,3,4}=20, r(X_1^1)=\lambda z.a^*z\} \vdash \{aX_1^1abaX_1^1bX_2bX_3bX_4c \doteq aaX_1^1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.aX_1^1z\} & \texttt{(Imitate}_*\texttt{)} \\ \hline
\{d..,r(X_1^1)=\lambda z.a^*z\} \vdash \{X_1^1abaX_1^1bX_2bX_3bX_4c \doteq aX_1^1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.aX_1^1z\} & \texttt{(Decomp)} \\ \hline
\{d..,r(X_1^{81})=\lambda z.a^*z\} \vdash \{X_1^{81}aba^{81}X_1^{81}bX_2bX_3bX_4c \doteq aX_1^{81}bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}X_1^{81}z\} & \texttt{(Imitate}_*\texttt{),(Decomp)}\times 80 \\ \hline
\{d..,r(X_1^{81})=\lambda z.z\} \vdash \{X_1^{81}aba^{81}X_1^{81}bX_2bX_3bX_4c \doteq aX_1^{81}bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}X_1^{81}z\} & \texttt{(Skip)} \\ \hline
\{d(X_{2,3,4}=672\} \vdash \{aba^{81}bX_2bX_3bX_4c \doteq abX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_1^{81} \doteq \lambda z.z\} & \texttt{(Project)} \\ \hline
\{d..\} \vdash \{a^{81}bX_2bX_3bX_4c \doteq X_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_1^{81} \doteq \lambda z.z\} & \texttt{(Decomp)}\times 2 \\ \hline
\{d..,r(X_2)=\lambda z.a^*z\} \vdash \{a^{81}bX_2bX_3bX_4c \doteq X_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_1^{81} \doteq \lambda z.z\} & \texttt{rec} \\ \hline
\{d(X_2^{27})=672, d(X_{3,4}=672, r(X_2^{27})=\lambda z.a^*z\} \vdash \{a^{54}ba^{27}X_2^{27}bX_3bX_4c \doteq X_2^{27}a^{27}X_2^{27}a^{27}X_2^{27}bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq a^{27}X^{27}z\} & \texttt{(Imitate}_*\texttt{),(Decomp)}\times 27 \\ \hline
\{d..,r(X_2^{27})=\lambda z.z\} \vdash \{a^{54}ba^{27}X_2^{27}bX_3bX_4c \doteq X_2^{27}a^{27}X_2^{27}a^{27}X_2^{27}bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}X^{27}z\} & \texttt{(Skip)} \\ \hline
\{d(X_{3,4})=504\} \vdash \{a^{54}ba^{27}bX_3bX_4c \doteq a^{27}a^{27}bX_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z\} & \texttt{(Project)} \\ \hline
\{d..\} \vdash \{a^{27}bX_3bX_4c \doteq X_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z\} & \texttt{(Decomp)}\times 55 \\ \hline
\{d..,r(X_3)=\lambda z.a^*z\} \vdash \{a^{27}bX_3bX_4c \doteq X_3X_3X_3bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z\} & \texttt{rec} \\ \hline
\{d(X_3^9)=504, d(X_4)=504, r^9(X_3)=\lambda z.a^*z\} \vdash \{a^{18}ba^9X_3^9bX_4c \doteq X_3^9a^9X_3^9a^9X_3^9bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9X_3^9z\} & \texttt{(Imitate}_*\texttt{),(Decomp)}\times 9 \\ \hline
\{d..,r^9(X_3)=\lambda z.z\} \vdash \{a^{18}ba^9X_3^9bX_4c \doteq X_3^9a^9X_3^9a^9X_3^9bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9X_3^9z\} & \texttt{(Skip)} \\ \hline
\{d(X_4)=120\} \vdash \{a^{18}ba^9bX_4c \doteq a^9a^9bX_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z\} & \texttt{(Project)} \\ \hline
\{d..\} \vdash \{a^9bX_4c \doteq X_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z\} & \texttt{(Decomp)}\times 19 \\ \hline
\{d..,r(X_4)=\lambda z.a^*z\} \vdash \{a^9bX_4c \doteq X_4X_4X_4baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z\} & \texttt{rec} \\ \hline
\{d(X_4^3)=120, r(X_4^3)=\lambda z.a^*z\} \vdash \{a^6ba^3X_4^3c \doteq X_4^3a^3X_4^3a^3X_4^3baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z, X_4 \doteq \lambda z.a^3X_4^3z\} & \texttt{(Imitate}_*\texttt{),(Decomp)}\times 3 \\ \hline
\{d(X_4^3)=120, r(X_4^3)=\lambda z.z\} \vdash \{a^6ba^3X_4^3c \doteq X_4^3a^3X_4^3a^3X_4^3baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z, X_4 \doteq \lambda z.a^3X_4^3z\} & \texttt{(Skip)} \\ \hline
\emptyset \vdash \{a^6ba^3c \doteq a^3a^3baaac, X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z, X_4 \doteq \lambda z.a^3z\} & \texttt{(Project)} \\ \hline
\emptyset \vdash \{X_1 \doteq \lambda z.a^{81}z, X_2 \doteq \lambda z.a^{27}z, X_3 \doteq \lambda z.a^9z, X_4 \doteq \lambda z.a^3z\} & \texttt{(Delete)}
\end{array}$$

Figure 4.3: Running `BUA` on Eq. 4.21

4.4 Soundness and Completeness

In this section we will prove the soundness and completeness of BUA with regard to $\texttt{PUA}_B$. Since $\texttt{PUA}_B$ was proved to enumerate all pre-unifiers, the correctness proofs will mean that BUA enumerates all pre-unifiers as well.

The proofs of the majority of the lemmas in this section are straightforward adaption to higher-order logic of proofs for lemmas from Sec. 3.4.

Theorem 4.4.1 (Soundness)**.** If S' is obtained from a unification system S using BUA and is in pre-solved form then $\sigma_{S'}|_{\texttt{FV}(S)} \in \texttt{PreUnifiers}(S)$.

Proof. The rules in BUA are the same as the rules in $\texttt{PUA}_B$ but pose more restrictions on the generated substitutions. First, we restrict the depth of terms using the depth constraints and second, we generated partial bindings which are less general than the ones generated in $\texttt{PUA}_B$ due to the use of the binding constraints. Formally, we will show that for each pre-solved form S which is obtained using BUA, there is a pre-solved form S' which can be obtained using $\texttt{PUA}_B$ such that $\sigma_{S'} \leq \sigma_S$. The simulation of the rules (Delete), (Decomp), (Bind) and (Project) is straightforward. The simulation of the different imitation rules is done by simply replacing them with the single (Imitate) rule in $\texttt{PUA}_B$. Since it generates a partial binding which is equal to the one generated by $(\texttt{Imitate}_{pb})$ and is more general than the ones generated by $(\texttt{Imitate}_{*})$ and $(\texttt{Imitate}_{0})$, the substitution is more general as well. The (Skip) and (Rec) rules can be ignored completely as they affect the environment only. □

We first prove that in acyclic systems the depth of terms mapped to variables is bound by fbound.

Lemma 4.4.2. Let S be a system such that all variables in it are acyclic according to $<_c$ and x be a variable in S. Then, for any system S' obtainable from S by using $\texttt{PUA}_B$ without the application of a (Project), we have $\texttt{d}(\sigma_{S'}(x)) \leq \texttt{fbound}(S)$.

Proof. We follow the proof of Lemma 3.4.6. □

Lemma 4.4.3. If σ unifies a system S, then it unifies a problem restriction S' of S.

Proof. We follow the proof of Lemma 3.4.11. □

The next lemma states that if a variable is mapped in some pre-unifier to a large term, then this variable or a smaller one can be related to a standard cycle. In order to prove it we will use the same relation from Def. 3.4.2 as well as the repeated variables from Def. 3.4.4.

Lemma 4.4.4. If a variable x in a system S is repeated, then it is either cyclic or there is a smaller variable in S, according to $<_c$, which is cyclic.

Proof. We follow the proof of Lemma 3.4.12. □

The next lemmas show that if we have a cyclic variable, a standard cycle can be obtained using BUA.

Lemma 4.4.5. Given a system S with environment E and assume S contains a cyclic variable and for all unsolved higher-order variables $x \in \mathtt{FV}(S)$ there is $v \geq \mathtt{fbound}(S)$ such that $(\mathtt{d}(x) \leq v) \in E$, then we can obtain a system S' and environment E' using the rules `(Delete)`, `(Decomp)`, `(Bind)` and $(\mathtt{Imitate}_{pb})$ such that S' contains a cycle over the variables $x_1, ., , x_n$ and $(\mathtt{d}(x_i) \leq v) \in E'$ for $0 < i \leq n$.

Proof. We follow the proof of Lemma 3.4.13. □

Lemma 4.4.6. Given a system S with environment E and assume S contains a pure cycle $\lambda\overline{z_{m_1}}.x_1(\overline{t^1_{n_1}}) \doteq \lambda\overline{z_{m_1}}.t_1, .., \lambda\overline{z_{m_k}}.x_k(\overline{t^k_{n_k}}) \doteq \lambda\overline{z_{m_k}}.t_k$, and assume further that for all $0 < i \leq k$ there is $v_i \geq \mathtt{fbound}(S)$ such that$(\mathtt{d}(x_i) \leq v_i) \in E$, then we can obtain a system S' and environment E' using the rules `(Delete)`, `(Decomp)`, `(Bind)` and $(\mathtt{Imitate}_{pb})$ such that S' contains a pure non-unique standard cycle over the variables $y_1, .., y_k$ and such that $(\mathtt{d}(y_i) \leq u_i) \in E'$ for $0 < i \leq k$ and $u_i > \mathtt{md}(S)$.

Proof. We prove this by induction on $l = \Sigma_{i=1}^{k-1} i * M_i$ where M_i is the size of the minimal position of x_{i+1} in t_i for $0 < i \leq k-1$. If $l = 0$, then we are done as $X_1 \doteq t_1$ cannot be a flex-flex constraint (see next) and we already have a non-unique standard cycle. If $l > 0$, then we apply $(\mathtt{Imitate}_{pb})$ on an equation $\lambda\overline{z_{m_j}}.x_j(\overline{t^j_{n_j}}) \doteq \lambda\overline{z_{m_j}}.t_j$ with $0 < j < m$ maximal such that $\mathtt{hd}(t_j) \notin \mathfrak{V}$. Assume further that $t_j = f(t'_1, .., t'_p)$ and that x_{j+1} occurs in t'_q for $0 < q \leq p$. The result, after applying `(Decomp)`, is again a cycle with a new variable x'_j instead of x_j. l is decreased in the new cycle as either $j > 1$ and then we get that M_j is decreased by 1 and M_{j-1} is increased by 1, or $j = 1$ and then M_1 is decreased by 1. In the second case, M_m is increased but we don't count it. Therefore, we can apply the induction hypothesis in order to obtain a non-unique standard cycle. The reason E' is as above is that we apply at most l $(\mathtt{Imitate}_{pb})$ steps and each one of these steps decreases one depth constraint by 1, so at worst case one constraint will be decreased by l. As we assumed the constraints to be of the form $\mathtt{d}(x_i) \geq v_i$ before we start, we will obtain, in the worst case, one constraint of the form $\mathtt{d}(x'_i) \leq v$ with $v \geq \mathtt{fbound}(S) - l$. Since $M_i \leq \mathtt{md}(S)$ for $0 < i \leq k$, $\mathtt{fbound}(S) = (K+1) \cdot \mathtt{md}(S)$ $(K = \mathtt{size}(\mathtt{FV}(S)))$ and $k \leq K$ we obtain that $v > \mathtt{md}(S)$. □

Lemma 4.4.7. Given a system S with environment E and assume S contains a pure non-unique standard cycle $\lambda\overline{z_{m_1}}.x_1(\overline{t^1_{n_1}}) \doteq \lambda\overline{z_{m_1}}.x_2(\overline{s^1_{n_2}}), .., \lambda\overline{z_{m_k}}.x_k(\overline{t^k_{n_k}}) \doteq \lambda\overline{z_{m_k}}.t_k$, and assume further that $(\mathtt{d}(x_i) \leq v) \in E$ for $0 < i \leq k$ and $v > v_0$ where v_0 is the size of the minimal position of x_1 in t_k, then we can obtain a system S' with a pure (unique) standard cycle using the rules `(Delete)`, `(Decomp)`, `(Bind)` and $(\mathtt{Imitate}_{pb})$.

Proof. First, if the non-unique standard cycle is also standard, then we are done. Otherwise, let p_m be the minimal position in t_k of x_1. The way to achieve a standard cycle is similar to what was done in the proof of the previous lemma. By applying $\mathtt{size}(p_m)$ times the rule $(\mathtt{Imitate}_{pb})$ on the last equation we will obtain a standard cycle. Applying the rule $\mathtt{size}(p_m)$ times is possible according to the depth constraints. □

Lemma 4.4.8. Let S_0 be a system with a cyclic variable, then there is a system S with a cyclic variable and with environment E such that S_0 is obtainable from S using no application of the rule `(Project)` and for all unsolved variables x in S, $(\mathtt{d}(x) \leq v) \in E$ where $v \geq \mathtt{fbound}(S)$.

Proof. Let φ be the derivation of S_0 and let S_1 be the last system in the derivation which is either an initial system or immediately after the application of a `(Project)`. If there is a cyclic variable in S_1, then we choose $S = S_1$ and have $(\mathtt{d}(x) \leq 2 \cdot \mathtt{fbound}(S)) \in E$ for all unsolved variables x in S_1 and we are done. Otherwise, since S_1 is acyclic, let $S = S_0$ and we can use Lemma 4.4.2 in order to obtain S such that $(\mathtt{d}(x) \leq v) \in E$ for all unsolved variables x in S where $v \geq \mathtt{fbound}(S)$. □

Lemma 4.4.9. Given a system $(S, \hat{b})$ with environment E and assume it contains a cyclic variable, then we can obtain either:

- a pure (unique) standard cycle using `BUA` while applying only the rules `(Delete)`, `(Decomp)`, `(Bind)` and `(Imitate`$_{pb}$`)` or
- for every pre-unifier σ of S, we can obtain a system $(S', \hat{b}')$ such that $\mathtt{b\text{-}measure}(S', \hat{b}') < \mathtt{b\text{-}measure}(S, \hat{b})$ and $\sigma \circ \theta$ is a pre-unifier of S' for some substitution θ.

Proof. We first use Lemma 4.4.8 in order to obtain a system with a cyclic variable such that for all $x \in \mathtt{FV}(S)$ there is $v \geq \mathtt{fbound}(S)$ such that $(\mathtt{d}(x) \leq v) \in E$. We use now Lemma 4.4.5 in order to obtain a cycle without having the environment changed. We now consider two cases:

- if the cycle is pure, then we obtain a non-unique standard cycle over the variables $x_1, .., x_k$ such that $(\mathtt{d}(x_i) \leq v) \in E'$ for $0 < i \leq n$ and $v > \mathtt{md}(S)$ using Lemma 4.4.6. The last step is to obtain a standard cycle using Lemma 4.4.7 and here we note that the size of the minimal position of x_1 in t_k must be smaller than $\mathtt{md}(S)$. This is because the rigid positions of variables cannot become deeper by applying the `(Imitate`$_{pb}$`)` rule.

- if the cycle is impure, then we consider the constraint $\lambda\overline{z_{m_j}}.x_j(\overline{t^j_{n_j}}) \doteq \lambda\overline{z_{m_j}}.t_j$ where t_j is impure for some $0 < j \leq k$. In order to preserve completeness, we must consider both applications of `(Project)` and `(Imitate`$_{pb}$`)`. Since an application of `(Project)` will decrease the bounding measure, we assume we apply `(Imitate`$_{pb}$`)` only, but after at most $\mathtt{d}(t_j)$ applications, where $\mathtt{d}(t_j) \leq \mathtt{md}(S) \leq \mathtt{fbound}(S)$, we will get a solved constraint $x_j \doteq t'_j$ where t'_j contains unsolved variables $x'_1, .., x'_l$ and since t_j was impure, $\hat{b}(x'_i) < \hat{b}(x_j)$ for $0 < i \leq l$ and therefore $\mathtt{b\text{-}measure}(S', \hat{b}') < \mathtt{b\text{-}measure}(S, \hat{b})$.

□

In the following lemma we show that for any pre-unifier σ of a problem containing a pure standard cycle, we can derive a problem with a reduced bounded measure which is pre-unifiable by σ.

Lemma 4.4.10. Given a system $(S, \hat{b})$ with a pure standard cycle, then for any ground unifier σ of S, there is a derivation $(S', \hat{b}')$ of $(S, \hat{b})$ using `BUA` such that S' is unifiable by σ and $\mathtt{b\text{-}measure}(S', \hat{b}') < \mathtt{b\text{-}measure}(S, \hat{b})$.

Proof. We first use Lemma 4.2.20 in order to obtain that there is a variable x such that $\sigma(x) \in_s \texttt{insts}(x, C', n)$ for C' the standard cycles' context and n its size. Assume there is a term $t \in \texttt{insts}(x, C', n)$ and a substitution θ such that $\sigma(x) = t\theta$. We now use Lemma 4.3.4 in order to obtain a system S' such that $\sigma_{S'}(x) = t$ and therefore $\sigma_{S'} \leq \sigma$. Since the definition of `insts` is based on `reg` which strictly reduces the `b-measure`, we get that $\texttt{b-measure}(S', \hat{b}') < \texttt{b-measure}(S, \hat{b})$. □

Theorem 4.4.11 (Completeness)**.** If a bounded unification system S is pre-unifiable by θ, then there exists a pre-solved system S', which is obtainable from S using `BUA` such that $\sigma_{S'}|_{\texttt{FV}(S)} \leq \theta$.

Proof. We will prove by induction over the bounding measure `b-measure`, that each pre-solved form obtainable using $\texttt{PUA}_B$ can be also obtained by `BUA`. The induction hypothesis is therefore: for a given system S having bounding measure m, if it is possible to obtain a pre-unifier of S using $\texttt{PUA}_B$, then it is possible to obtain the same pre-unifier using `BUA`. Induction base ($m = \emptyset$): we can replace all variables by first-order variables and clearly cyclic systems are not unifiable. Therefore, we can simulate any run of the complete $\texttt{PUA}_B$ with `BUA`. Induction step: Once we apply `(Project)` in $\texttt{PUA}_B$, we can use the induction hypothesis so we assume we need to simulate, using `BUA`, the remaining rules only. We notice that all rules except `(Imitate)` are the same. `(Imitate)` differs from $\texttt{(Imitate}_{pb}\texttt{)}$ in `BUA` with regard to cyclic variables only. We consider the following two cases. If `(Imitate)` is applied on a variable which is not repeated and is not cyclic, then we can use Lemma 4.4.2 and obtain the same system using the rule $\texttt{(Imitate}_{pb}\texttt{)}$ of `BUA`. Now, assume we apply `(Imitate)` in $\texttt{PUA}_B$ on a variable that is either cyclic or repeated. Using Lemma 4.4.4 we know that if the variable is repeated, then there is a cyclic variable in the system. We now show that without losing any pre-unifier, we can reduce the bounding measure and therefore apply the induction hypothesis. Since the only two non-deterministic rules to apply are $\texttt{(Imitate}_{pb}\texttt{)}$ and `(Project)` and an application of `(Project)` will allow us to use the induction hypothesis, we can use Lemma 4.4.9, without losing any pre-unifier, in order to obtain either a system with a smaller bounding measure or system with a pure standard cycle. We use Lemma 4.4.10 and the fact that a pre-unifier can be extended easily into a ground unifier in order to derive, in the second case, a system using `BUA` which is unifiable by θ and which has a smaller bounding measure. Either way, we can use the induction hypothesis. □

4.5 Termination and Minimal Unifiers

In this section we will show that in practice, the number of recursive calls to $\texttt{(Imitate}_{*}\texttt{)}$ can be bounded due to the following result [72]:

Definition 4.5.1 (Minimal unifiers)**.** Given a system S, a unifier σ of S is called minimal if there is no other unifier σ' of S with $\Sigma_{x \in \texttt{FV}(S)} \texttt{size}(\sigma'(x)) < \Sigma_{x \in \texttt{FV}(S)} \texttt{size}(\sigma(x))$.

Definition 4.5.2 (Exponent of periodicity)**.** A ground unifier σ has an exponent of periodicity n iff n is the maximal number such that there is some variable x and ground contexts A and B as well as a term t such that $\sigma(x) = \lambda\overline{z_m}.AB^n t$ for $m \geq 0$.

Next we define a bound on the exponents. The bound is based on functions defined in [74], please refer to the definitions and proofs there for correctness and intuition for these functions and values.

Definition 4.5.3 (Exponent function (from Lemma 4.7 in [74]))**.** Given a system S, then the exponent function $\texttt{eop}(S) = ((5 \cdot \texttt{fsize}(S) - 6) \cdot (e^{1/e})^{2 \cdot \texttt{fsize}(S) - 3}) - 2$ where:

- $\texttt{fsize}(S) = 2_{\texttt{ord}(S)}(\texttt{repn}(S) \cdot \texttt{sbeqnf}(S))$.
- $\texttt{repn}(S) = 6 \cdot \texttt{maxb}(S) \cdot \texttt{maxar}(S) + 22 \cdot \texttt{maxb}(S) + 2$.
- $\texttt{maxb}(S)$ is the maximal value v in $E[\hat{b}(x) = v]$ for environment E and a variable x in S.
- $\texttt{maxar}(S)$ is the maximal arity of a term in S.
- $\texttt{sbeqnf}(S) = \texttt{seqnf}(S)^{2_{\texttt{ord}(S)}(\texttt{seqnf}(S))}$.
- $\texttt{seqnf}(S) = 3 \cdot \texttt{size}(S) \cdot \texttt{maxts}(S)$.
- $\texttt{maxts}(S)$ is the maximal size of a type of a term in S.

Lemma 4.5.4 ([74])**.** For every unifiable system S and for every minimal unifier σ of S, its exponent of periodicity is less than $\texttt{eop}(S)$.

The Restricted BUA

The exponent computed in the lemma above allows us to replace the Kleene star in the regular terms with a concrete value.

Definition 4.5.5 (Restricted `derail`)**.** Given a system S, the restricted `derail` function for S ($\texttt{derail}_S$) is defined as `derail` but instead of introducing the Kleene star, the function introduce the number $\texttt{eop}(S)$. The produced terms are called restricted regular terms or just regular terms.

Definition 4.5.6 (Restricted instantiations and descriptions)**.** Similarly to Def 4.2.16, we define a restricted instantiation of a restricted regular term t as the term obtained by replacing the n occurrences of the exponent e by values $k_1, .., k_n$ such that $k_i \leq e$ for $0 < i \leq n$. In addition we define the set of all instantiations of a restricted regular term t as the finite set containing all possible restricted instantiations. In the remaining of this section `insts` will refer to its restricted version.

Example 4.5.7. *The term* $\lambda z.(f([.], w))^4 f(x_1(z), x_2(z))$ *is a restricted instantiation of the restricted regular term* $\lambda z.(f([.], w))^e f(x_1(z), x_2(z))$ *for* $e = 8$.

Definition 4.5.8 (Environments and constraints)**.** The notions of environments and of binding and depth constraints are the same as for BUA. The only difference is that we identify each binding constraint over a restricted regular term t with a natural number such that this number is the maximal size of an `mpath` of all maximal contexts of terms contained in $\texttt{insts}(t)$ $+2$. The intuition behind this value is to describe the maximal depth of a bound variable in the regular term. Since the language is (now) finite, it is possible to compute this value. We call this number the value of the constraint.

Example 4.5.9. *Assume we have a constraint $x \lhd t$ where t is the restricted regular context from the previous example, then the value of this constraint is $8 + 2 = 10$.*

Definition 4.5.10 (Restricted `BUA` (`RBUA`))**.** Let $(S_0, \hat{b}_0)$ be the initial system, then the restricted `BUA` (`RBUA`) consists of the rules `(Delete)`, `(Decomp)`, `(Bind)`, `(Imitate`$_{pb}$`)`, `(Project)` and `(Imitate`$_0$`)` from Fig. 4.2 together with the three rules in Fig. 4.4.

Example 4.5.11. *By replacing all the Kleene stars in Ex. 4.3.5 with the exponent of periodicity of the problem we can obtain exactly the same derivation using* `RBUA`*.*

Lemma 4.5.12 (Soundness)**.** If S' is obtained from a unification system S using `RBUA` then `PreUnifiers`$(S') \subseteq$ `PreUnifiers`(S).

Proof. Following from Thm. 4.4.1. □

Lemma 4.5.13. If θ is a minimal unifier of a unification system S, then there exists a pre-solved system S', which is obtainable from S using `RBUA` such that $\sigma_{S'}|_{\mathtt{FV}(S)} \leq \theta$.

Proof. From the completeness of `BUA` we know that we can obtain such a pre-unifier for each unifier of S. By using Lemma 4.5.4 we can show that we do not need to seek pre-unifiers with term depth bigger than the exponent of periodicity, which is exactly the bound we use in the algorithm. □

Definition 4.5.14 (Regular measure)**.** Let E be an environment and let $d_1, .., d_n$ be the values of the binding constraints in E for all unsolved variables in S, then the regular measure of E is the sum $\Sigma_{0<i\leq n} d_i$.

Definition 4.5.15 (Depth measure)**.** Let E be an environment and let $m_1, .., m_n$ be all the numbers occurring in depth constraints in E for all unsolved variables in S, then the depth measure of E is the sum $\Sigma_{0<i\leq n} m_i$.

Theorem 4.5.16. Given a system $(S, \hat{b})$, `RBUA` terminates on $(S, \hat{b})$.

Proof. The algorithm is finitely branching. We will show termination of a specific run by taking the lexicographic ordering of the following measure $\mu =< m_1, m_2, m_3, m_4, m_5, m_6 >$ where

- m_1 is the bounding measure `b-measure`$(S, \hat{b})$,
- m_2 is is the multiset $\{\hat{b}(x) - \mathtt{ar}(x) | x \in V\}$ where V contains all variables which do not occur also in a binding constraint in E.
- m_3 is the regular measure,
- m_4 is the depth measure,

$$\frac{E[r(x) = \lambda\overline{z_k}.C^l(C_r)] \vdash S}{E[r(x) = \lambda\overline{z_k}.C_r] \vdash S}\ \texttt{(Skip)}$$

$$\frac{E[r(x) = \epsilon] \vdash S \qquad c \in \texttt{scy}(S, E), x \in c, t \in \texttt{derail}_{S_0}(x, \texttt{size}(c), \texttt{ccon}(c))}{E[r(x) = t] \vdash S}\ \texttt{(Rec)}$$

$$\frac{E[r(x) = \lambda\overline{z_n}.t^l(tr)] \vdash S \qquad \lambda\overline{z_k}.x^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{v_p}) \in S, t = f(t_1, .., t'([.])_{@k}, .., t_p), u = \texttt{PB}^b(f, \alpha), \sigma = [u/x]}{E[r(x_1) = t_1, .., r(x_k) = t'(t^{l-1}(tr)), .., r(x_p) = t_p]\sigma \vdash S\sigma \cup \{x \doteq u\}}\ \texttt{(Imitate}_*\texttt{)}^1$$

1. $x_1, .., x_p$ are all fresh variables introduced by $\texttt{PB}^b$.

Figure 4.4: RBUA - Restricted bounded unification rules

- m_5 is the number of unsolved variables x with $\hat{b}(x) = 0$ and
- m_6 is the number of symbols other than $\doteq$ in the problem.

We prove that the measure μ is decreased after any application of `RBUA`.

- An application of `(Delete)` or `(Decomp)` decreases m_6 and does not increase any other measure.
- An application of `(Bind)` either decreases m_1 and m_2 if $\hat{b}(x) > 0$ or decreases m_5 if $\hat{b}(x) = 0$. It does not increase m_1, m_2 or m_4. It also does not increase m_3 since the size of `mpath` of maximal contexts is not affected by `(Bind)`.
- An application of `(Imitate`$_{pb}$`)` decreases m_4 and does not increase m_1 or m_2 (although it might decrease them). It does not increase m_3 following the previous argument.
- An application of `(Skip)`, `(Imitate`$_0$`)` and `(Imitate`$_*$`)` decreases m_3 as we decrease the size of the `mpath` of the maximal context in the new constraint by at least 1. It does not increase m_1 or m_2.
- An application of a `(Project)` decreases m_1.
- An application of a `(Rec)` decreases m_2 as it introduces a binding constraint for a variable and is applicable only if one did not exists. It does not increase m_1.

Therefore the measure μ decreases after each application of `RBUA`. □

Theorem 4.5.17. The unifiability question of bounded unification problems is decidable.

Proof. Following lemmas 4.5.12 and 4.5.13 and Thm. 4.5.16. □

Corollary 4.5.18. The monadic second-order unification problem [28] is decidable.

Proof. Since the problem is second-order, bounded variables must be of basic type only and as the signature contains only monadic function symbols and constants, any ground unifier of the problem will map the variables of the problem to terms with at most 1 bounded variable occurrence. The maximal number of λ-binders can be deduced from the original variables in the problem and hence a fixed bound can be given for such a problem and we can use Thm. 4.5.17 in order to decide its unifiability. □

CHAPTER 5

Higher-order Resolution

The constrained resolution calculus defined in Sec. 2.3 has one main drawback. In order to avoid the need to choose a unifier out of infinitely-many possibilities, the calculus postpones calling unification. But, since unifiability is undecidable in higher-order logic, this means the search space is greatly increased.

Fully-automated resolution normally proceeds by creating new resolvents until the empty clause is found. The search space already in first-order resolution tends to be very big, despite the many refinements which were found for first-order automated reasoning. Any further increase due to the higher-order of the terms may make it impractical.

In this chapter we will survey and compare several methods and strategies for using unification in the constrained resolution calculus. We will first introduce to the reader several test clause sets which will be used in order to test each of the variants of the constraint resolution calculus presented later in this chapter. The first calculus to be compared and tested will be the original one, which was presented in Sec. 2.3. Most higher-order theorem provers, such as Leo II [9], are placing bounds on the search for unifiers. The bound, though, is pretty trivial and can be easily shown to be insufficient. This bound will be integrated into the second variant while the third variant will use a slightly improved one. The fourth variant will employ a unification algorithm which is based on a more "semantical" bound [74]. The unifiability question with regard to these bounds was shown to be decidable, which in general, makes the algorithm usable in theorem proving. While Huet's unification algorithm returns all pre-unifiers within the bounds, this algorithm returns only a finite subset of them, which prevents the resolution calculus employing it from being complete with regard to the full set of bounded unifiers. In the last section we will present a calculus based on the unification algorithm from Chap. 4. It will use the same bounds as the algorithm from Sec. 5.2 but unlike this algorithm, it will enumerate the infinitely-many pre-unifiers in a way which will make the calculus usable in practice.

5.1 The Test Sets

In this section we will present several sets of clauses, which will serve as test sets for the various calculi presented later in this chapter. The test sets are chosen in order to emphasize particular strengths and weaknesses in the compared calculi and for that end, relatively trivial examples are enough. Although it is possible to find examples of arbitrary complexities as well, such examples will be ill suited for the comparison of the calculi. Nevertheless, the examples given below can be easily made schematic and therefore each example can be extended into an infinite set of examples, all ending with the same result.

In order for our comparison to be fair, we will use the following heuristics when choosing which clauses and literals to process:

- choose shortest clauses according to the number of characters.
- choose shallowest literals, when the depth of a literal is the maximal depths of a term in it.
- the search for a refutation applies the unification rules depth-first and the remaining rules breadth-first.

Please note that applying all rules depth-first might give an increase in performance but can easily lead to non-termination. By applying all rules breadth-first, most calculi presented in this chapter can find a refutation for the refutable clause sets but result in a decreased performance.

Remark 5.1.1. A full comparison between depth and breadth-first search for unifiers and refutations is beyond the scope of this thesis. In the basic case, depth-first is always using the last clause that was generated and breadth-first uses first all clauses that were generated before. There are many combinations and improvements of these strategies which will not be discussed further. When using the depth-first strategy, even a search for a refutation of a first-order clause set might not terminate, even if the set is refutable. An example is the following set, using which we can always resolve the last generated clause with the first clause in the set and continue so indefinitely.

$$\{[P(x)]^{\mathrm{F}} \vee [P(f(x))]^{\mathrm{T}}, [P(a)]^{\mathrm{T}}, [P(f(f(a)))]^{\mathrm{F}}\} \tag{5.1}$$

On the other hand, since the number of clauses in the problem grows very fast, applying breadth-first search can be extremely inefficient. If we also postpone unification, we will add to the list of clauses many clauses which have non-unifiable constraints and the number of possible clauses will soon grow so large that it will pose a serious space problem.

Each of the test sets we give next is refutable and we will show it by giving a refutation, using the constrained resolution calculus from Sec. 2.3, which will *not* follow the heuristics above.

Since the pre-unification algorithm, which is employed by this calculus, always find a unifier if one exists, we will normally skip the unification tests and give the unifier directly. This will be done by using a slightly different version of Huet's constrained resolution calculus, which more resembles Andrews' resolution calculus [3].

Definition 5.1.2 (The constrained resolution calculus)**.** The resolution calculus used in this section is based on the simplifications, resolution and factorization rules from figures 2.3 and 2.4, the `(Delete)`, `(Decomp)` and `(Bind)` rules from Figure 2.1 and the `(Sub)` rule from Fig. 5.1.

$$\frac{C \qquad \sigma = [t/x]}{C\sigma}\;\texttt{(Sub)}$$

Figure 5.1: Substitution rule

Theorem 5.1.3. The resolution calculus from Def. 5.1.2 is sound and complete with regard to the constrained resolution calculus from Sec. 2.3.

Proof. Clear as the `sub` rule can return all substitutions obtained by the omitted rules and is still sound. □

Clause Set 1

The main weakness of the constrained resolution calculus is that it may build up a set of unification constraints which is not unifiable. The worst case is when the search for unifiers of this set will not terminate. In this case the only way to force the algorithm to terminate if the set is refutable is by searching for a unifier using a breadth-first search.

Remark 5.1.4. With regard to the application of monadic function symbols, please note that they associate to the right (see Remark 2.1.53).

Definition 5.1.5 (Clause set 1)**.**

$$\{[P(Xfa, fXa, gXa)]^{\mathrm{T}}, [P(y,y,gz)]^{\mathrm{F}}, [P(y,z,y)]^{\mathrm{F}}\} \tag{5.2}$$

Fig. 5.2 gives a refutation of this clause set.

$$\cfrac{\cfrac{\cfrac{[P(Xfa, fXa, gXa)]^{\mathrm{T}} \qquad [P(y,y,gz)]^{\mathrm{F}}}{[P(Xfa, fXa, gXa) \doteq P(y,y,gz)]^{\mathrm{F}}}\;\texttt{(Resolve)}}{[P(fa,fa,ga) \doteq P(fa,fa,ga)]^{\mathrm{F}}}\;\texttt{(Sub)}\;([\lambda z.z/X, fa/y, a/z])}{}\;\texttt{(Delete)}$$

Figure 5.2: A refutation of clause set 1

Clause Set 2

In order to force the pre-unification algorithm to terminate, one can restrict the depth of the terms obtained. Although the unification algorithm will always terminate, the completeness of the calculus is greatly impaired.

Definition 5.1.6 (Clause set 2)**.**

$$\{[Xa]^{\mathrm{F}}, [Pf^n a]^{\mathrm{T}}\} \tag{5.3}$$

where $n \geq 0$.

Fig. 5.3 gives a refutation of this clause set.

$$\begin{array}{c}
[Xa]^{\mathrm{F}} \qquad [Pf^n a]^{\mathrm{T}} \\ \hline
[Xa \doteq Pf^n a]^{\mathrm{F}} \\ \hline
[Pf^n a \doteq Pf^n a]^{\mathrm{F}} \\ \hline
\end{array}
\begin{array}{l}
\texttt{(Resolve)} \\
\texttt{(Sub)}\ ([\lambda z.Pf^n z/X]) \\
\texttt{(Delete)}
\end{array}$$

Figure 5.3: A refutation of clause set 2

Clause Set 3

A bound which will allow for a more "Semantical incompleteness", is the first-order bound which was first introduced in Chap. 3 (see Def. 3.2.2). Although using such a bound will be enough for the above example, it will fall short for cyclic ones, such as the one adapted from Ex. 1.3.1.

Definition 5.1.7 (Clause set 3)**.**

$$\{[Q(X_1abX_1bX_2bX_3bX_4c, aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac)]^{\mathrm{F}}, [Q(y,y)]^{\mathrm{T}}\} \tag{5.4}$$

Fig. 5.4 gives a refutation of this clause set.

$$\begin{array}{c}
[Q(X_1abX_1bX_2bX_3bX_4c, aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac)]^{\mathrm{F}} \qquad [Q(y,y)]^{\mathrm{T}} \\ \hline
[Q(X_1abX_1bX_2bX_3bX_4c, aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac) \doteq Q(y,y)]^{\mathrm{F}} \\ \hline
[X_1abX_1bX_2bX_3bX_4c \doteq y]^{\mathrm{F}}, [aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac \doteq y)]^{\mathrm{F}} \\ \hline
[X_1abX_1bX_2bX_3bX_4c \doteq aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac)]^{\mathrm{F}} \\ \hline
[a^{81}aba^{81}ba^{27}ba^{9}ba^{3}c \doteq aa^{81}ba^{27}a^{27}a^{27}ba^{9}a^{9}a^{9}ba^{3}a^{3}a^{3}baaac]^{\mathrm{F}} \\ \hline
\end{array}
\begin{array}{l}
\texttt{(Resolve)} \\
\texttt{(Decomp)} \\
\texttt{(Bind)} \\
\texttt{(Sub)}\ \text{(see Eq. 5.5)} \\
\texttt{(Delete)}
\end{array}$$

Figure 5.4: A refutation of clause set 3

where the substitution is

$$[\lambda z.a^{81}z/X_1, \lambda z.a^{27}z/X_2, \lambda z.a^{9}z/X_3, \lambda z.a^{3}z/X_4] \tag{5.5}$$

Clause Set 4

Makanin [59] was the first to give a way to deal with cyclic constraints, or more generally, with infinitary unification problems. His approach was to search for a finite subset of all unifiers and to prove that this subset is never empty if the problem is unifiable. This approach was developed further in order to deal with monadic second-order unification problems [28], context unification problems [72], linear unification problems [55] and bounded unification problems [74].

A major drawback of this approach, when considering resolution, is the incomplete set of unifiers computed. The following clause set is built in such a way that this drawback comes into play.

Definition 5.1.8 (Clause set 4)**.** Let $n > 0$ then:

- $\Gamma(n) = [Q(Yac, aYc)]^{\mathrm{T}} \vee [P_1(Yc)]^{\mathrm{T}} \vee .. \vee [P_n(Yc)]^{\mathrm{T}}$
- $\Lambda = [Q(z, z)]^{\mathrm{F}}$
- $S_0 = \{\Gamma(n), \Lambda\}$
- $\Delta(m) = [P_m(\overbrace{a..a}^{e_m} y_m)]^{\mathrm{F}}$ for all $0 < m \leq n$
- $S_m = \{\Gamma(n), \Lambda, \Delta(m)\}$ for all $0 < m \leq n$
- $e_m = \mathtt{eop}(S_{m-1}) + 1$ for $0 < m \leq n$

and clause set 4 is :

$$\{\Gamma(n), \Lambda, \Delta(1), .., \Delta(n)\} \tag{5.6}$$

The next lemma is easily proved.

Lemma 5.1.9. Let σ be the composed substitution obtained from a refutation of clause set 4, then $\sigma(Y) = \lambda z.a^k z$ for $k \geq e_n$.

Proof. After applying `(Resolve)`s, `(Decomp)`s and `(Bind)`s, we get, among others clauses, also the following two constraints:

- the unification constraint $[Yac \doteq aYc]^{\mathrm{F}}$ which implies $\sigma(Y) = \lambda z.a^k z$ for $k \geq 0$
- the constraint $[Yc \doteq \overbrace{a..a}^{e_n} y_n]^{\mathrm{F}}$ which implies $\sigma(Y) = \lambda z.a^{e_n} vz$ and $\sigma(y_n) = vc$ for some context v.

□

Fig. 5.5 gives a refutation of this clause set for $n = 2$.
where the substitution is

$$[\lambda z. \overbrace{a..a}^{e_2} z/Y, \overbrace{a..a}^{e_2 - e_1} c/y_1, c/y_2] \tag{5.7}$$

5.2 Variants of the Constrained Resolution Calculus

In this section we present some variations of the constrained resolution calculus, which are either used in practice in resolution theorem provers or can be created by adding rules from known unification algorithms.

$$\dfrac{[Q(Yac, aYc)]^{\mathrm{T}} \vee [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \qquad [Q(z,z)]^{\mathrm{F}}}{[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Q(Yac, aYc) \doteq Q(z,z)]^{\mathrm{F}}}\;\texttt{(Resolve)}$$

$$\dfrac{[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Q(Yac, aYc) \doteq Q(z,z)]^{\mathrm{F}}}{[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq z]^{\mathrm{F}} \vee [aYc \doteq z]^{\mathrm{F}}}\;\texttt{(Decomp)}$$

$$\dfrac{[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq z]^{\mathrm{F}} \vee [aYc \doteq z]^{\mathrm{F}}}{[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}\;\texttt{(Bind)}$$

$$\dfrac{[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}} \qquad [P_1(\overbrace{a..a}^{e_1}\, y_1)]^{\mathrm{F}}}{[P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}} \vee [P_1(Yc) \doteq P_1(\overbrace{a..a}^{e_1}\, y_1)]^{\mathrm{F}}}\;\texttt{(Resolve)}$$

$$\dfrac{[P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}} \vee [P_1(Yc) \doteq P_1(\overbrace{a..a}^{e_1}\, y_1)]^{\mathrm{F}}}{[P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_1}\, y_1]^{\mathrm{F}}}\;\texttt{(Decomp)}$$

$$\dfrac{[P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_1}\, y_1]^{\mathrm{F}} \qquad [P_2(\overbrace{a..a}^{e_2}\, y_2)]^{\mathrm{F}}}{[Yac \doteq aYc]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_1}\, y_1]^{\mathrm{F}} \vee [P_2(Yc) \doteq P_2(\overbrace{a..a}^{e_2}\, y_2)]^{\mathrm{F}}}\;\texttt{(Resolve)}$$

$$\dfrac{[Yac \doteq aYc]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_1}\, y_1]^{\mathrm{F}} \vee [P_2(Yc) \doteq P_2(\overbrace{a..a}^{e_2}\, y_2)]^{\mathrm{F}}}{[Yac \doteq aYc]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_1}\, y_1]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_2}\, y_2]^{\mathrm{F}}}\;\texttt{(Decomp)}$$

$$\dfrac{[Yac \doteq aYc]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_1}\, y_1]^{\mathrm{F}} \vee [Yc \doteq \overbrace{a..a}^{e_2}\, y_2]^{\mathrm{F}}}{[\overbrace{a..a}^{e_2}\, ac \doteq a\,\overbrace{a..a}^{e_2}\, c]^{\mathrm{F}} \vee [\overbrace{a..a}^{e_2}\, c \doteq \overbrace{a..a}^{e_1}\;\overbrace{a..a}^{e_2-e_1}\; c]^{\mathrm{F}} \vee [\overbrace{a..a}^{e_2}\, c \doteq \overbrace{a..a}^{e_2}\, c]^{\mathrm{F}}}\;\texttt{(Sub)}\text{ (see Eq. 5.7)}$$

$$\dfrac{[\overbrace{a..a}^{e_2}\, ac \doteq a\,\overbrace{a..a}^{e_2}\, c]^{\mathrm{F}} \vee [\overbrace{a..a}^{e_2}\, c \doteq \overbrace{a..a}^{e_1}\;\overbrace{a..a}^{e_2-e_1}\; c]^{\mathrm{F}} \vee [\overbrace{a..a}^{e_2}\, c \doteq \overbrace{a..a}^{e_2}\, c]^{\mathrm{F}}}{}\;\texttt{(Delete)}\times 3$$

Figure 5.5: A refutation of clause set 4 for $n = 2$

The Original Calculus

The first calculus we will compare with regard to our test sets will be Huet's original constrained resolution calculus as was presented in Sec. 2.3. Since the unification algorithm is non-terminating, the strategy which is normally chosen when using the calculus is to call unification only when necessary. This is obtained by postponing the call to unification until a clause is obtained, which is made of unification constraints only. Huet has mentioned that some unification calls, such as first-order unification, can be performed eagerly. Nevertheless, the algorithm may fail to find a refutation, even if one exists.

In the remaining of this chapter, we will refer to the unmodified constrained resolution calculus as `CRC`.

We will now apply the algorithm to our test cases:

Lemma 5.2.1. `CRC` fails to terminate on clause set 1 and never finds a refutation.

Proof. By choosing the shortest clauses and the shallowest literals first, we obtain the derivation in Fig. 5.6. Clearly, the clause set is not unifiable but the unification algorithm will produce infinitely many mappings for x. Note that applying `(Project)` instead of `(Imitate)` in the derivation in Fig. 5.6 will also make the clause set non-unifiable. The fact that the derivation does not terminate can be seen from the similarity between lines 4 and 6 in the derivation. □

$$\begin{array}{cl}
[P(Xfa, fXa, gXa)]^{\mathrm{T}} \qquad [P(y,z,y)]^{\mathrm{F}} & \\
\hline
[P(Xfa, fXa, gXa) \doteq P(y,z,y)]^{\mathrm{F}} & \texttt{(Resolve)} \\
\hline
[Xfa \doteq y]^{\mathrm{F}}, [fXa \doteq z]^{\mathrm{F}}, [gXa \doteq y]^{\mathrm{F}} & \texttt{(Decomp)} \\
\hline
[Xfa \doteq gXa]^{\mathrm{F}}, [fXa \doteq z]^{\mathrm{F}} & \texttt{(Bind)} \\
\hline
[gX_1fa \doteq ggX_1a]^{\mathrm{F}}, [fgX_1a \doteq z]^{\mathrm{F}} & \texttt{(Imitate)} \\
\hline
[X_1fa \doteq gX_1a]^{\mathrm{F}}, [fgX_1a \doteq z]^{\mathrm{F}} & \texttt{(Decomp)}
\end{array}$$

Figure 5.6: A non-terminating derivation

Lemma 5.2.2. `CRC` finds a refutation for each of the clause sets 2 and 3.

Proof. Since both have only two clauses, each with one literal only, we can obtain derivations of the clauses $[Xa \doteq Pf^na]^{\mathrm{F}}$ and $[X_1abX_1bX_2bX_3bX_4c \doteq aX_1bX_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac)]^{\mathrm{F}}$ respectively. Since both constraints are unifiable (see figures 5.3 and 5.4) and `PUA` always computes a unifier if one exists, we can obtain a refutation for each clause set. □

Lemma 5.2.3. `CRC` finds a refutation for clause set 4 for all $n > 0$.

Proof. Any choice of clauses and literals will result in unifiable constraint set and therefore we can obtain a refutation. □

Depth-bounded Resolution

In this section we will describe two calculi which place a bound on the depth of the obtained terms.

Fixed depth

A very popular approach for forcing PUA to terminates is to put a bound on the depth of the terms generated by the algorithm. The very popular higher-order theorem prover Leo II [9] has a fixed number (usually between 3 and 8) as the bound [10], [11]. This bound was determined to be enough for the refutation of a considerable amount of higher-order problems from the TPTP library [78].

Definition 5.2.4 (Depth-bounded resolution calculus ($\texttt{CRC}_d$))**.** let $b > 0$ be a pre-defined number, the depth-bounded resolution calculus has the same rules as CRC, but all the rules have as a prerequisite that the clauses in the upper parts of the rules do not contain a solved constraint of the form $x \doteq t$ where $\texttt{d}(t) \geq b$ and x is a higher-order variable.

Example 5.2.5. *Given $b = 3$, then no rule of $\texttt{CRC}_d$ is applicable to the clause $C \vee [Xa \doteq fffy]$.*

Lemma 5.2.6. $\texttt{CRC}_d$ obtains a refutation for clause set 1.

Proof. Clear, as for any $b > 1$ we can obtain the same unifier as in Fig. 5.2. □

Lemma 5.2.7. For any given b, $\texttt{CRC}_d$ fails to find a refutation for clause set 2 for $n > b - 1$.

Proof. Clear, as we need to apply the (Imitate) rule $n + 1$ times, after which, we will have the solved constraint $X \doteq \lambda z.Pf^n z$ in the obtained clause. □

Lemma 5.2.8. $\texttt{CRC}_d$ fails to find a refutation for clause set 3 for $b < 81$.

Proof. We first prove that the substitution in Fig. 5.4 is the only substitution that unifies the constraint. It is not difficult to see that X_1 must be mapped to a term of the form a^n. After applying this mapping and (Decomp)s, we get the constraint $[a^n bX_2bX_3bX_4c \doteq X_2X_2X_2bX_3X_3X_3bX_4X_4X_4baaac)]^{\mathrm{F}}$. Now we can only get a unifier if X_2 is mapped to a term of the form $a^{n/3}$ and after applying this mapping and (Decomp)s, we get the constraint $[a^{n/3}bX_3bX_4c \doteq X_3X_3X_3bX_4X_4X_4baaac)]^{\mathrm{F}}$. Similarly we get that X_3 must be mapped to $a^{n/9}$ and X_4 must be mapped to both $a^{n/27}$ and to a^3, which concludes the proof. Now, since X_1 is mapped to a term of depth 81, b must be at least 81 in order for the rules to be applicable. □

Lemma 5.2.9. $\texttt{CRC}_d$ fails to find a refutation for clause set 4 for $b < e_n$.

Proof. Following Lemma 5.1.9. □

Problem-based depth

A more flexible bound on the depth of the produced terms is one that is based on the input problem. In Def. 3.2.2 we have defined a bound on the depth of terms in case of acyclic unification constraints. We will use this bound for defining the next variant of $\mathtt{CRC}$.

Definition 5.2.10 (FOL-bounded resolution calculus ($\mathtt{CRC}_{fol}$))**.** The FOL-bounded resolution calculus is a $\mathtt{CRC}_d$ with $b = \mathtt{fbound}(S)$ where S is the input clause set.

Lemma 5.2.11. $\mathtt{CRC}_{fol}$ obtains a refutation for clause set 1.

Proof. Clearly, $b = \mathtt{fbound}(S) > \mathtt{md}(S) = 4$ and we can obtain the same unifier as in Fig. 5.2. □

Lemma 5.2.12. $\mathtt{CRC}_{fol}$ obtains a refutation for clause set 2.

Proof. Clearly, $b = \mathtt{fbound}(S) > \mathtt{md}(S) = n + 1$ and we can obtain the same unifier as in Fig. 5.3. □

Lemma 5.2.13. $\mathtt{CRC}_{fol}$ fails to find a refutation for clause set 3.

Proof. We compute $b = \mathtt{fbound}(S) = 10$ and use Lemma 5.2.8. □

Lemma 5.2.14. $\mathtt{CRC}_{fol}$ obtains a refutation for clause set 4 for $n > 1$.

Proof. The $\mathtt{fbound}$ of the set of constraints obtained from resolving S_m is $e_n \cdot 6$ and therefore for all $n > 1$ we have $e_n < e_n \cdot 6$. □

Projection-Bounded Resolution

The first-order bound used in the previous calculus does not fail only on cyclic sets of unification problems. Clearly the calculus will fail to find the unifier even of the simple problem:

$$\{[Xa \doteq g(a,b)]^{\mathrm{F}}, [Xb \doteq g(b,b)]^{\mathrm{F}}, [g(Ya,b) \doteq Xf^9a]^{\mathrm{F}}\} \tag{5.8}$$

since X must be mapped, in any possible unifier, to $\lambda z.g(z,b)$ and this then leaves the constraint $[Ya \doteq f^9a]$ which means Y must be of depth at least 9 but the first-order bound is $3 \cdot 2 = 6$.

It is not known yet if we can pre-compute a bound on the depth of terms for acyclic problems such that it will not harm the completeness of the calculus, but it is clear that if we just recompute the first-order bound every time we apply a projection then the method will be complete for acyclic problems.

Although it will be complete, we do not have termination now since the number of required projections is not known. In order to give a terminating unification algorithm, we will just put an arbitrary bound on the number of allowed projections.

Definition 5.2.15 (Projection-bounded resolution calculus ($\mathtt{CRC}_p$))**.** Let $p > 0$ be a pre-defined number, the projection-bounded resolution calculus has the same rules as $\mathtt{CRC}_{fol}$, but every time `(Project)` is called, we recompute the first-order bound for the unsolved unification constraints and we allow at most p calls to this rule.

Lemma 5.2.16. $\texttt{CRC}_p$ obtains a refutation for clause set 1.

Proof. Clearly, $b = \texttt{fbound}(S) > \texttt{md}(S) = 4$ and we can obtain the same unifier as in Fig. 5.2. □

Lemma 5.2.17. $\texttt{CRC}_p$ obtains a refutation for clause set 2.

Proof. Clearly, $b = \texttt{fbound}(S) > \texttt{md}(S) = n + 1$ and we can obtain the same unifier as in Fig. 5.3. □

Although the calculus is much stronger than $\texttt{CRC}_{fol}$ it still fails on cyclic problems.

Lemma 5.2.18. $\texttt{CRC}_p$ fails to find a refutation for clause set 3.

Proof. We compute $b = \texttt{fbound}(S) = 10$ and use Lemma 5.2.8. □

Lemma 5.2.19. $\texttt{CRC}_p$ obtains a refutation for clause set 4 for $n > 1$.

Proof. The fbound of the set of constraints obtained from resolving S_m is $e_n \cdot 6$ and therefore for all $n > 1$ we have $e_n < e_n \cdot 6$. □

Minimal Binders Resolution

The most straight forward way to put a bound on the number of allowed projections is to put a bound on the number of allowed bound variables in the range of unifiers.

In many cases, this bound is "semantically" more interesting than a bound on the depth of the generated terms. If we consider all higher-order variables as representing sets, the number of bounded variables occurring in the terms mapped to these variables clearly relate to the definitions of these sets. For example if X represents a set, which denotes the union of two other sets, represented by Y and Z, then the number of occurrences of bound variables in $\sigma(X)$ cannot exceed that in $\sigma(Y)$ plus that in $\sigma(Z)$. If we want to denote that the set contains all elements bigger than 2014, we can denote it using two occurrences of bound variables and λ-binders by $\lambda z.(z > 2014)$. Note that the use of Church numbers [17] is also made possible using this bound.

Many unification problems have a natural bound on the number of allowed projections. In word unification [59] and in context unification [21] the terms mapped to higher-order variables can have exactly one bound variable while in monadic second-order unification [28] they can have at most one such variable. In both cases we can use the number of the original higher-order variables in the problem as a bound to the number of projections.

This still does not help us to put a bound on the depth of terms in cyclic problems. The algorithms which decide the problems above are using another trick - they are looking for minimal unifiers only.

An algorithm of this type for checking unifiability of bounded higher-order unification problems was given in [74]. In the remaining of this section we will compare several resolution calculi based on bounded unification algorithms.

Definition 5.2.20 (Minimal binders resolution calculus ($\texttt{CRC}_\lambda$))**.** The minimal binders resolution calculus has all the rules from figures 2.3, 2.4 and 2.5 and in addition has the unification rules presented in [74] and adapted to our clause notations.

The partial completeness of the unification algorithm presented in [74] is based on the following lemma, which can be deduced from [74]:

Lemma 5.2.21 ([74])**.** If a constraint set is unifiable by a bounded unifier, then the algorithm finds (at least one and) all bounded unifiers such that their exponent is at most E, where E is the exponent of periodicity (see Def. 4.5.2).

Lemma 5.2.22. $\texttt{CRC}_\lambda$ obtains a refutation for each of the clause sets 1, 2 and 3.

Proof. By running the unification algorithm from [74] we can obtain each of the unifiers. □

In order to evaluate how well $\texttt{CRC}_\lambda$ runs on clause set 4, we will use the following function.

Definition 5.2.23 (Evaluation function for unification algorithms (Ψ_{eval}))**.** For each substitution σ obtained by the unification algorithm during the search, the evaluation function adds $\Sigma_{x \in V} \mathtt{d}(\sigma(x))$ for V the set of all higher-order variables.

The intuition behind Ψ_{eval} is to compute the number of imitations applied to each higher-order variable.

Lemma 5.2.24. $\texttt{CRC}_\lambda$ obtains a refutation for clause set 4 with
$\Psi_{eval} \geq \Sigma_{i=1..n}(e_{i-1} + \Sigma_{j=e_{i-1}..e_i} j)$.

Proof. Since we always try shallower literals first, we will resolve Γ with Λ before Δ_i and with Δ_i before Δ_{i+1} for all $0 < i < n$. We evaluate the algorithm on each such iteration and define Ψ^i_{eval} to be the sum of the i^{th} iteration. For $i = 1$ we resolve Γ with Λ and obtain the minimal unifier $[\lambda z.z/y]$, then we resolve with Δ_1 and have to backtrack and try another unifier $e_1 + 1$ times, where the last attempt fails because we have reached the exponent of periodicity. Each of these backtrackings computes a substitution with y mapped to a term of depth $0 \leq k \leq e_1$ for all k. Therefore $\Psi^1_{eval} = \Sigma_{j=1..e_1} j$. For $i > 1$, we resolve first Γ with Δ_{i-1}, then with Λ and then with all Δ_l for $0 < l \leq i-2$. The exponent of periodicity computed for the resolvent of Γ with Λ is now based also on the size of Δ_{i-1}, i.e. it is equal to k for $e_{i-1} \leq k < e_i$. After calling $\texttt{(Imitate)}$ and $\texttt{(Decomp)}$ e_{i-1} times we get the constraints $[y'c \doteq y_{i-1}]^{\mathrm{F}}$ and $[y'ac \doteq ay'c]^{\mathrm{F}}$. Now, we can resolve with all Δ_l for $0 < l \leq n-2$ and just have to compute the values for the first-order variables y_l as all of them are unifiable with the substitution obtained above. Last, we resolve with Δ_i and have to backtrack $e_i - e_{i-1} + 1$ times and fail the last time as the exponent is e_{i-1}. For each of this backtracks, we compute substitutions where y is mapped to $e_i - e_{i-1} < k \leq e_i$ and therefore $\Psi^i_{eval} = e_{i-1} + \Sigma_{j=e_{i-1}..e_i} j$. Since each iteration for $i < n$ fails, we can sum up all the Ψ^i_{eval} and get the required value. □

Note that it is possible to modify $\texttt{CRC}_\lambda$ such that unification will be applied lazily, as is done normally in $\texttt{CRC}$. In this case there will not be any backtracking as in the above case but, as we already discussed in the introduction of this chapter, such a calculus will be highly impractical.

Definition 5.2.25 (Lazy minimal binders resolution calculus ($\mathtt{CRC}_\lambda^{lazy}$))**.** The lazy minimal binders resolution calculus has the same rules as $\mathtt{CRC}_\lambda$ with the exception of applying unification rules only on clauses which contain unification constraints only.

Theorem 5.2.26. If a clause set S is refutable by $\mathtt{CRC}$ such that the derivation substitution is bounded by $\hat{b}$, then we can refute it using $\mathtt{CRC}_\lambda^{lazy}$ and using $\hat{b}$.

Proof. The only difference between the calculi is in the set of unification rules. We can use Lemma 2.3.15 in order to obtain the same derivation of a clause C in both calculi such that C is pre-unifiable. Clearly, if it is pre-unifiable by a bounded substitution, then we can obtain the refutation with $\mathtt{CRC}_\lambda^{lazy}$. □

Corollary 5.2.27 (Completeness)**.** If a clause set is unsatisfiable with regard to V-complexes such that the substitution used in a counter-model is bounded by $\hat{b}$, then we can refute it using $\mathtt{CRC}_\lambda^{lazy}$and using $\hat{b}$.

Proof. Following theorems 2.3.17 and 5.2.26. □

Corollary 5.2.28. If a clause set S is refutable using $\mathtt{CRC}_\lambda^{lazy}$ then we can obtain a refutation of S containing a clause C, such that below it we have only unification rule applications and above it only rule applications from the other sets.

Proof. Following Lemma 2.3.15 and Thm. 5.2.26. □

5.3 Regular Binders Resolution

In this section we will define a resolution calculus which will be based on the regular unification algorithm from Chap. 4.

The constrained resolution calculus postpones the computation of unifiers but also allows for unification to be applied eagerly in very few cases, such as for first-order constraints or any other constrained which is of finitary class. Such classes are described further in [61] and [67]. Since we can consider the unification constraints as a finite representation of the infinitely-many unifiers, in the general case we postpone unification in order to keep this finite representation.

When dealing with clause sets which can be refuted using a bounded substitution, like the ones described in Sec. 5.2, postponing unification may have no real advantage over eager unification. The gain we have for keeping a compact representation of the (now finitely-many) unifiers, may be no more than the additional price we pay for searching in branches which have non-unifiable sets of constraints.

On the other hand, we have seen in Lemma. 5.2.24 that restricting the set of unifiers may leads to an inefficient search. The search could be more efficient if we could obtain two things. First, we could avoid the re-computation of the same substitutions and second, we could try to guess the “right” exponent, which in this case was e_n.

The calculus introduced in this section allows us to search for refutations while enjoying these two things. The re-computation of the same substitutions is prevented by using regular terms while guessing the "right" exponents is obtained by partially postponing unification and using different values for the computation of the exponents.

In order to allow for different values to be used in the computation of the exponents, we will need to use information about the state of the search. This will be obtained by keeping track of the search using a graph.

Definition 5.3.1 (Constraints of a clause)**.** Given a clause C, the set of all unification constraints of C is denoted by $\texttt{const}(C)$.

Example 5.3.2. *The set of constraints of the clause:*

$$[P(a)]^T \vee [xfa \doteq fxa]^F \tag{5.9}$$

is $\{[xfa \doteq fxa]^F\}$*.*

Definition 5.3.3 (Search graphs)**.** Given a clause set C, a search graph for C is a directed acyclic graph, with labeling function $\texttt{lbl}$ from nodes to clauses such that:

- the root nodes are labeled by clauses from C.
- if there is an edge from node v_1 to node v_2 then the clause $\texttt{lbl}(v_2)$ can be obtained from clause $\texttt{lbl}(v_1)$ using one rule from the sets defined in definitions 2.3.8, 2.3.9 and 2.3.10.

A full search graph is a search graph that in addition has:

- if there are nodes v and v_1 and $\texttt{lbl}(v)$ is obtained from $\texttt{lbl}(v_1)$ and some clause c by a binary rule, then there is a node v_2 such that $\texttt{lbl}(v_2) = c$ and there is an edge from v_2 to v.

Note that since we might have many ways to derive each clause using the allowed rules, we might also have many edges coming into each node in the search graph.

Example 5.3.4. *In this section we will use, as a running example, clause set 4 (*see *Df. 5.1.8) for* $n = 2$*. A full search graph for the clause C:*

$$[P_2(yc) \doteq P_2(\overbrace{a..a}^{e_2}y_2)]^F \vee [P_1(yc) \doteq P_1(\overbrace{a..a}^{e_1}y_1)]^F \vee [Q(yac, ayc) \doteq Q(z,z)]^F \tag{5.10}$$

can be seen in Fig. 5.7 where:

- *C_1 is $[P_1(yc)]^T \vee [P_2(yc)]^T \vee [Q(yac, ayc) \doteq Q(z,z)]^F$ and*
- *C_2 is $[P_2(yc)]^T \vee [P_1(yc) \doteq P_1(\overbrace{a..a}^{e_1}y_1)]^F \vee [Q(yac, ayc) \doteq Q(z,z)]^F$*
- *all rules applications are* $\texttt{(Resolve)}$*.*

The intuition behind the search graphs is to keep information about the possibly infinitely-many unifiers, even when we apply unification eagerly and compute a finite subset of them. The search graphs factor out all eager applications of unification rules.

We will also define the following function on nodes in search graphs.

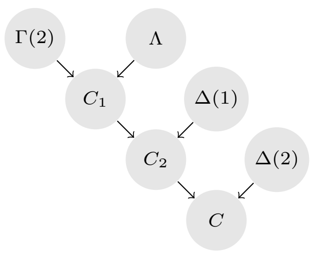


Figure 5.7: A full search graph for clause C

Definition 5.3.5 (Maximal descendant (`maxDesc`))**.** Given a node v, its maximal descendant (`maxDesc`) is the node whose label has the maximal exponent of periodicity of all nodes which are descendants of v.

Example 5.3.6. *In the tree from Fig. 5.7, the node labeled by C is the maximal descendant of all other nodes.*

In order to use this information, we will keep one search graph for the whole search for a refutation and will add new nodes to it, whenever new clauses are found. In addition, the environment of each clause will contain a reference to one node in the search graph.

We will also define several new functions and update the definitions of some others.

Definition 5.3.7. The following definitions will be used in the remaining part of this chapter:

- Given a clause C, $\texttt{eop}(C)$ is the exponent of all the unification constraints in C.
- Given a clause C, $\texttt{env}(C)$ is the environment of C.
- We assign, to each clause, a node on the search graph. Given a clause C, E^n is the node corresponding to C where $E = \texttt{env}(C)$.
- $\texttt{derail}(E, x, s, c) = \texttt{derail}_{\texttt{const(lbl(maxDesc(}E^n\texttt{)))}}(x, s, c)$ (see Def. 4.5.5).

Example 5.3.8. *Since a descendant node always has at least the same exponent of periodicity as any of its ancestors (as no unification is applied in the search graph), the exponent of periodicity value used in the* `derail` *function for all nodes in Fig 5.7 will be the same as that of the maximal descendant and therefore will equal the exponent of periodicity of C.*

Since we deal with clauses instead of unification equations as in Chap. 4, we will update some definitions.

Definition 5.3.9 (First-order bound)**.** Given a clause C, its first-order bound is $(k + 1) \cdot \texttt{md}(C)$, where k is the number of unsolved variables in $\texttt{const}(C)$ and $\texttt{md}(C)$ is the maximal size of a rigid position in any term in $\texttt{const}(C)$. It is denoted by $\texttt{fbound}(C)$.

Definition 5.3.10 (Some utility functions)**.** In order to make the presentation of the rules simpler, we will use the following utility functions:

- given an environment E and a clause C, $\texttt{reset}(E, C)$ is the environment obtained from E by deleting all values $E[r]$ and setting all values $E[d]$ to $2 \cdot \texttt{fbound}(C)$. The other values do not change.
- let C be a clause derived using rule r and clauses C_1,..,C_m, then $f_n(E_1, .., E_m)$ where $E_1 = \texttt{env}(C_1), .., E_m = \texttt{env}(C_m)$ returns an environment which contains:
  - a pointer E^n, in the environment E, to the node which is the child of parents $E^n_1, .., E^n_m$ using rule r. Note that if the node does not exist then we create it.
  - $E[d(x) = 2 \cdot \texttt{fbound}(C)]$ for all higher-order variables x in $\texttt{const}(C)$.
  - if $E_i[\hat{b}(x) = b]$ for some $0 < i \leq n$ and variable x, then $E[\hat{b}(x) = b]$. Note that since we always take variants, the sets of variables in the environments are disjoint.

Example 5.3.11. *Let C be the clause:*

$$[xa \doteq fffa]^F \tag{5.11}$$

then $f_n(E)$ results in E, where we delete all regular and depth constraints for x and add $E[d(x) = 16]$.

We will now present the resolution calculus. The rules in figures 5.8 and 5.9, and 5.10 are exactly as those in definitions 2.3.8, 2.3.9 and 2.3.10 except for the added environment.

Definition 5.3.12 (Regular binders resolution calculus($\texttt{CRC}_R$))**.** The regular binders resolution calculus contains the rules from figures 5.8, 5.9, 5.10 and 5.11 such that unification is always applied eagerly. Axioms C are assigned environments E such that:

- for each higher-order variable x, $E[d(x) = 2 \cdot \texttt{fbound}(C)]$.
- for each clause C and its environment $E = \texttt{env}(C)$, $E^n = v$ such that $\texttt{lbl}(v) = C$.

If the unification constraints in a clause are not in pre-solved form but there is no unification rule applicable, then we fail.

$$\frac{E \vdash C \vee [\neg D]^{\mathtt{T}}}{f_n(E) \vdash C \vee [D]^{\mathtt{F}}}\ (\neg^T) \qquad \frac{E \vdash C \vee [\neg D]^{\mathtt{F}}}{f_n(E) \vdash C \vee [D]^{\mathtt{T}}}\ (\neg^F)$$

$$\frac{E \vdash C \vee [D_1 \vee D_1]^{\mathtt{T}}}{f_n(E) \vdash C \vee [D_1]^{\mathtt{T}} \vee [D_2]^{\mathtt{T}}}\ (\vee^T) \qquad \frac{E \vdash C \vee [D_1 \vee D_2]^{\mathtt{F}}}{f_n(E) \vdash C \vee [D_1]^{\mathtt{F}}}\ (\vee^F_l) \qquad \frac{E \vdash C \vee [D_1 \vee D_2]^{\mathtt{F}}}{f_n(E) \vdash C \vee [D_2]^{\mathtt{F}}}\ (\vee^F_r)$$

$$\frac{E \vdash C \vee [\Pi_\alpha A]^{\mathtt{T}}}{f_n(E) \vdash C \vee [Ax^\alpha]^{\mathtt{T}}}\ (\Pi^T) \qquad \frac{E \vdash C \vee [\Pi_\alpha A]^{\mathtt{F}}}{f_n(E) \vdash C \vee [As_\alpha]^{\mathtt{T}}}\ (\Pi^F)^1$$

1. s_α is a new Skolem term

Figure 5.8: Simplification Rules

$$\frac{E_1 \vdash [A]^p \vee C \qquad E_2 \vdash [B]^{\neg p} \vee D}{f_n(E_1, E_2) \vdash C \vee D \vee [A \doteq B]^{\mathtt{F}}}\ \texttt{(Resolve)} \qquad \frac{E \vdash [A]^p \vee [B]^p \vee C}{f_n(E) \vdash [A]^p \vee C \vee [A \doteq B]^{\mathtt{F}}}\ \texttt{(Factor)}$$

Figure 5.9: Resolution and factorization rules

$$\frac{E \vdash C \vee [x(\overline{t_n})]^{\mathtt{T}}}{f_n(E) \vdash C \vee [y]^{\mathtt{T}} \vee [z]^{\mathtt{T}} \vee [x(\overline{t_n}) \doteq (y \vee z)]^{\mathtt{F}}}\ (S^T_\vee) \qquad \frac{E \vdash C \vee [x(\overline{t_n})]^p}{f_n(E) \vdash C \vee [y]^{\neg p} \vee [x(\overline{t_n}) \doteq \neg y)]^{\mathtt{F}}}\ (S^{TF}_\neg)$$

$$\frac{E \vdash C \vee [x(\overline{t_n})]^{\mathtt{F}}}{f_n(E) \vdash C \vee [y]^{\mathtt{F}} \vee [x(\overline{t_n}) \doteq (y \vee z)]^{\mathtt{F}}}\ (S^{F_l}_\vee) \qquad \frac{E \vdash C \vee [x(\overline{t_n})]^{\mathtt{F}}}{f_n(E) \vdash C \vee [z]^{\mathtt{F}} \vee [x(\overline{t_n}) \doteq (y \vee z)]^{\mathtt{F}}}\ (S^{F_r}_\vee)$$

$$\frac{E \vdash C \vee [x(\overline{t_n})]^{\mathtt{T}}}{f_n(E) \vdash C \vee [yz^\alpha]^{\mathtt{T}} \vee [x(\overline{t_n}) \doteq \Pi_\alpha y]^{\mathtt{F}}}\ (S^T_\Pi) \qquad \frac{E \vdash C \vee [x(\overline{t_n})]^{\mathtt{F}}}{f_n(E) \vdash C \vee [ys_\alpha]^{\mathtt{F}} \vee [x(\overline{t_n}) \doteq \Pi_\alpha y]^{\mathtt{F}}}\ (S^F_\Pi)^1$$

1. s_α is a new Skolem term

Figure 5.10: Splitting Rules

Definition 5.3.13 (Search graphs and derivations)**.** Whenever a new node v in the graph is added, we do the following:

1. let $v_1, .., v_n$ be all ancestors of v.
2. let $m_i = \texttt{maxDesc}(v_i)$ for all $0 < i \le n$ where we exclude v from the search for max-descendants (i.e. using the search graph before the addition of v).
3. let $u_i = \texttt{eop}(\texttt{lbl}(v)) - \texttt{eop}(\texttt{lbl}(m_i))$ for all $0 < i \le n$.
4. let $c_1, .., c_k$ be all the clauses in the clause set.
5. if $\texttt{env}(c_j)^n = v_i$ for all $0 < j \le k$ and some $0 < i \le n$, if $u_i > 0$ then add u_i to all exponents in binding constraints in $\texttt{env}(c_j)$.

Remark 5.3.14. Although the above process may be expensive if done naively, it can be optimized in practice by keeping the exponents as variables.

Example 5.3.15. *Let v_1 and v_2 be the parents of the newly added node v and let $\texttt{eop}(\texttt{lbl}(v_1)) = e_1$, $\texttt{eop}(\texttt{lbl}(v_2)) = e_2$ and $\texttt{eop}(\texttt{lbl}(v)) = e$. Therefore, $u_1 = e - e_1$ and $u_2 = e - e_2$. Now assume that $\texttt{env}(c_1)^n = \texttt{env}(c_2)^n = v_1$, $\texttt{env}(c_3)^n = v_2$ for clauses c_1, c_2, c_3 and there are the binding constraints $r_1, \ldots, r_k$ in $\texttt{env}(c_1)^n$ and $\texttt{env}(c_2)^n$ and the binding constraints $r'_1, \ldots, r'_l$ in $\texttt{env}(c_3)^n$. If $u_1 > 0$ but $u_2 = 0$, then we add u_1 to all the exponents in $r_1, \ldots, r_k$ and do not change those in $r'_1, \ldots, r'_l$.*

Example 5.3.16. *Let S be clause set 4 and assume $\hat{b}(Y) = 2$. We will refute clause set 4 (*see *Df. 5.1.8) for $n = 2$. The respective environments are given in Table 5.1. The labels of the nodes are given in Table 5.2. The search for a refutation will be carried over in 3 steps. For each step we will give the derivation and the search graph (figures 5.12, 5.14 and 5.16 for derivations and figures 5.13 and 5.15 for search graphs). Please note that we sometimes change the environments of some clauses after they were derived, so we might derive a clause $E \vdash \Gamma$ but use in the example clause $E' \vdash \Gamma$. In order to save space and since the environment does not always change using derivation rules, we might denote two different clauses $E \vdash \Gamma$ and $E \vdash \Gamma'$ using the same environment E.*

1. *in the first step we derive the clause*

$$E_8 \vdash [P_2(c)]^T \vee [c \doteq \overbrace{a..a}^{e_1} y_1]^F \tag{5.12}$$

as can be seen in Fig. 5.12. Since the unification constraints are not in pre-solved form but no unification rule is applicable, we fail. The addition of node n_5 to the graph (see Fig. 5.13) triggers Def. 5.3.13 and therefore the clause

$$E_4 \vdash [P_1(Yc)]^T \vee [P_2(Yc)]^T \vee [Yac \doteq aYc]^F \tag{5.13}$$

is modified into clause

$$c = E_9 \vdash [P_1(Yc)]^T \vee [P_2(Yc)]^T \vee [Yac \doteq aYc]^F \tag{5.14}$$

and we continue with step 2.

$$\frac{E \vdash C \vee [A \doteq A]^{\mathtt{F}}}{E \vdash C}\ \texttt{(Delete)} \qquad \frac{E \vdash C \vee [\lambda\overline{z_k}.f(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_n})]^{\mathtt{F}}}{E \vdash C \vee [\lambda\overline{z_k}.s_1 \doteq \lambda\overline{z_k}.t_1]^{\mathtt{F}} \vee .. \vee [\lambda\overline{z_k}.s_n \doteq \lambda\overline{z_k}.t_n]^{\mathtt{F}}}\ \texttt{(Decomp)}$$

$$\frac{E[r(x) = \lambda\overline{z_k}.t^n(t_r)] \vdash C}{E[r(x) = \lambda\overline{z_k}.t_r] \vdash C}\ \texttt{(Skip)} \qquad \frac{E \vdash C \vee [\lambda\overline{z_k}.x(\overline{z_k}) \doteq \lambda\overline{z_k}.t]^{\mathtt{F}} \qquad x \notin \mathtt{FV}(t), \sigma = [\lambda\overline{z_k}.t/x]}{E\sigma \vdash C\sigma \vee [x \doteq \lambda\overline{z_k}.t]^{\mathtt{F}}}\ \texttt{(Bind)}$$

$$\frac{E \vdash C \qquad [\lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m})]^{\mathtt{F}} \in C, 0 < i \leq k, u = \mathtt{PB}^b(i,\alpha), \sigma = [u/x]}{\mathtt{reset}(E\sigma, C\sigma) \vdash C\sigma \vee [x \doteq u]^{\mathtt{F}}}\ \texttt{(Project)}^2$$

$$\frac{E[r(x) = \epsilon] \vdash C \qquad c \in \mathtt{scy}(C), x \in c, t \in \mathtt{derail}(E, x, \mathtt{size}(c), \mathtt{ccon}(c))}{E[r(x) = t] \vdash C}\ \texttt{(Rec)}$$

$$\frac{E[r(x) = \epsilon, d(x) = m > 0] \vdash C \qquad [\lambda\overline{z_k}.x^{\alpha}(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_p})]^{\mathtt{F}} \in C, u = \mathtt{PB}^b(f,\alpha), \sigma = [u/x]}{E[d(x_1) = m-1, .., d(x_p) = m-1]\sigma \vdash C\sigma \vee [x \doteq u]^{\mathtt{F}}}\ \texttt{(Imitate}_{pb}\texttt{)}^1$$

$$\frac{E[r(x) = \lambda\overline{z_n}.t^l(t_r)] \vdash C \qquad [\lambda\overline{z_k}.x(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{v_p})]^{\mathtt{F}} \in C, t = f(t_1, .., t_k([.]), .., t_p), u = \mathtt{PB}^b(f,\alpha), \sigma = [u/x]}{E[r(x_1) = t_1, .., r(x_k) = t_k(t^{l-1}(t_r)), .., r(x_p) = t_p]\sigma \vdash C\sigma \vee [x \doteq u]^{\mathtt{F}}}\ \texttt{(Imitate}_*\texttt{)}^1$$

$$\frac{E[r(x) = \lambda\overline{z_n}.t] \vdash C \qquad [\lambda\overline{z_k}.x(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{v_p})]^{\mathtt{F}} \in C, t = f(t_1, .., t_p), u = \mathtt{PB}^b(f,\alpha), \sigma = [u/x]}{E[r(x_1) = t_1, .., r(x_p) = t_p]\sigma \vdash C\sigma \vee [x \doteq u]^{\mathtt{F}}}\ \texttt{(Imitate}_0\texttt{)}^1$$

1. the $x_1, .., x_p$ are all the new variables introduced in $\mathtt{PB}^b$.

2. a is either a function symbol or a bound variable.

Figure 5.11: Bounded unification rules

$$\dfrac{E_1 \vdash [Q(Yac, aYc)]^{\mathrm{T}} \vee [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \qquad E_2 \vdash [Q(z,z)]^{\mathrm{F}}}{E_3 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Q(Yac, aYc) \doteq Q(z,z)]^{\mathrm{F}}}\ \texttt{(Resolve)}$$

$$\dfrac{E_3 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Q(Yac, aYc) \doteq Q(z,z)]^{\mathrm{F}}}{E_3 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq z]^{\mathrm{F}} \vee [aYc \doteq z]^{\mathrm{F}}}\ \texttt{(Decomp)}$$

$$\dfrac{E_3 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq z]^{\mathrm{F}} \vee [aYc \doteq z]^{\mathrm{F}}}{E_3 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}\ \texttt{(Bind)}$$

$$\dfrac{E_3 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}{E_4 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}\ \texttt{(Rec)}$$

$$\dfrac{E_4 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}{E_5 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}\ \texttt{(Skip)}$$

$$\dfrac{E_5 \vdash [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Yac \doteq aYc]^{\mathrm{F}}}{E_6 \vdash [P_1(c)]^{\mathrm{T}} \vee [P_2(c)]^{\mathrm{T}} \vee [ac \doteq ac]^{\mathrm{F}}}\ \texttt{(Project)}$$

$$\dfrac{E_6 \vdash [P_1(c)]^{\mathrm{T}} \vee [P_2(c)]^{\mathrm{T}} \vee [ac \doteq ac]^{\mathrm{F}}}{E_6 \vdash [P_1(c)]^{\mathrm{T}} \vee [P_2(c)]^{\mathrm{T}}}\ \texttt{(Delete)}$$

$$\dfrac{E_6 \vdash [P_1(c)]^{\mathrm{T}} \vee [P_2(c)]^{\mathrm{T}} \qquad E_7 \vdash [P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}}{E_8 \vdash [P_2(c)]^{\mathrm{T}} \vee [P_1(c) \doteq P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}}\ \texttt{(Resolve)}$$

$$\dfrac{E_8 \vdash [P_2(c)]^{\mathrm{T}} \vee [P_1(c) \doteq P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}}{E_8 \vdash [P_2(c)]^{\mathrm{T}} \vee [c \doteq \overbrace{a..a}^{e_1} y_1]^{\mathrm{F}}}\ \texttt{(Decomp)}$$

Figure 5.12: Step 1 - derivation

2. *we use now the resolvent* c *and obtain the clause*

$$c_1 = E_{15} \vdash [c \doteq \overbrace{a..a}^{e_2 - m}\ y_2]^F \tag{5.15}$$

as can be seen in Fig. 5.14. After attempting to obtain the derivation for $m < e_1$ *and failing, we obtain it for* $m = e_1$*. The clause* c_1 *we obtain is, nevertheless, not in pre-solved form and there is no unification rule which is applicable. For the above to be correct, we need to prove that, since* m *is bounded by* e_1'*, then* $e_2 > e_1' \geq e_1$*. Following the definition of* e_0, e_1, e_2 *and the computations defined in definitions 5.3.12 and 5.3.13 we obtain that* $e_1' = e_2 - 1$ *which satisfies the above inequality. Since we added new nodes to the search graph (see Fig. 5.15), we update the clause*

$$E_{10} \vdash [P_1(\overbrace{a..a}^{m} Y_m c)]^T \vee [P_2(\overbrace{a..a}^{m} Y_m c)]^T \vee [Y_m ac \doteq aY_m c]^F \tag{5.16}$$

into clause

$$c_2 = E_{15} \vdash [P_1(\overbrace{a..a}^{m} Y_m c)]^T \vee [P_2(\overbrace{a..a}^{m} Y_m c)]^T \vee [Y_m ac \doteq aY_m c]^F \tag{5.17}$$

and continue with step 3.

3. *using the resolvent* c_2 *and similar rule applications to the ones in Fig. 5.14, we obtain the empty clause (see Fig. 5.16). We just note that we need to repeat applying* `(Imitate`$_*$`)` *for* $e_2 - m$ *more times and since* $e_2' > e_2$ *(using the same argument as above), we can delete this unification constraint.*

Definition 5.3.17 (Skeletons (of derivations))**.** A skeleton of a derivation D is a tree `skeleton`(D) created recursively as follows:

- if D is an axiom then `skeleton`$(D) = D$.

$$\begin{tikzcd}
n_1 \arrow[dr] & & n_2 \arrow[dl] & \\
& n_3 \arrow[dr] & & n_4 \arrow[dl] \\
& & n_5 &
\end{tikzcd}$$

Figure 5.13: Step 1 - search graph

env	$\hat{b}$	d	r	n	comment
E_1	$\hat{b}(Y) = 2$	$\emptyset$	$\emptyset$	n_1	
E_2	$\emptyset$	$\emptyset$	$\emptyset$	n_2	
E_3	$\hat{b}(Y) = 2$	$d(Y) = 18$	$\emptyset$	n_3	
E_4	$\hat{b}(Y) = 2$	$\emptyset$	$r(Y) = \lambda z.a^{e'_0} z$	n_3	$e'_0 = \mathtt{eop}(\mathtt{lbl}(n_3))$
E_5	$\hat{b}(Y) = 2$	$\emptyset$	$r(Y) = \lambda z.z$	n_3	
E_6	$\emptyset$	$\emptyset$	$\emptyset$	n_3	
E_7	$\emptyset$	$\emptyset$	$\emptyset$	n_4	
E_8	$\emptyset$	$\emptyset$	$\emptyset$	n_5	
E_9	$\hat{b}(Y) = 2$	$\emptyset$	$r(Y) = \lambda z.a^{e'_1} z$	n_3	$e'_1 = \mathtt{eop}(\mathtt{lbl}(n_5))$
E_{10}	$\hat{b}(Y_m) = 2$	$\emptyset$	$r(Y_m) = \lambda z.a^{e'_1 - m} z$	n_3	
E_{11}	$\hat{b}(Y_m) = 2$	$\emptyset$	$r(Y_m) = \lambda z.z$	n_3	
E_{12}	$\emptyset$	$\emptyset$	$\emptyset$	n_3	
E_{13}	$\emptyset$	$\emptyset$	$\emptyset$	n_6	
E_{14}	$\emptyset$	$\emptyset$	$\emptyset$	n_7	
E_{15}	$\hat{b}(Y_m) = 2$	$\emptyset$	$r(Y_m) = \lambda z.a^{e'_2 - m} z$	n_3	$e'_2 = \mathtt{eop}(\mathtt{lbl}(n_7))$
E_{16}	$\hat{b}(Y_k) = 2$	$\emptyset$	$r(Y_k) = \lambda z.a^{e'_2 - k} z$	n_3	
E_{17}	$\hat{b}(Y_k) = 2$	$\emptyset$	$r(Y_k) = \lambda z.z$	n_3	
E_{18}	$\emptyset$	$\emptyset$	$\emptyset$	n_3	

Table 5.1: Environment's values

node	label
n_1	$[Q(Yac, aYc)]^{\mathrm{T}} \vee [P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}}$
n_2	$[Q(z, z)]^{\mathrm{F}}$
n_3	$[P_1(Yc)]^{\mathrm{T}} \vee [P_2(Yc)]^{\mathrm{T}} \vee [Q(Yac, aYc) \doteq Q(z, z)]^{\mathrm{F}}$
n_4	$[P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}$
n_5	$[P_2(Yc)]^{\mathrm{T}} \vee [Q(Yac, aYc) \doteq Q(z, z)]^{\mathrm{F}} \vee [P_1(Yc) \doteq P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}$
n_6	$[P_2(\overbrace{a..a}^{e_2} y_2)]^{\mathrm{F}}$
n_7	$[Q(Yac, aYc) \doteq Q(z, z)]^{\mathrm{F}} \vee [P_1(Yc) \doteq P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}} \vee [P_2(Yc) \doteq P_2(\overbrace{a..a}^{e_2} y_2)]^{\mathrm{F}}$

Table 5.2: Search graph's labels

$$\dfrac{E_9 \vdash [P_1(Y c)]^{\mathrm{T}} \vee [P_2(Y c)]^{\mathrm{T}} \vee [Y ac \doteq aY c]^{\mathrm{F}}}{E_{10} \vdash [P_1(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [Y_m ac \doteq aY_m c]^{\mathrm{F}}}\ \texttt{(Imitate}_*\texttt{), (Decomp)}\times m$$

$$\dfrac{E_{10} \vdash [P_1(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [Y_m ac \doteq aY_m c]^{\mathrm{F}}}{E_{11} \vdash [P_1(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [Y_m ac \doteq aY_m c]^{\mathrm{F}}}\ \texttt{(Skip)}$$

$$\dfrac{E_{11} \vdash [P_1(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} Y_m c)]^{\mathrm{T}} \vee [Y_m ac \doteq aY_m c]^{\mathrm{F}}}{E_{12} \vdash [P_1(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [ac \doteq ac]^{\mathrm{F}}}\ \texttt{(Project)}$$

$$\dfrac{E_{12} \vdash [P_1(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [ac \doteq ac]^{\mathrm{F}}}{E_{12} \vdash [P_1(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}}}\ \texttt{(Delete)}$$

$$\dfrac{E_{12} \vdash [P_1(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \qquad E_7 \vdash [P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}}{E_8 \vdash [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [P_1(\overbrace{a..a}^{m} c) \doteq P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}}\ \texttt{(Resolve)}$$

$$\dfrac{E_8 \vdash [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [P_1(\overbrace{a..a}^{m} c) \doteq P_1(\overbrace{a..a}^{e_1} y_1)]^{\mathrm{F}}}{E_8 \vdash [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [\ \overbrace{a..a}^{m-e_1}\ c \doteq y_1]^{\mathrm{F}}}\ \texttt{(Decomp)}\times e_1$$

$$\dfrac{E_8 \vdash [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \vee [\ \overbrace{a..a}^{m-e_1}\ c \doteq y_1]^{\mathrm{F}}}{E_8 \vdash [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}}}\ \texttt{(Bind)}$$

$$\dfrac{E_8 \vdash [P_2(\overbrace{a..a}^{m} c)]^{\mathrm{T}} \qquad E_{13} \vdash [P_2(\overbrace{a..a}^{e_2} y_2)]^{\mathrm{F}}}{E_{14} \vdash [P_2(\overbrace{a..a}^{m} c) \doteq P_2(\overbrace{a..a}^{e_2} y_2)]^{\mathrm{F}}}\ \texttt{(Resolve)}$$

$$\dfrac{E_{14} \vdash [P_2(\overbrace{a..a}^{m} c) \doteq P_2(\overbrace{a..a}^{e_2} y_2)]^{\mathrm{F}}}{E_{14} \vdash [\overbrace{a..a}^{m} c \doteq \overbrace{a..a}^{e_2} y_2]^{\mathrm{F}}}\ \texttt{(Decomp)}$$

$$\dfrac{E_{14} \vdash [\overbrace{a..a}^{m} c \doteq \overbrace{a..a}^{e_2} y_2]^{\mathrm{F}}}{E_{14} \vdash [c \doteq\ \overbrace{a..a}^{e_2-m}\ y_2]^{\mathrm{F}}}\ \texttt{(Decomp)}\times m$$

Figure 5.14: Step 2 - derivation

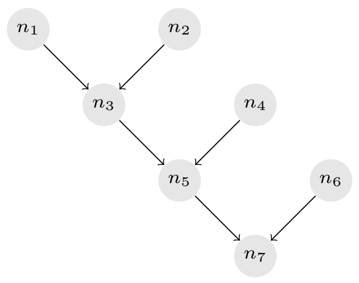


Figure 5.15: Steps 2 and 3 - search graph

- if D is a rule ρ applied to upper derivations $D_1, .., D_n$ and resulting in clause C then:
  - if ρ is from figures 2.3.8, 2.3.9 and 2.3.10, then $\texttt{skeleton}(D)$ has C as child and $\texttt{skeleton}(D_1), .., \texttt{skeleton}(D_n)$ as parents.
  - else ρ is from figure 5.11 and therefore $n = 1$. In this case we take $\texttt{skeleton}(D) = \texttt{skeleton}(D_1)$.

Example 5.3.18. *The skeleton of the derivation from Fig. 5.12 is equal to the tree in Fig. 5.13.*

Lemma 5.3.19. For any derivation D using $\texttt{CRC}_R$, the exponents of periodicity of the terms in D do not exceed $\texttt{eop}(C)$, where C is the lowermost clause (the root clause) in $\texttt{skeleton}(D)$.

Proof. The bounds on the exponents depend on the the function $\texttt{maxDesc}(v)$ for v the node corresponding to any clause in the derivation. If we add later new nodes, we update these values (see Def. 5.3.13). According to the definitions of search graphs and skeletons, C is in the search graph and a descendant of all nodes corresponding to clauses in the derivation and therefore $\texttt{eop}(\texttt{maxDesc}(v)) \geq \texttt{eop}(C)$. $\square$

Lemma 5.3.20. For any derivation D of a clause C which is obtained using $\texttt{CRC}_\lambda^{lazy}$ without the application of unification and such that $\texttt{const}(C)$ is unifiable by σ (using $\texttt{CRC}_\lambda^{lazy}$), we can obtain a derivation D' of the clause $C\sigma$ using $\texttt{CRC}_R$.

Proof. Let $Sk = \texttt{skeleton}(D)$ ($= D$ since we have no application of a unification rule). By an induction on Sk, we build a valid derivation of $C\sigma$ using $\texttt{CRC}_R$, note that since the rules from Fig. 5.11 are sound, complete and terminating, we can obtain each unifier.

- if Sk is an axiom, we can derive it also using $\texttt{CRC}_R$.
- if Sk is a rule application which does not introduce new unification constraints or the introduced unification constraints are in pre-solved form then we can apply the same rule using $\texttt{CRC}_R$.
- otherwise, Sk is a rule application which introduces unification constraints not in a pre-solved form. Since we do not solve unification constraints when using $\texttt{CRC}_\lambda^{lazy}$, these

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{
            \dfrac{
              \dfrac{
                E_{15} \vdash [P_1(\overbrace{a..a}^{m}\,Y_m c)]^{\mathtt{T}} \vee [P_2(\overbrace{a..a}^{m}\,Y_m c)]^{\mathtt{T}} \vee [Y_m ac \doteq aY_m c]^{\mathtt{F}}
              }{
                E_{16} \vdash [P_1(\overbrace{a..a}^{k}\,Y_k c)]^{\mathtt{T}} \vee [P_2(\overbrace{a..a}^{k}\,Y_k c)]^{\mathtt{T}} \vee [Y_k ac \doteq aY_k c]^{\mathtt{F}}
              }\;\texttt{(Imitate}_*\texttt{), (Decomp)}\times(k-m)
            }{
              E_{17} \vdash [P_1(\overbrace{a..a}^{k}\,Y_k c)]^{\mathtt{T}} \vee [P_2(\overbrace{a..a}^{k}\,Y_k c)]^{\mathtt{T}} \vee [Y_k ac \doteq aY_k c]^{\mathtt{F}}
            }\;\texttt{(Skip)}
          }{
            E_{18} \vdash [P_1(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}} \vee [P_2(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}} \vee [ac \doteq ac]^{\mathtt{F}}
          }\;\texttt{(Project)}
        }{
          E_{18} \vdash [P_1(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}} \vee [P_2(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}}
        }\;\texttt{(Delete)}
        \qquad
        E_7 \vdash [P_1(\overbrace{a..a}^{e_1}\,y_1)]^{\mathtt{F}}
      }{
        E_8 \vdash [P_2(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}} \vee [P_1(\overbrace{a..a}^{k}\,c) \doteq P_1(\overbrace{a..a}^{e_1}\,y_1)]^{\mathtt{F}}
      }\;\texttt{(Resolve)}
    }{
      E_8 \vdash [P_2(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}} \vee [\overbrace{a..a}^{k-e_1}\;c \doteq y_1]^{\mathtt{F}}
    }\;\texttt{(Decomp)}\times e_1
  }{
    E_8 \vdash [P_2(\overbrace{a..a}^{k}\,c)]^{\mathtt{T}}
  }\;\texttt{(Bind)}
  \qquad
  E_{13} \vdash [P_2(\overbrace{a..a}^{e_2}\,y_2)]^{\mathtt{F}}
}{
  \dfrac{
    \dfrac{
      \dfrac{
        E_{14} \vdash [P_2(\overbrace{a..a}^{k}\,c) \doteq P_2(\overbrace{a..a}^{e_2}\,y_2)]^{\mathtt{F}}
      }{
        E_{14} \vdash [\overbrace{a..a}^{k}\,c \doteq \overbrace{a..a}^{e_2}\,y_2]^{\mathtt{F}}
      }\;\texttt{(Decomp)}
    }{
      E_{14} \vdash [\overbrace{a..a}^{k-e_2}\;c \doteq y_2]^{\mathtt{F}}
    }\;\texttt{(Decomp)}\times e_2
  }{}\;\texttt{(Bind)}
}\;\texttt{(Resolve)}
$$

Figure 5.16: Step 3 - derivation

unification constraints also occur in C and as C is unifiable, there is a substitution σ which unifies them such that the exponents of periodicity of the terms in the range of σ are bounded by $\texttt{eop}(C)$. Let the upper clauses of Sk be $C_1, .., C_n$. Since $\texttt{const}(C_i)$ is a subset of $\texttt{const}(C)$ for all $0 < i \leq n$ (we do not solve unification constraints), it is unifiable by some $\sigma_i \leq \sigma$ for $0 < i \leq n$. We now use the induction hypothesis in order to obtain the clauses $C_1\sigma_i, .., C_n\sigma_n$ using $\texttt{CRC}_R$. Let θ be such that $\sigma = \sigma_1 \circ .. \circ \sigma_n \circ \theta$. Then, θ pre-unifies $\texttt{const}(C_1\sigma_i), .., \texttt{const}(C_n\sigma_n)$. Clearly, the exponents of periodicity of the terms in the range of θ do not exceed $\texttt{eop}(C)$ and we can compute this substitution using $\texttt{CRC}_R$ (using Lemma 5.3.19). We therefore obtain a derivation of $C\sigma$ using $\texttt{CRC}_R$.

□

Theorem 5.3.21. Let S be a clause set and assume there is a refutation of S using $\texttt{CRC}_\lambda^{lazy}$ with a substitution σ, then running $\texttt{CRC}_R$ on S we obtain a refutation of S with substitution σ.

Proof. Using Cor. 5.2.28 we can obtain a refutation using $\texttt{CRC}_\lambda^{lazy}$ and a clause C in the refutation such that above it we have only rules from figures 2.4, 2.3 and 2.5 and below it only unification rules. Since we can obtain a refutation, C is pre-unifiable by some substitution σ, we can use Lemma 5.3.20 in order to obtain a derivation D of $C\sigma$ using $\texttt{CRC}_R$. Since C is pre-unifiable by σ and contains only unification constraints, $C\sigma$ is in pre-solved form and D is a refutation. □

Corollary 5.3.22 (Completeness)**.** If a clause set is unsatisfiable with regard to V-complexes such that the substitution used in the counter-model is bounded by $\hat{b}$, then we can refute it using $\texttt{CRC}_R$ and using $\hat{b}$.

Proof. Following Cor. 5.2.27 and Thm. 5.3.21. □

Lemma 5.3.23. Using $\texttt{CRC}_R$ we can obtains a refutation for each of the clause sets 1, 2 and 3.

Proof. By eagerly running the unification algorithm from Fig. 5.11 we can obtain each of the unifiers. □

Lemma 5.3.24. Using $\texttt{CRC}_R$ we can obtains a refutation for clause set 4 with $\Psi_{eval} = e_n$.

Proof. In Ex. 5.3.16 we have shown the application of $\texttt{CRC}_R$ on clause set 4 with $n = 2$ and it is easy to see that with the choice of clauses done in the example, $\Psi_{eval} = e_n$. We need to justify our order of choosing clauses. The choice of the two negative clauses is clear as the second one has deeper literals than the first. It remains to argue why we always, when backtracking, choose the deepest resolvent we obtained so far of the forms

$$E_{10} \vdash [P_1(\overbrace{a..a}^{m}Y_m c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{m}Y_m c)]^{\mathrm{T}} \vee [Y_m ac \doteq aY_m c]^{\mathrm{F}} \tag{5.18}$$

for the second step and

$$E_{16} \vdash [P_1(\overbrace{a..a}^{k}Y_k c)]^{\mathrm{T}} \vee [P_2(\overbrace{a..a}^{k}Y_k c)]^{\mathrm{T}} \vee [Y_k ac \doteq aY_k c]^{\mathrm{F}} \tag{5.19}$$

for the third step of the search. If we consider each of these clauses and as we are searching breadth-first, we notice that we cannot use the rule `(Imitate`$_*$`)` on the clause more than once and the only other rule applicable in this case is `(Skip)`, which does not affect Ψ_{eval}. □

Corollary 5.3.25 (Speedup)**.** The calculus $\texttt{CRC}_R$ has a quadratic speedup over the calculus $\texttt{CRC}_\lambda$ on clause set 4 when using the evaluation function Ψ_{eval}.

Proof. Using Lemma 5.2.24 ,we have $\Psi_{eval} = \Sigma_{i=1..n}(e_{i-1} + \Sigma_{j=e_{i-1}..e_i} j) = \Sigma_{i=0..n} e_i + \Sigma_{j=0..e_n} j$ when running $\texttt{CRC}_\lambda$. By using the Gauss sum formula we obtain that $\Psi_{eval} > \frac{e_n^2}{2}$ when running $\texttt{CRC}_\lambda$. Since Lemma 5.3.24 tells us that $\Psi_{eval} = e_n$ when running $\texttt{CRC}_R$, we obtain a quadratic speedup using the evaluation function. □

5.4 Conclusion

In this chapter we have investigated several resolution calculi, each with a different set of unification rules and different unification strategy.

The original constrained resolution calculus, which was introduced by Huet, is complete with respect to any unsatisfiable set of clauses. The calculus deals with the possibility of having infinitely-many pre-unifiers at each step by postponing all unification rules (with some exceptions). The disadvantage of this strategy, which we call lazy unification, is that many of the derivations are of non-unifiable clauses. It is trivial to prove that a calculus with an eager unification strategy might have any form of speedup over a lazy strategy. Moreover, since we normally proceed using a breadth-first search, the search space in this case is huge.

The solution many implementations of higher-order theorem provers adapt is to restrict the depth of terms generated by the unification algorithm. In this case unification generates finitely-many unifiers and can be applied eagerly. We call this algorithm fixed-depth unification. The disadvantage of such an approach is that it is applicable only to problems which are very shallow. The terms mapped to variables when unifying Church's numerals for example, cannot be restricted by such a bound. We give a simple example on which such a strategy fails to find a refutation.

We analyze the above case further and argue that the bound can be optimized if based on properties from the problem. We first show that using a problem-based bound, such as the first-order bound, is normally optimal to the fixed-depth one. This bound performs not so well when the arguments of higher-order variables contain too much information. If a bound on the number of possible projections in the derivation can be guessed in advance and if the problem contains no cycles, we claim that using the first-order bound eagerly is complete.

This leads us to the remaining calculi. These calculi assume each variable can be mapped to a term with at most a pre-defined number of bound variables. This restriction allows us to place a bound on the number of projections which can be applied in each derivation and use the first-order bound in order to compute a finite and complete (with regard to the bounds chosen) sets of pre-unifiers. The algorithms also treat with cyclic problems and extend a technique, first deployed by Makanin [59], that searches for minimal unifiers only. The first calculus we consider is based on the bounded higher-order unification algorithm [74]. We show that the restriction to minimal unifiers greatly harm the efficiency of this calculus and give a simple example on which this calculus performs in a sub-optimal way. Since the algorithm computes minimal unifiers with regard to local problems, it is also possible to show that a calculus which deploys this algorithm in an eager strategy is incomplete. I.e. even if the clause set is unsatisfiable (with a bounded substitution), there might be no refutation of it. When the unification strategy is changed into lazy unification, then the calculus is complete (with regard to the above definition) but as was already argued, may perform extremely bad due to the postponement of unification.

The last calculus we considered is based on eager unification and deploys the unification algorithm which was developed in Chap. 4. This calculus is also complete only with regard to clause sets which have a counter-model with a bounded substitution. But we show that the unification rules can always be applied eagerly without harming completeness. Moreover, we prove a quadratic speedup result over the calculus which is based on minimal unifiers.

We conclude the chapter and the thesis with a table comparing the results obtained in this chapter.

Calculus	Unification	Completeness	clause set 1	clause set 2	clause set 3	clause set 4
$\mathtt{CRC}$	lazy	full	non-termination	success	success	success
$\mathtt{CRC}_d$	eager	depth-bound	success	failure	failure	failure
$\mathtt{CRC}_{fol}$	eager	depth-bound	success	success	failure	success
$\mathtt{CRC}_p$	eager	projection-bound	success	success	failure	success
$\mathtt{CRC}_\lambda$	eager	projection-bound	success	success	success	$\Psi_{eval} = \frac{e_n^2}{2}$
$\mathtt{CRC}_\lambda^{lazy}$	lazy	projection-bound	success	success	success	success
$\mathtt{CRC}_R$	eager	projection-bound	success	success	success	$\Psi_{eval} = e_n$

Figure 5.17: Comparison of different resolution calculi

Index

Bibliography

[1] Habib Abdulrab, Pavel Goralcik, and G. S. Makanin. Towards parametrizing word equations. *ITA*, 35(4):331–350, 2001.

[2] Peter Andrews, Sunil Issar, Daniel Nesmith, and Frank Pfenning. The tps theorem proving system. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 760–761. Springer Berlin / Heidelberg, 1988. 10.1007/BFb0012885.

[3] Peter B. Andrews. Resolution in type theory. *J. Symb. Log.*, 36(3):414–432, 1971.

[4] Vincent Aravantinos, Ricardo Caferra, and Nicolas Peltier. Regstab: A sat solver for propositional schemata. In *IJCAR*, pages 309–315, 2010.

[5] Matthias Baaz and Alexander Leitsch. Cut-elimination and redundancy-elimination by resolution. *J. Symb. Comput.*, 29(2):149–177, 2000.

[6] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[7] Christoph Benzmüller. Extensional higher-order paramodulation and rue-resolution. In *CADE*, pages 399–413, 1999.

[8] Christoph Benzmüller and Michael Kohlhase. Extensional higher-order resolution. In *CADE*, pages 56–71, 1998.

[9] Christoph Benzmüller, Larry Paulson, Frank Theiss, and Arnaud Fietzke. The LEO-II project. In *Proceedings of the Fourteenth Workshop on Automated Reasoning, Bridging the Gap between Theory and Practice*. Imperial College, London, England, 2007.

[10] Christoph Benzmüller, Larry Paulson, Frank Theiss, and Arnaud Fietzke. Progress report on leo-ii - an automatic theorem prover for higher-order logic. In *In Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics - Emerging Trends*, 2007.

[11] Christoph Benzmüller, Lawrence Paulson, Frank Theiss, and Arnaud Fietzke. Leo-ii - a cooperative automatic theorem prover for classical higher-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated*

Reasoning, volume 5195 of *Lecture Notes in Computer Science*, pages 162–170. Springer Berlin / Heidelberg, 2008.

[12] Susanne Biundo, B. Hummel, Dieter Hutter, and Christoph Walther. The karlsruhe induction theorem proving system. In *Proceedings of the 8th International Conference on Automated Deduction*, pages 672–674, London, UK, UK, 1986. Springer-Verlag.

[13] G. Boole. *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*. George Boole's collected logical works. Walton and Maberly, 1854.

[14] C.B. Boyer and C.B. Boyer. *A History of Mathematics*. Wiley International Editions. John Wiley & Sons Canada, Limited, 1968.

[15] Alan Bundy, Frank van Harmelen, Christian Horn, and Alan Smaill. The oyster-clam system. In *Proceedings of the 10th International Conference on Automated Deduction*, pages 647–648, London, UK, UK, 1990. Springer-Verlag.

[16] Benzmüller C. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–335, 2002.

[17] Cristian S. Calude and Cristian S. Calude. *Randomness And Complexity, from Leibniz To Chaitin*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[18] Philippe Le Chenadec. On the logic of unification. *J. Symb. Comput.*, 8(1-2):141–199, July 1989.

[19] Alonzo Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.

[20] H. Comon. *Completion of Rewrite Systems with Membership Constraints*. Rapports de recherche. Université Paris-Sud, Centre d'Orsay, Laboratoire de recherche en Informatique, 1991.

[21] Hubert Comon. Completion of rewrite systems with membership constraints. part i: Deduction rules. *J. Symb. Comput.*, 25(4):397–419, 1998.

[22] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95 – 169, 1983.

[23] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[24] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476, August 1979.

[25] C. Dunchev, A. Leitsch, M. Rukhaia, and D. Weller. Ceres for propositional proof schemata. Technical report, Vienna University of Technology, 2012.

[26] Tsvetan Dunchev, Alexander Leitsch, Tomer Libal, Daniel Weller, and Bruno Woltzenlogel Paleo. System description: The proof transformation system ceres. In *IJCAR*, pages 427–433, 2010.

[27] Katrin Erk and Joachim Niehren. Dominance constraints in stratified context unification. *Inf. Process. Lett.*, 101(4):141–147, 2007.

[28] William M. Farmer. A unification algorithm for second-order monadic terms. *Annals of Pure and Applied Logic*, 39(2):131–174, 1988.

[29] William M. Farmer. Simple second-order languages for which unification is undecidable. *Theor. Comput. Sci.*, 87(1):25–41, 1991.

[30] Gottlob Frege. *Begriffsschrift: eine der arithmetische nachgebildete Formelsprache des reinen Denkens*. L. Nebert, Halle a/S, 1879/1997.

[31] G. Friedlein. *Procli Diadochi in primum Euclidis Elementorum librum commentarii*. Number v. 161 in Bibliotheca scriptorum Graecorum et Romanorum Teubneriana. in aedibus B. G. Teubneri, 1873.

[32] Kreisler G. Mathematical logic. In T.L. Saaty, editor, *Lectures on Modern Mathematics*, number v. 3 in Lectures on Modern Mathematics. John Wiley & Sons, 1965.

[33] D.M. Gabbay and J. Woods. *Handbook of the History of Logic: Greek, Indian, and Arabic logic*. Handbook of the History of Logic: Greek, Indian and Arabic Logic. Elsevier, 2004.

[34] Adria Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context unification with one context variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

[35] G. Gentzen. *Untersuchungen über das logische Schließen*. J. Springer, 1934.

[36] P. C. Gilmore. A proof method for quantification theory: its justification and realization. *IBM J. Res. Dev.*, 4(1):28–35, January 1960.

[37] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

[38] Jean Goubault-Larrecq. Ramified higher-order unification. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, pages 410–, Washington, DC, USA, 1997. IEEE Computer Society.

[39] W.E. Gould. *A Matching Procedure for [omega]-order Logic*. 1966.

[40] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[41] Allen Hazen. Predicative logics. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 1, pages 331–407. 1983.

[42] Leon Henkin. Completeness in the theory of types. *The Journal of Symbolic Logic*, 15(2):pp. 81–91, 1950.

[43] J. Herbrand. *Recherches sur la Theorie de la Demonstration, Travaux de la Societe des Sciences et de Lettres de Varsovie*. PhD thesis, 1930.

[44] D. Hilbert and P. Bernays. *Grundlagen Der Mathematik II*. Grundlehren der mathematischen Wissenschaften. Springer, 1970.

[45] J.I. Hmelevskii. *Equations in Free Semigroups*. Proceedings of the Steklov Institute of Mathematics. American Mathematical Society, 1976.

[46] H.F.J. Horstmanshoff, M. Stol, and C. Tilburg. *Magic And Rationality In Ancient Near Eastern And Graeco-roman Medicine*. Studies in Ancient Medicine. Brill, 2004.

[47] G. Huet. *Résolution d'équations dans des langages d'ordre 1,2,..,ω*. PhD thesis, Université Paris VII, 1976.

[48] Gérard P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic*. PhD thesis, Case Western Reserve University, 1972.

[49] Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.

[50] Matt Kaufmann and J. Strother Moore. How can i do that with acl2? recent enhancements to acl2. In *ACL2*, pages 46–60, 2011.

[51] W. Kneale and M. Kneale. *The Development of Logic*. Oxford University Press, USA, 1985.

[52] A. Koscielski and L. Pacholski. Complexity of unification in free groups and free semigroups. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, SFCS '90, pages 824–829 vol.2, Washington, DC, USA, 1990. IEEE Computer Society.

[53] Philippe Le Chenadec. The finite automaton of an elementary cyclic set. Technical Report RR-0824, INRIA, April 1988.

[54] G.W. Leibniz, K. Gerhardt, and G.H. Pertz. *Leibnizens mathematische Schriften: Mathematik*. [Leibnizens gesammelte Werke aus den Handschriften der Königlichen Bibliothek zu Hannover herausgewgeben von Georg Heinrich Pertz. Dritte Folge.]. A. Asher & Company, 1858.

[55] Jordi Levy. Linear second-order unification. In *RTA*, pages 332–346, 1996.

[56] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.

[57] Jordi Levy and Mateu Villaret. Context unification and traversal equations. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, RTA '01, pages 169–184, London, UK, UK, 2001. Springer-Verlag.

[58] Tomer Libal. Cut elimination in inductive proofs of weakly quantified theorems. Master's thesis, Vienna University of Technology, 2008.

[59] G. S. Makanin. On the decidability of the theory of free groups (in russian). In *FCT*, pages 279–284, 1985.

[60] William Mccune. Solution of the robbins problem. *Journal of Automated Reasoning*, 19:263–276, 1997.

[61] Dale Miller. Unification of simply typed lambda-terms as logic programming. In *In Eighth International Logic Programming Conference*, pages 255–269. MIT Press, 1991.

[62] Dale A. Miller. Proofs in higher-order logic. 1983.

[63] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *CADE*, pages 34–48, 1997.

[64] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In *ACL*, pages 410–417, 1997.

[65] Joachim Niehren, Sophie Tison, and Ralf Treinen. On rewrite constraints and context unification. *Inf. Process. Lett.*, 74(1-2):35–40, 2000.

[66] Lawrence Paulson. Isabelle: The next seven hundred theorem provers. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 772–773. Springer Berlin / Heidelberg, 1988. 10.1007/BFb0012891.

[67] Christian Prehofer. Decidable higher-order unification problems. In *Proceedings of the 12th International Conference on Automated Deduction*, CADE-12, pages 635–649, London, UK, UK, 1994. Springer-Verlag.

[68] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[69] Manfred Schmidt-Schauß. A decision algorithm for distributive unification. *Theoretical Computer Science*, 208:111–148, 1998.

[70] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Log. Comput.*, 12(6):929–953, 2002.

[71] Manfred Schmidt-Schauß. Decidability of bounded second order unification. *Inf. Comput.*, 188:143–178, January 2004.

[72] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, pages 61–75, 1998.

[73] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symb. Comput.*, 33(1):77–122, 2002.

[74] Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher-order unification. *J. Symb. Comput.*, 40(2):905–954, August 2005.

[75] S.G. Simpson. *Subsystems of Second Order Arithmetic*. Perspectives in Logic. Cambridge University Press, 2010.

[76] Wayne Snyder and Jean H. Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symb. Comput.*, 8(1/2):101–140, 1989.

[77] Wayne S. Snyder. *Complete sets of transformations for general unification*. PhD thesis, Philadelphia, PA, USA, 1988. AAI8824793.

[78] Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The tptp problem library. In Alan Bundy, editor, *Automated Deduction — CADE-12*, volume 814 of *Lecture Notes in Computer Science*, pages 252–266. Springer Berlin / Heidelberg, 1994.

[79] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[80] Marek Zaionc. The regular expression descriptions of unifier sets in the typed lambda calculus. In *Fundamenta Informaticae X*, pages 309–322. North-Holland, 1987.