Variants of unification considering compression and context variables

Tesi doctoral presentada al Departament de Llenguatges i Sistemes Informàtics (LSI)
de la Universitat Politècnica de Catalunya (UPC)

per a optar al grau de
Doctor en informàtica

per

Adrià Gascón Caro

sota la direcció del doctor

Guillem Godoy Balil

Barcelona, Abril 2014

Abstract

Term unification is a basic operation in several areas of computer science, specially in those related to logic. Generally speaking, it consists on solving equations over expressions called terms. Depending on the kind of variables allowed to occur in the terms and under which conditions two terms are considered to be equal, several frameworks of unification such as *first-order unification*, *higher-order unification*, *syntactic unification*, and *unification modulo theories* can be distinguished. Moreover, other variants of term unification arise when we consider nontrivial representations for terms. In this thesis we study variants of the classic first-order syntactic term unification problem resulting from the introduction of context variables, i.e. variables standing for contexts, and/or assuming that the input is given in some kind of compressed representation. We focus on two of of such representations: *Directed Acyclic Graphs* (DAGs) and *Singleton Tree Grammars* (STGs). Similarly as DAGs allow compression by exploiting the reusement of repeated instances of a subterm in a term, STGs are a grammar-based compression mechanism based on the reusement of repeated (multi)contexts. An interesting property of the STG formalism is that many operations on terms can be efficiently performed directly in their compressed representation.

In the first part of this thesis, we study the classical first-order syntactic unification and matching problems under the assumption that the input terms are represented using STGs. We present polynomial time algorithms for these problems, as well as proposed implementations and experimental results. These results are used later in this thesis in the analysis of compressed context unification and matching.

In the second part of the thesis, we focus on variants of *context unification.* We show that context matching is NP-Complete even when the input terms are represented with STGs. We also study the particular case of context unification where only one context variable is allowed in the input equations, called *one context unification.* We show that one context unification can be solved in nondeterministic polynomial time, not only when a explicit representation for terms is used, but also when the input is represented by STGs or DAGs. We also present a partial result in the task of finding a polynomial time algorithm for this problem. Finally, we show that the restriction of context matching where the number of context variables is bounded by a constant k can be solved in polynomial time even when DAGs are used for term representation.

Acknowledgements [1]

En primer lloc, vull donar les gràcies al meu tutor, en Guillem Godoy, per introduir-me en el món de la recerca amb honestedat, generositat, paciència i bon humor. Li estic especialment agrait per les fantàstiques hores que he passat al seu costat plantant-li cara a tot de problemes (i perdent ànima quan calia). Trobo molt a faltar aquelles sessions. Li agraeixo molt tot el que he après amb ell. Des de l'assignatura de TC fins a l'últim ”acertijo”.

A en Jorge i el seu ”sign & forget”, el Nikita, el Dani, en Josep Lluís, l'Alessandra, el Miquel, en Sergi, en Javier, en Ramon, l'Eva, l'Albert i en Marc els agraeixo haver estat tan bons companys d'aventura. Gràcies per tants senyals de suport, respecte, camaraderia i bon rotllo.

Vull agrair al Lander i al Carles la qualitat del temps que hem passat treballant junts. L'experiència d'escriure amb el Carles es mereix una menció especial, aquelles sessions també les trobo a faltar moltíssim.

I also want to mention Sebastian Maneth, thanks for making me feel so welcomed in Sydney and always being so honest, accesible, and easygoing. Thanks also to Ashish Tiwari for giving me the possibility of visiting at SRI. His way of approaching research is a great inspiration to me and I keep on learning from him. Moltes gràcies també als professors que van saber fer de la universitat una experiència estimulant, especialment a la Carme Àlvarez, l'Albert Atserias i l'Albert Oliveras.

Pel que fa al veritable inici d'aquesta aventura, vull agrair infinits moments de qualitat durant i després de la carrera a l'Èric (Texas inclós), el Miguel i en Guillem. També a l'Àxel, el Marc, el Rafa i el Nin, per sempre ser-hi, més enllà de problemes, teoremes, deadlines, demostracions i programes.

Als meus pares, moltes gràcies per tot el suport, l'exigència i confiança, els ànims i la disponibilitat que m'heu ofert al llarg dels anys. Gran part del necessari per a encarar aquest repte us ho dec a vosaltres. A la Clara, gràcies per sempre ser-hi (presentació de màster inclosa), i per saber fer-me sentir tan important i orgullós.

Finalment, i molt especialment, vull agrair-li a la Lídia ser font de recolzament incondicional i companya d'aventures tot aquest temps. El seu respecte, generositat i estima han fet més fàcil cada pas d'aquesta tasca.

[1] Part of this work was supported by the FORMALISM project (TIN2007-66523), funded by the Spanish government, and an FPU grant by the Spanish Ministry of Education.

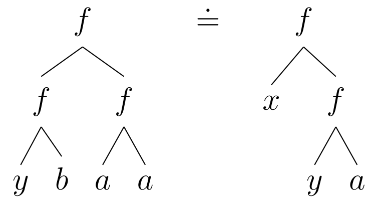
Contents

Chapter 1

Introduction

Term unification, the central concept of this thesis, was firstly introduced as such in the work by J.A. Robinson, which settled the foundations of automated theorem proving and logic programming. More concretely, Robinson presented in [Rob65] a procedure to determine the validity of a first-order sentence that has term unification as its main subprocess. In the context of Robinson's work, unification corresponds to the task of combining two premises with the aim of constructing a deductive logical argument. Later, term unification was also used by Knuth and Bendix as a key component of their critical pairs method to determine local confluence of term rewrite systems. For a general survey in unification theory, we refer the reader to Chapter 8 of the Handbook of Automated Reasoning [BS01].

Generally speaking, unifying two terms s and t corresponds to solving an equation $s \doteq t$, which consists on finding a substitution σ for variables occurring in s and t such that $\sigma(s) = \sigma(t)$ holds. Hence, different frameworks of unification are defined by specifying a range for the variables, the kind of expressions s and t, and their semantics, as well as the semantics of $=$. In Robinson's unification, the well-known *syntactic first-order unification*, expressions s and t are terms with leaf variables standing for terms (first-order variables), all function symbols are noninterpreted, and $=$ is interpreted as syntactic equality. For example, consider the following instance of the syntactic first-order unification problem.



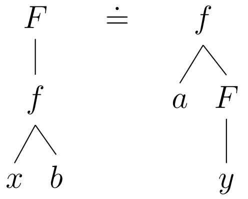
Note that the substitution $\sigma = \{x \mapsto f(a, b), y \mapsto a\}$ satisfies the conditions stated above.

The *first-order term matching* problem is a particular case of first-order term unification. It is characterized by the condition that one of the sides of the equation $s \doteq t$, say t, does not contain variables. Both first-order term unification and matching are common problems in many areas of computer science; specially those related to logic, such as functional and logic programming, and automated deduction. These two problems were deeply investigated in the last century (see [BS01]). Among other results, linear time algorithms were discovered [MM82, PW78]. However, first-order unification is not always expressive enough to deal with the problems arising in the areas mentioned above and therefore several variants have been considered. A remarkable extension is *unification modulo theories*. In this notion of unification, equality between terms is interpreted under equational theories such as associativity, commutativity, and distributivity, among others [BS01].

Another widely considered notion of unification allows variables of arity one standing for contexts, in addition to the leaf variables of first-order unification. This extension is called *context unification*. Context unification is in fact a particular case of *second-order unification*, which is itself a particular case of *higher-order unification*. Intuitively, in second-order unification, not only first-order variables may appear in the input, but also variables of arity greater than 0 standing for transformations on terms.

While second-order unification is known to be undecidable [Gol81], decidability of context unification is still open, although a proof that the problem is in fact in PSPACE has been recently proposed. However, some interesting results have been found for some particular cases, with applications in computational linguistics [LNV05, EN07, LSSV11]. Furthermore, Schmidt-Schauss and Schulz [SSS02] showed decidability of the case where at most two distinct context variables appear in the equations, but their algorithm is rather involved and no complexity bounds are known for it. Another recent result is [LSSV11], where it was shown that stratified context unification is NP-complete. Besides the higher-order unification perspective, context unification can also be seen as an extension of the problem of solving equations on words.

In this thesis we consider another particular case of context unification: *one context unification*. One of the motivations for the study of this problem is its close relation to interprocedural program analysis [GT07], whose goal is to compute all simple invariants of imperative procedural programs. In one context unification, only one context variable, possibly with many occurrences, may appear in the input terms. Consider the following example:



where x, y are first-order variables and F is a context variable that can be replaced by any context. One of the possible solutions of this instance is the substitution $\{F \mapsto f(a, \bullet), x \mapsto a, y \mapsto b\}$. Note that when we instantiate F by $f(a, \bullet)$ in the equation, replacing the occurrence of $\bullet$ by the argument of F in each of its occurrences, we get $f(a, f(x, b)) \doteq f(a, f(a, y))$, and thus both sides of the equations become equal after applying $\{x \mapsto a, y \mapsto b\}$. As we will see in Chapter 5 of this thesis, one context unification can be solved in nondeterministic polynomial time [GGSST10]. However, it is not known neither whether it is NP-hard nor whether a polynomial time algorithm exists for this problem. In Chapter 8 we present an intermediate result in the effort of finding a polynomial time algorithm for this problem.

Another particular case of context unification is *context matching*. The input of context matching is an equation $s \doteq t$ such that s may contain context variables and first-order variables, and t does not contain variables of any kind. Although this problem is known to be NP-complete, there are several subcases that can be solved efficiently [GGSS11, SSS04]. In Chapter 7 we consider k-context matching, the particular case of context matching where the number of different context variables that may occur in the input is bounded by a constant number k. As one of the contributions of this thesis, originally presented in [GGSS08], in Chapter 7 we present an algorithm for k-context matching that runs in polynomial time even when DAGs are used for term representation.

Interesting applications of one context unification and matching arise in the search/extraction of information from tree data structures. For example, a simple matching equation of the form $F(s) \doteq t$, where F is the context variable, t is ground, and s may contain first-order variables but does not contain occurrences of F, corresponds to searching instances of s within t. Another example of the expressivity of context matching is a conjuntive search of the form $F_1(s_1) \doteq t \wedge \cdots \wedge F_n(s_n) \doteq t$, where the F_is are pairwise different and do not occur elsewhere. These equations correspond to searching for a subterm u_i of t that can be matched by s_i, for every $i \in \{1, \ldots, n\}$; with the additional constraint that variables within the s_is must have a common instance in t (see [GKS06] for the analysis of conjunctive query mechanisms over trees). More generally, multiple occurrences of the same context variable in the term s of a context equation $s \doteq t$, correspond to searching for instances of subterms of t that differ at, at most, one position. This has applications in computational linguistics [NPR97]. It is also easy to encode questions that ask for subtrees that are equal up to several positions.

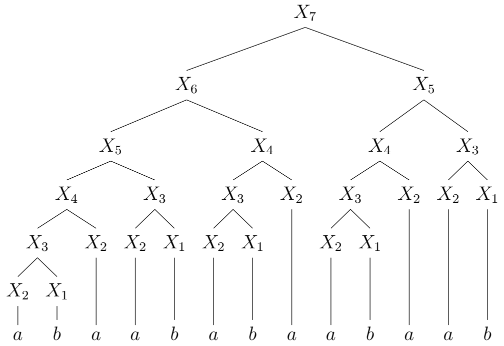
Finally, another application domain of variants of term unification is logic programming languages for XML. Examples of such languages are Xcerpt [BBSW03], which uses a form of asymmetric unification called *simulation*, and Xcentric [CF07], which uses the variant of term unification studied in [Kut02].

Term representation

Many of the applications dealing with the problems we have introduced and their variants require some kind of internal succinct representation for terms. Sometimes, such representation is useful to achieve efficiency. This is the case of DAGs when solving first-order unification (see [BS01]). In other cases, the succinctness of the representation is required in order to guarantee computability in an environment with a limited amount of resources. This is the case, for example, of the manipulation of XML data, which represents hierarchical data in the form of unranked trees. Therefore, *compressed* in-memory representations have been developed, such as succinct trees (see, e.g., [SN10]), or grammar-compressed trees [BLM08]. Finally, having succinct term representations and efficient algorithms that operate on such representations is useful also from a theoretical perspective [LSSV08, LSSV11, CGG12]. For example, the main technical approach used to show membership of stratified context unification in NP is based on the use of a grammar-based representation for terms to avoid a space explosion due to variable instantiations.

However, it is sometimes the case that the advantages of space efficient encodings come at a price. In this thesis, we reconsider complexity issues for some of the unification problems mentioned so far by assuming that the input terms are given in some compressed representation. For instance, it is well-known that first-order unification may require exponential space with a plain term representation whereas only polynomial space is required when DAGs are used for representing terms. Similarly, if terms are large but have lots of common subterms, like $t_1 = f(a, b)$, $t_2 = f(t_1, t_1), \ldots, t_n = f(t_{n-1}, t_{n-1})$, then the context matching equation $F(a) = t_n$ requires exponential space using the plain term representation to represent t_n, whereas a DAG representation requires linear space. This motivates the investigation of the context matching problem with compression techniques like DAGs that we present in Chapter 7 of this thesis.

Besides the DAG representation, more general grammar-based compression mechanisms for terms have recently drawn considerable attention in research. Grammar-based compression was invented in the 1990s for string representation (see [Ryt04] for a survey). The idea is to construct a context-free grammar that generates only the string s that we want to represent. Compression is obtained by reusing nonterminals that generate repeated substrings of s. For example, assume that we want to represent the string $s = abaababaabaab$. The following picture shows the derivation tree to obtain s with a grammar with nonterminals $X_1, \ldots, X_7$. Note that s, which has length 13, can be represented by a grammar with 7 rules.



Grammar-based compression allows to compress a string exponentially. For example, the string a^{2^n} can be easy generated by grammar with n rules. Unfortunately, finding a minimal grammar representation of a string is NP-complete [CLL$^+$05], but several well-behaved approximation algorithms exist [CLL$^+$02]. One of the properties that makes grammar-based compression useful is that efficient algorithms that run directly on the compressed representation of their input string(s), and thus avoid the "decompress and check" approach, have been developed. For example, we have algorithms for equivalence checking and pattern matching [Pla94, Lif07, SSS12, Jez12], checking membership for various classes of languages [LM11, Loh10, LM11], and efficient indexing techniques have also been developed [CN12]. It is important to remark that the technique introduced in [Jez12] led to important advances in compression and solvability of word equations. The reader is referred to [Loh12] for a survey in algorithms on compressed strings.

Later, grammar-based compression was extended to terms/trees [BLM08, CDG$^+$07] with applications in XML tree structure compression [BLM08] and XPATH [LM06]. This extension led to *Singleton Tree Grammars (STGs)*, a grammar-based formalism more general than DAGs that has recently drawn considerable attention in research. With an STG we can succinctly represent terms that are exponentially big in size and height. Similarly to the string case, algorithms running on the compressed representation of the corresponding term(s), have been found, and applications have been developed.

Some examples are linear subpattern matching [SS13], i.e. finding instances of a linear terms s within a ground term t, first-order and unification and matching [GGSS11, GMR11], one context unification [CGG12], equivalence checking [BLM08], congruence closure [SSSA11], and membership in several classes of languages represented by tree automata [LM06, LMSS12]. Moreover, STG compressors have already been developed and proven useful in practice for XML representation and processing [BLM08, LMM13], and termination analysis of Term Rewrite Systems [BLNW13].

Contributions and thesis organization

Although we already covered the results presented in this thesis, let us give a more detailed overview of the contents of each chapter.

- In Chapter 2, we introduce basic concepts and notations used in this thesis, along with explanations and examples, with the aim of helping the reader to get familiar with the intuitions behind the ideas presented in subsequent chapters.

- In Chapter 3 we formally define Directed Acyclic Graphs and Singleton Tree Grammars from the perspective of term representation. Also in this chapter, we present constructions on STG-compressed terms that will be crucial in the results presented in subsequent chapters.

- In Chapter 4 we study the first-order unification and matching problems when the input is compressed using Singleton Tree Grammars. We present the polynomial time algorithms for these problems originally presented in [GGSS11], whose running time improves previous results [GGSS08, GGSS09]. Furthermore, in Section 4.3 of the same Chapter we present an experimental comparison of three different implementations of these algorithms originally published in [GMR11].

- In Chapter 5 we show that one context unification can be solved in nondeterministic polynomial time [GGSST10]. We also present two interesting particular cases that can be solved in polynomial time.

- In Chapter 6 we extend the result of the previous chapter to the compressed setting, that is, we prove that one context unification with STG-compressed terms is in NP, as originally presented in [CGG12].

- In Chapter 7 we prove the two results for the context matching problem originally presented in [GGSS08] and [GGSS11]. We first show, in Section 7.1, that the k-context matching problem can be solved in polynomial time even when DAGs are used for term representation. Then, in Section 7.2 of the same chapter we prove that context matching is NP-Complete also in the case when STGs are used for term representation.

- In Chapter 8, we describe a recent partial result in the effort of finding a polynomial time algorithm for one context unification.

- Finally, in Chapter 9, we present some possible lines of future research that arise from the work presented in this thesis.

The following table summarizes the results presented in this thesis, together with the previously known results and open questions related to first-order unification and matching and the variants of context unification covered in this work. Let us remark that a solution positive solution to he decidability of context unification hs been recently proposed.

Problem		**Term representation formalism**		
		Explicit	**DAG**	**STG**
Context unification		?	?	?
	One context unification	NP [GGSST10]	NP [CGG12]	NP [CGG12]
	Context matching	NP-Complete	NP-Complete	NP-Complete [GGSS11]
	k-context matching	Ptime	Ptime [GGSS11, GGSS08]	?
First-order unification		Ptime	Ptime	Ptime [GGSS11, GGSS09]
	First-order matching	Ptime	Ptime	Ptime [GGSS11, GGSS08]

The work presented in this thesis led to the following publications.

[CGG12] Carles Creus, Adrià Gascón, and Guillem Godoy. One-context Unification with STG-Compressed Terms is in NP. In *23rd International Conference on Rewriting Techniques and Applications, RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, pages 149–164, 2012.

[GGSS08] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Context Matching for Compressed Terms. In *Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 93–102, 2008.

[GGSS09] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification with Singleton Tree Grammars. In *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, pages 365–379, 2009.

[GGSS11] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification and Matching on Compressed Terms. *ACM Trans. Comput. Log.*, 12(4):26, 2011.

[GGSST10] Adrià Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context Unification with One Context Variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

[GMR11] Adrià Gascón, Sebastian Maneth, and Lander Ramos. First-order Unification on Compressed Terms. In *22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, pages 51–60, 2011.

Chapter 2

Basic notions and notation

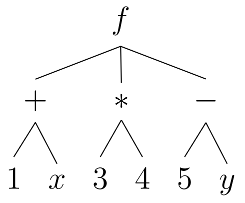
In this chapter the necessary concepts and definitions in the scope of this work are introduced. Most of the basic definitions and explanations regarding terms and term unification were borrowed from [CDG+07] and [BS01].

sectionTerms Generally speaking, terms allow the representation of data with substructure. A term is either:

- A constant symbol
- A variable
- A compound term

A compound term consists of a function symbol applied on a sequence of one or more terms called arguments. It sometimes helps to think of a compound term as a tree structure.

Example 2.0.1 *The formula* $f(1 + x, 3 * 4, 5 - y)$ *could be depicted as the structure:*



where $f, +, *, -$ *are function symbols* $1, 3, 4, 5$ *are constants, and* x, y *represent variables.*

The number of arguments taken by a function symbol is called its *arity*. In the example above, $+, -$ and $*$ have an arity of 2 and f has arity 3. The rest of symbols have arity 0.

Once introduced the intuitive idea on what the terms are, we can define them, as well as the notation used in this thesis, in a more precise way.

Terms and Positions

A *ranked alphabet* is a set $\mathcal{F}$ together with a function $\mathtt{ar} : \mathcal{F} \to \mathbb{N}$. Members of $\mathcal{F}$ are called function symbols, and $\mathtt{ar}(f)$ is called the *arity* of the function symbol f. Given a set of function symbols $\mathcal{F} = \{f_1, \ldots, f_n\}$, we may denote a ranked alphabet more explicitly as $\{f_1 : \mathtt{ar}(f_1), \ldots, f_n : \mathtt{ar}(f_n)\}$. Function symbols of arity 0 are called *constants*. Let $\mathcal{V}$ be a set disjoint from $\mathcal{F}$ whose elements are called *variables*. We assume the function $\mathtt{ar}$ to be also defined for variables, i.e. $\mathtt{ar} : (\mathcal{F} \cup \mathcal{V}) \to \mathbb{N}$, but with $\mathtt{ar}(V) \in \{0, 1\}$ for variables $V \in \mathcal{V}$. Variables with arity 0, denoted x, y, z with possible indexes, are called *first-order variables*, and variables with arity 1, denoted F with possible subscripts, are called *context variables*. We use f, g, h for denoting an element in $\mathcal{F}$, and α for denoting an element in $\mathcal{F} \cup \mathcal{V}$.

The set $\mathcal{T}(\mathcal{F} \cup \mathcal{V})$ of terms over $\mathcal{F}$ and $\mathcal{V}$, also denoted $\mathcal{T}(\mathcal{F}, \mathcal{V})$, is defined to be the smallest set having the property that $\alpha(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F} \cup \mathcal{V})$ whenever $\alpha \in (\mathcal{F} \cup \mathcal{V})$, $m = \mathtt{ar}(\alpha)$ and $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{F} \cup \mathcal{V})$. The set $\mathcal{T}(\mathcal{F})$ is called the set of ground terms over $\mathcal{F}$, that is, the subset of terms of $\mathcal{T}(\mathcal{F} \cup \mathcal{V})$ with no occurrences of variables. We denote by s, t, with possible indexes, terms in $\mathcal{T}(\mathcal{F} \cup \mathcal{V})$.

Example 2.0.2 *Let $\mathcal{F} = \{f : 2, h : 1, a : 0, b : 0\}$ be a ranked alphabet with symbols $f, h, a,$ and b with arity $2, 1, 0,$ and 0, respectively. Let $\mathcal{V} = \{x : 0, y : 0, F : 1\}$ be a set of first-order and context variables. Then, $f(a, a)$ and $f(h(a), b)$ are ground terms in $\mathcal{T}(\mathcal{F})$, and $f(x, y)$ and $h(F(f(a, x)))$ are terms in $\mathcal{T}(\mathcal{F} \cup \mathcal{V})$.*

Positions of a term t, denoted p, q with possible subindexes, are sequences of natural numbers that are used to identify subterms of t. The set $\mathsf{Pos}(t)$ of *positions* of a term t is defined by $\mathsf{Pos}(t) = \{\lambda\}$ if t is a constant or a first-order variable variable, and $\mathsf{Pos}(t) = \{\lambda\} \cup \{1 \cdot p \mid p \in \mathsf{Pos}(t_1)\} \cup \ldots \cup \{m \cdot p \mid p \in \mathsf{Pos}(t_m)\}$ if $t = \alpha(t_1, \ldots, t_m)$, where λ denotes the empty sequence, also called the *root position*, and $p.q$, or simply pq, denotes the concatenation of p and q. If t is a term and p a position, then $t|_p$ denotes the *subterm* of t at position p. More formally defined, $t|_\lambda = t$ and $\alpha(t_1, \ldots, t_m)|_{i.p} = t_i|_p$. We use $\prec$ to denote the subterm relation on terms.

The *length* of a position p is denoted by $|p|$. Note that $|\lambda| = 0$ and $|i.p| = 1 + |p|$ hold. A position p_1 is a *prefix* of a position p, denoted $p_1 \leq p$, if there is a position p_2 such that $p_1.p_2 = p$ holds. Also, p_1 is a *proper prefix* of p, denoted $p_1 < p$, if $p_1 \leq p$ and $p_1 \neq p$ hold. A *suffix* and a *proper suffix* of a position p is defined analogously. Two positions p, p' are called *parallel* or *disjoint*, denoted by $p \| p'$, if $p \not\leq p'$ and $p' \not\leq p$ hold.

Functions on Terms

The *size* of a term t, denoted by $|t|$ and the *height* of t, denoted by $\texttt{height}(t)$ are recursively defined as:

- $\texttt{height}(t) = 0$ and $|t| = 1$, if t is a constant or a first-order variable, and
- $\texttt{height}(t) = 1 + \max_{1 \leq i \leq n}(\texttt{height}(t_i))$ and $|t| = 1 + \sum_{i=0}^{n} |t_i|$, if t is of the form $t = \alpha(t_1, \ldots, t_m)$.

Therefore, the height of a term t refers to the number of edges in the deepest branch of the tree representing t, and its size refers to its total number of nodes.

We denote by $\texttt{root}(t)$ the symbol at the root position of a term $t = \alpha(t_1, \ldots, t_m)$, which is defined as $\texttt{root}(\alpha(t_1, \ldots, t_m)) = \alpha$, where $m = \texttt{ar}(\alpha)$. Given a term t, a position $p \in \mathsf{Pos}(t)$ and a symbol α, we say that p is *labeled* by α in t if $\texttt{root}(t|_p) = \alpha$, i.e $t|_p$ is of the form $\alpha(t_1, \ldots, t_m)$.

The *replacement* of a term t at a position $p \in \mathsf{Pos}(t)$ by a term s, denoted $t[s]_p$, is defined recursively as $t[s]_p = s$, if $p = \lambda$, and $\alpha(t_1, \ldots, t_m)[s]_{i.p} = \alpha(t_1, \ldots, t_i[s]_p, \ldots, t_m)$, otherwise.

Example 2.0.3 *Let t be the term $f(g(a, g(a, x)), f(a, b, a), a)$. Note that $\texttt{height}(t) = 3$, $\texttt{root}(t) = f$, and $|t| = 11$. Moreover, the term $g(a, x)$ occurs at position 1.2 and thus $t|_{1.2} = g(a, x)$. Finally, note that $t[g(b, b)]_{1.1} = f(g(g(b, b), g(a, x)), f(a, b, a), a)$.*

Preorder Traversal of a Term

We denote by $\mathrm{Pre}(t)$ the preorder traversal (as a word) of a term t. It is recursively defined as $\mathrm{Pre}(t) = t$, if t is a constant or a first-order variable, and $\mathrm{Pre}(t) = \alpha.\mathrm{Pre}(t_1). \ldots .\mathrm{Pre}(t_m)$, if $t = f(t_1, \ldots, t_m)$ with $m > 0$. Two different trees may have the same preorder traversal, but when they represent terms over a fixed alphabet where the arity of every function symbol is fixed, the preorder traversal is unique for every term. Given a term t, there is a natural bijective mapping between the indexes $\{1, \ldots, |\mathrm{Pre}(t)|\}$ of $\mathrm{Pre}(t)$ and the positions $\mathsf{Pos}(t)$ of t, which associates every position $p \in \mathsf{Pos}(t)$ to the index $i \in \{1, \ldots, |\mathrm{Pre}(t)|\}$ you find at $\texttt{root}(t|p)$ while traversing the tree in preorder. We can recursively define the two mappings $\texttt{pIndex}(t, p) \rightarrow \{1, \ldots, |\texttt{pre}(t)|\}$ and $\texttt{iPos}(t, i) \rightarrow \texttt{Pos}(t)$ as follows.

- $\texttt{pIndex}(t, p) = 1$, if $p = \lambda$, and

- $\texttt{pIndex}(\alpha(t_1,\ldots,t_m), i.p) = (1+|t_1|+\ldots+|t_{i-1}|)+\texttt{pIndex}(t_i,p)$, otherwise.

- $\texttt{iPos}(t,i)=\lambda$, if $i=1$, and

- $\texttt{iPos}(\alpha(t_1,\ldots,t_m), 1+|t_1|+\ldots+|t_{j-1}|+k) = i.\texttt{iPos}(t_i,k)$ for $1\leq k\leq |t_j|$.

2.1 Contexts and Tree Patterns

Intuitively, contexts are terms with a single occurrence of a hole, denoted $\bullet$, into which terms (or other contexts) may be inserted. We can provide a formal definition by considering a context to be a term in an extended alphabet that includes an extra constant symbol $\bullet$. Hence, the smallest context, also called *empty context*, contains just the hole and has size 1.

The set of contexts over a ranked alphabet $\mathcal{F}$ and a set of variables $\mathcal{V}$ is denoted $\mathcal{C}(\mathcal{F},\mathcal{V})$. We use c, d, with possible subindexes, to denote a context in $\mathcal{C}(\mathcal{F},\mathcal{V})$. The operations on terms defined above are extended to contexts in the natural way. The *hole position* of a context c, denoted $\texttt{hp}(c)$, is the position in $\mathsf{Pos}(c)$ labeled by $\bullet$.

Operations on Contexts

Given a context c and a term t, we define the term $c[t]$ as the replacement $c[t]_{\texttt{hp}(c)}$. Similarly, given another context d, the *concatenation* cd is the context $c[d]_{\texttt{hp}(c)}$. Note that $\texttt{hp}(cc') = \texttt{hp}(c).\texttt{hp}(c')$ holds. In other words, if c and d are contexts and t is a term, cd and $c[t]$ represent the term (context) that is like c except that the occurrence of $\bullet$ is replaced by t (d).

The *exponentiation* of a context c to a natural number e, denoted c^e, is the context recursively defined as

- $c^e = cc^{e-|\texttt{hp}(c)|}$, if $e > |\texttt{hp}(c)| > 0$, and

- $c^e = c[\bullet]_p$, if $|\texttt{hp}(c)| \geq e$, where $p \leq \texttt{hp}(c)$ and $|p| = e$.

Note that $|\texttt{hp}(c^e)| = e$ holds for any context c with $|\texttt{hp}(c)| > 0$ and natural number e.

We say that c is a *subcontext* of a term t if $c = t|_{p_1}[\bullet]_{p_2}$, for $p_1.p_2 \in \mathsf{Pos}(t)$. Finally, if $c_1 = c_2c_3$ for contexts c_1, c_2, c_3, then c_2 is called to be a *prefix* of c_1, and c_3 is a *suffix* of c_1.

Example 2.1.1 *Let c be the context $g(f(g(a,\bullet),a),F(b))$, d be the context $h(\bullet)$ and t be the term $g(c,x)$. Then* $\texttt{hp}(c) = 1.1.2$*,* $|c| = 8$*,* $c[t] = g(f(g(a,g(c,x)),a),F(b))$*,* $|c[t]| = 10$*,* $cd = g(f(g(a,h(\bullet)),a),F(b))$*, and* $\texttt{hp}(cd) = hp(c).\texttt{hp}(d) = 1.1.2.1$*.*

2.2 Substitutions

Fixed a ranked alphabet $\mathcal{F}$ and a set of variables $\mathcal{V}$, a *substitution*, denoted by σ, θ, is a total function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{C}(\mathcal{F},\mathcal{V})$ such that $\sigma(\alpha) \in \mathcal{T}(\mathcal{F},\mathcal{V})$ if α is a first-order variable and $\sigma(\alpha) \in \mathcal{C}(\mathcal{F},\mathcal{V})$ if α is a context variable. The concept of domain for substitutions differs from the usual one for arbitrary functions. We consider substitutions to be total functions by assuming that $\sigma(\alpha) = \alpha$ if $\sigma(\alpha)$ is not defined. Thus, we define the *domain* of a substitution σ as $\mathsf{Dom}(\sigma) = \{\alpha \mid \sigma(\alpha) \neq \alpha\}$. For this reason, when defining a particular substitution σ we do not make explicit $\sigma(\alpha)$ for variables $\alpha \notin \mathsf{Dom}(\sigma)$. We also define the variables occurring in a substitution σ as $\mathsf{Vars}(\sigma) = \bigcup_{\alpha \in \mathsf{Dom}(\sigma)} \{\alpha\} \cup \mathsf{Vars}(\sigma(\alpha))$. Moreover, substitutions are extended to be mappings from terms to terms, i.e. $\sigma : \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F},\mathcal{V})$, as follows: $\sigma(g(t_1,\ldots,t_n)) = g(\sigma(t_1),\ldots,\sigma(t_n))$, where g is a function symbol, and $\sigma(F(t)) = \sigma(F)[\sigma(t)]$, where $\mathcal{F}$ is a context variable in $\mathcal{V}$. Similarly, substitutions are also extended to be mappings from contexts to contexts, i.e. $\sigma : \mathcal{C}(\mathcal{F},\mathcal{V}) \to \mathcal{C}(\mathcal{F},\mathcal{V})$. The *composition* of two substitutions σ and θ, denoted $\theta \circ \sigma$, is defined as $\{\alpha \mapsto \theta(\sigma(\alpha)) \mid \alpha \in \mathsf{Dom}(\sigma) \cup \mathsf{Dom}(\theta)\}$. Substitutions mapping from $\mathcal{V}$ into $\mathcal{T}(\mathcal{F})$ are called *ground*. Finally, given substitutions σ, θ, $\sigma = \theta$ if $\forall x \in \mathcal{X} : \sigma(x) = \theta(x)$. Moreover, σ is *more general* than θ, denoted $\sigma \leq \theta$, if there exists η such that $\sigma = \theta \circ \eta$.

Example 2.2.1 *Let t be the term $f(x_1,F(x_2))$, where x_1,x_2 are first-order variables and F is a context variable, and let σ be the substitution $\{x_1 \mapsto g(a,b), F \mapsto g(a,\bullet)\}$. Then $\sigma(t) = f(g(a,b),g(a,x_2))$ and the domain of σ is $\{x_1,F\}$.*

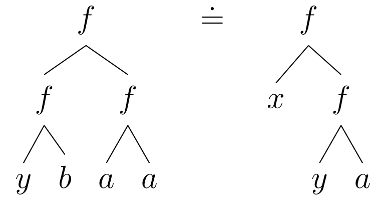
2.3 Unification and Matching

Very generally speaking, unification tries to make two symbolic expressions equal by replacing certain subexpresions (variables) by other expressions. Hence, this task consists on solving equations $s \doteq t$ by finding a substitution σ for variables occurring in both expressions s and t such that $\sigma(s) = \sigma(t)$.

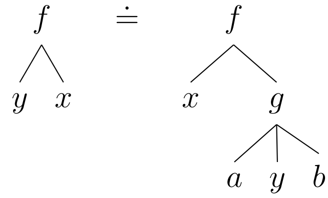
2.3.1 First-Order Unification and Matching

The classical *first-order term unification problem* seeks to find solutions for term equations built over uninterpreted function symbols and *first-order* variables where $=$ is interpreted as syntactic equality.

Example 2.3.1 *Consider the following instance of the first-order unification problem.*



It has the substitution $\sigma = \{x \mapsto f(a,b), y \mapsto a\}$ *as a solution. On the other hand, the following equation has no solution.*



The term matching problem is a particular case of term unification. It is characterized by the condition that one of the sides of the equation $s \doteq t$, say t, contains no variables.

Definition 2.3.2 *Given a ranked alphabet* $\mathcal{F}$ *and a set of first-order variables* $\mathcal{X}$*, an instance of the first-order unification (matching) problem is a set* Δ *of equations* $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ *where* $t_i, s_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *(*$s_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *and* $t_i \in \mathcal{T}(\mathcal{F})$*). The question is to compute a substitution* σ *such that* $\sigma(s_i) = \sigma(t_i)$ *for all* $i \in \{1, \ldots, n\}$*.*

A first-order unification algorithm should not only state (non)unifiability of the input terms s, t but also produce a solution σ when it exists. In that sense, it is sufficient to consider a *most general unifier* of s and t, $\texttt{mgu}(s,t)$ in short, from which every other solution can be obtained by instantiation, and is unique up to renaiming of variables. When s and t do not unify, we say that $\texttt{mgu}(s,t)$ is not defined. It can be proven that, for unifiable first-order terms s, t, $\texttt{mgu}(s,t)$ is guaranteed to exist.

A simple algorithm for first-order unification might be the one described in Figure 2.1. Its correctness can be proved by induction on the complexity measure $\langle n_1, n_2 \rangle$ on terms, ordered by the (well-founded) lexicographic ordering on pairs of natural numbers where n_1 denotes the number of distinct variables in s and t, and $n_2 = \texttt{height}(s)$. Its termination can be argued in a similar way. In this section we will not get into a detailed analysis of its running time since the only purpose of this example is to illustrate the intuition behind some of the algorithms presented later in this work.

Note that an iterative version of this algorithm would run while s and t are different and traverse simultaneously $\texttt{pre}(s)$ and $\texttt{pre}(t)$ until finding a position k such that $\texttt{pre}(s)[k] \neq \texttt{pre}(t)[k]$. If $\texttt{pre}(s)[k]$ and $\texttt{pre}(t)[k]$ are function symbols then the unification fails. Otherwise, either $\texttt{pre}(s)$ or $\texttt{pre}(t)$, say $\texttt{pre}(s)$, contains a variable x at k. Note that, since s and t are terms, their symbols have fixed arity and thus the index k corresponds to a unique position $p \in \texttt{Pos}(s) \cap \texttt{Pos}(t)$, as commented in Section 2. If x properly occurs in the subterm of t at p, then we terminate, again stating non-unifiability. Otherwise, we replace x by the subterm of t at p everywhere, and re-start the process until both s and t become syntactically equal, in which case we state unifiability.

```
Global $\sigma$:  substitution; {Initialized to $\emptyset$}
------------------------------------------------------------
Function Unify($s, t$) returns boolean:
   If $s = t$ Then return True
   If $s = f(s_1, \ldots, s_m) \wedge t = f(t_1, \ldots, t_m)$, $m > 0$ Then
      return (Unify($s_1, t_1$)
      $\wedge$ Unify($\sigma(s_2), \sigma(t_2)$)
      $\vdots$
      $\wedge$ Unify($\sigma(s_m), \sigma(t_m)$))
   EndIf
   If $s = f(s_1, \ldots, s_n) \wedge t = g(t_1, \ldots, t_m)$, $n, m \geq 0$ Then return False
   If $s = x$ is a variable Then
      If $x$ occurs in $t$ Then return False
      Else
         $\sigma := \sigma \cup \{x \to t\}$
         return True
      EndIf
   ElseIf $t = x$ is a variable Then
      If $x$ occurs in $s$ Then return False
      Else
         $\sigma := \sigma \cup \{x \to s\}$
         return True
      EndIf
   EndIf
```

Figure 2.1: A First-order Unification Algorithm

On the other hand, an algorithm for first-order matching could be the one shown in Figure 2.2. Its correctness can be shown by induction on $\texttt{height}(t)$.

Note that, in this case, an iterative version of this algorithm would proceed similarly as in the previous algorithm. However, in this case we do not need to check whether s and t became equal after each instantiation of a variable since σ will always be a ground substitution.

```
Global $\sigma$:  substitution; {Initialized to $\emptyset$}
----------------------------------------------------------------
Function Match$(s,t)$ returns boolean:
   return AuxMatch$(s,t) \wedge \sigma(s) = t$
----------------------------------------------------------------
Function AuxMatch$(s,t)$ returns boolean:
   If $s = t$ Then {Do Nothing}
   If $s = f(s_1,\dots,s_m) \wedge t = f(t_1,\dots,t_m)$, $m > 0$ Then
      return $\big($AuxMatch$(s_1,t_1)$
      $\wedge$ AuxMatch$(\sigma(s_2),t_2)$
      $\vdots$
      $\wedge$ AuxMatch$(\sigma(s_m),t_m)\big)$
   EndIf
   If $s = f(s_1,\dots,s_m) \wedge t = g(t_1,\dots,t_n)$, $f \neq g$, $n,m \geq 0$ Then return False
   If $s = x$ is a variable Then
      $\sigma := \sigma \cup \{x \to t\}$
      return true
   EndIf
```

Figure 2.2: A First-order Matching Algorithm

These two simple algorithms for solving first-order unification and matching will be important later in this thesis since we will adapt them to the case when the input terms s and t are compressed.

2.3.2 Context Unification and Matching

As commented in the introduction, several variants and generalizations of the first-order term matching and unification problems have been studied. The extension introduced in this section consists on allowing context variables to occur in the input terms besides first-order variables. Recall that context variables are variables of arity one that can be replaced by contexts. By allowing this kind of variables in the terms of the equations we define the *context unification problem* and, consequently, the *context matching problem.* Given two terms $s, t \in \mathcal{T}(\mathcal{F},\mathcal{V})$, the *context unification* problem consists on deciding whether s and t are unifiable, i.e. whether there exists a substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{C}(\mathcal{F},\mathcal{V})$ such that $\sigma(s) = \sigma(t)$. This definition generalizes naturally to a set of equations, as stated in the following definition.

Definition 2.3.3 *Given a ranked alphabet $\mathcal{F}$, a set of first-order variables $\mathcal{X}_0$, and a set of context variables $\mathcal{X}_1$, an instance of the context unification (matching) problem is a set Δ of equations $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ where $t_i, s_i \in \mathcal{T}(\mathcal{F}, \mathcal{X}_0, \mathcal{X}_1)$ ($s_i \in \mathcal{T}(\mathcal{F}, \mathcal{X}_0, \mathcal{X}_1)$ and $t_i \in \mathcal{T}(\mathcal{F})$). The question is to compute a substitution σ mapping first-order variables to terms and context variables to contexts, such that $\forall i \in \{1, \ldots, n\} : \sigma(s_i) = \sigma(t_i)$ holds.*

The decidability of context unification is a long-standing open question. Several variants and particular cases has been solved [Vil04], and the case where at most two different context variables occur in the input terms was proven decidable in [SSS02].

On the other hand, it is easy to prove that context matching (and thus context unification), is NP-hard. We will present a sketch of a proof to illustrate the difficulties to get NP-hardness of more restrictive cases of context matching/unification. In fact, no NP-hardness results are known for context unification with a bounded number of context variables. Moreover, the particular case where at most three different context variables occur in the input terms is neither known to be NP-hard nor decidable.

We will use a reduction from *monotone 1-in-3-SAT*, which is known to be NP-hard [Sch78]. The *monotone 1-in-3-SAT* problem consists on, given a propositional formula ϕ in conjuntive normal form such that

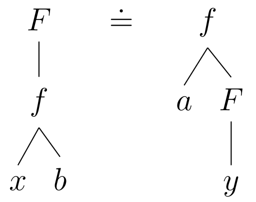
- each clause has exactly three literals, and
- all literals in ϕ are positive, i.e. clauses do not contain negated variables,

decide whether there is a satisfying assignment for ϕ making true exactly one variable of every clause.

For a clause $C = (x \vee y \vee z)$, we define $\mathtt{eq}(C)$ as the equation $F_C(f(x,y,z)) \doteq f(f(1,0,0), f(0,1,0), f(0,0,1))$, where F_C is a context variable, x, y, z are first-order variables, and $1, 0$ are constants. It is easy to see that an instance ϕ of monotone 1-in-3-SAT has a solution if and only if the context matching instance $\Delta = \{\mathtt{eq}(C) \mid C \in \phi\}$ has a solution. Moreover, Δ has polynomial size with respect to ϕ. Note that, in this reduction, it is crucial to have a different context variable F_C for each clause $C \in \phi$. It is not clear how to obtain a reduction using only a constant number of context variables, even if they are allowed to occur in the right-hand side of the equations.

As mentioned in the introduction, in this thesis we deal with the particular case of context unification where only one context variable may appear in the input terms, possibly with many occurrences. We call that particular case *one context unification.*

Example 2.3.4 *Consider the following term equation:*



where x, y are first-order variables and F is a context variable that can be replaced by any context. This concrete instance of the one context unification problem has several solutions, such as, $\{F \mapsto f(a, \bullet), x \mapsto a, y \mapsto b\}$ and $\{F \mapsto f(a, f(a, \bullet)), x \mapsto a, y \mapsto b\}$.

In [GGSST10], we proved that one context unification is in NP. That result is presented in Chapter 5 of this thesis. While one context unification is known to be in NP, neither a polynomial time algorithm nor a NP-hardness proof are known for this problem. An interesting partial result presented in Chapter 5 is that one context unification can be solved in polynomial time if the input set of equations contains an equation of the form $F(s) \doteq c(F(t))$, where F is the context variable and c is a non-empty context. It follows that, to find a polynomial time algorithm for one context unification, it suffices to consider the case where right hand-sides of equations do not contain the context variable. Let us present an interesting example of this case that we will reuse later in Chapter 8.

Example 2.3.5 *Let s be $f(x_0, x_0)$, let t^0 be $f(a, b)$, and let us define t^n, for any $n > 0$, recursively as $t^n = f(f(x_n, x_n), t^{n-1})$, where x_i is a first-order variable, for every $i \in \{0, \dots, n\}$. Hence, $t^1 = f(f(x_1, x_1), t^0) = f(f(x_1, x_1), f(a, b))$ and $t^2 = f(f(x_2, x_2), t^1) = f(f(x_2, x_2), f(f(x_1, x_1), f(a, b)))$.*

Consider, the following instance of the one context unification problem: $\{F(a) \doteq s, F(b) \doteq t^2\}$ where F is the context variable and a, b are constants. Note that there are four different solutions $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ that can be univoquely characterized by the hole path of their respective instantiations of F as $\mathtt{hp}(\sigma_1(F)) = 1.1.2$, $\mathtt{hp}(\sigma_2(F)) = 1.2.1$, $\mathtt{hp}(\sigma_3(F)) = 1.1.1$, and $\mathtt{hp}(\sigma_4(F)) = 1.2.2$.

Now consider the instance $\{F(a) \doteq s, F(b) \doteq t^n\}$ where F is the context variable and a, b are constants, which actually describes a family of instances parametrized by n. This instance has 2^n different solutions.

Another particular case of context unification covered in this work is k-context matching, the particular case of context matching where the number of different context variable occurring in the input is bounded by a constant number k. In [GGSS08], we proved that k-context matching can be solved in polynomial time even when the input terms are represented using a Directed Acyclic Graph. That result is presented in Chapter 7 of this thesis.

Chapter 3

Compressed Term Representation

As mentioned in the introduction, in this thesis we consider variants of the term unification problem depending on which formalism is used for term representation. In this chapter we introduce two different formalisms that allow to represent terms exponentially more succinctly than using an explicit representation: Directed Acyclic Graphs (DAGs) and Singleton Tree Grammars (STGs). While having a succinct representation has many advantages in practice, it also comes with a cost, since performing simple operations on terms efficiently may become more complex in the compressed setting that in the explicit representation. In this section we will also present constructions to perform several basic operations on STGs efficiently that will be used later in our algorithms and proofs.

3.1 DAGs

The main idea behind the well-known DAG representation is that, if a certain subterm t has many occurrences in a given term, then we do not need to maintain a copy of t for each of its occurrences, but simply a reference to it.

Definition 3.1.1 *A term DAG is a rooted Directed Acyclic Graph whose nodes are labeled with function symbols, constants, or variables, whose outgoing edges from a node are ordered, and the outdegree of any node labeled with a symbol f is equal to the arity of f.*

In such a graph, each node has an interpretation as a term, and hence we refer to nodes and terms as if they were the same. DAGs allow to represent

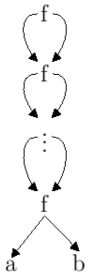


Figure 3.1: Encoding using a DAG of term in example 3.1.2

terms of exponential width in linear space, as illustrated in the following example.

Example 3.1.2 *Given the set of equations* $\{t_1 = f(a,b),\ t_2 = f(t_1,t_1), \ldots, t_n = f(t_{n-1}, t_{n-1})\}$*, using a DAG to represent* t_n *provides an efficient encoding as shown in Figure 3.1.*

One of the reasons for the exponential execution time of Robinson's algorithm for first-order unification [Rob65] is the exponential size increase of the terms to be unified due to instantiation of variables. The DAG structure for term representation is used in later algorithms to keep the size of this terms linearly bounded. Furthermore, note that since each node in a term DAG has an interpretation as a term, once two subterms are unified they are represented by the same node in the minimal DAG, which helps to avoid repeated calculations. Moreover, a minimal DAG representing a set of terms can be efficiently computed. We postpone giving a formal definition of DAGs to the next section, since DAGs can be seen as a particular case of STGs, a grammar-based formalism for term representation.

3.2 Grammars

In this work we consider *Singleton Tree Grammars (STG)* for term compression. This kind of grammars are a generalization of *Singleton Context-Free Grammars (SCFG)* [LSSV08, Pla94], which can only generate strings, extending the expressivity of SCFGs by terms and contexts. This is consistent with [BLM08], and also with the Context-Free tree Grammars in [CDG$^+$07].

First of all it is necessary to define the well-known *Context-Free Grammars (CFG)*. Then, by making a restriction on the form of the rules we define *Singleton Context Free Grammars (SCFG)*, and finally, by extending SCFGs to represent terms we introduce *Singleton Tree Grammars (STG)*.

Definition 3.2.1 *A context-free grammar is a quadruple* $G = (V, \Sigma, P, S)$ *where* V *is a finite set of variables (nonterminals),* Σ *is a finite set of terminals disjoint with* V*,* $S \in V$ *is the start symbol and* P *is a finite set of production rules of the form* $Z \to \alpha$ *where* $Z \in V$ *and* $\alpha \in (V \cup \Sigma)^*$ *.*

Example 3.2.2 *A context-free grammar for the language consisting of all strings over* $\{a, b\}$ *with a different number of a's and b's is the one with the following set of rules:*

$S \to U$	$S \to V$	$U \to TaU$
$U \to TaT$	$V \to TbV$	$V \to TbT$
$T \to aTbT$	$T \to bTaT$	$T \to \lambda$

Here, the nonterminal T *can generate all strings with the same number of a's as b's, the nonterminal* U *generates all strings with more a's than b's and the nonterminal* V *generates all strings with fewer a's than b's. The symbol* λ *denotes the empty string.*

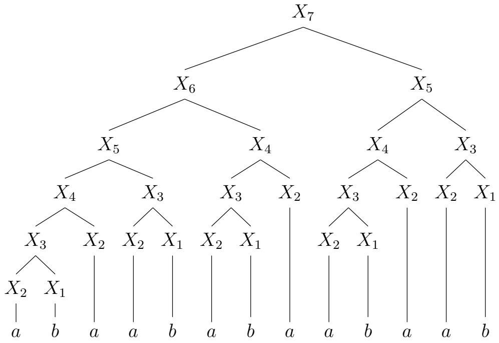
As we can see in the example above, the nonterminals T, U, V are recursive. For this reason, arbitrarily long strings may be generated. Furthermore, there is more than one rule containing a given nonterminal in its left-hand side, i.e. the grammar in nondeterministic, and hence, some nonterminals can generate more than one string. SCFGs are called singleton because each nonterminal generates just one string. This property is guaranteed by forcing such grammars to be deterministic and nonrecursive by definition.

Definition 3.2.3 *A* Singleton Context Free grammar (SCFG) *is a nonrecursive context-free grammar such that every nonterminal occurs in the left hand-side of a rule exactly once. Then every nonterminal* Z *generates just one word, denoted* w_Z*. We do not distinguish a particular start symbol. Hence, SCFGs are defined as a 3-tuple* $G = (V, \Sigma, P)$*, analogously to context-free grammars.*

Alternatively, SCFGs can be defined in the following way. They contain nonterminals $X_1, \ldots, X_n$ where every nonterminal X_i occurs in a left hand-side of exactly one rule of the form either $X_i \to c$, for some $c \in \Sigma$, or $X_i \to X_j X_k$, for some $j, k < i$. Note that, with this alternative definition, SCFGs are in Chomsky Normal Form.

Example 3.2.4 *Consider string* $s = abaababaabaab$*. It can be generated by an SCFG with the following set of rules:* $\{X_7 \to X_6X_5, X_6 \to X_5X_4, X_5 \to X_4X_3, X_4 \to X_3X_2, X_3 \to X_2X_1, X_2 \to a, X_1 \to b\}$*. We say that* X_7 *generates* s*, denoted* $w_{X_7} = s$*.*

The following picture shows the derivation tree to obtain s. Note that compression is obtained by reusing nonterminals that generate repeated subwords of s.



As shown in the following example, using SCFG, words of exponential size can be represented in linear space. This compression ratio is also an upper bound for the compression ratio of SCFGs, which can be easily shown by induction.

Example 3.2.5 *The SCFG G with set of rules $\{A_0 \to a, A_1 \to A_0A_0, \ldots, A_n \to A_{n-1}A_{n-1}\}$ generates the word a^{2^n}.*

Now we can define *Singleton Tree Grammars* (STGs) as an extension of the already presented SCFGs in order to capture terms and contexts. As a last ingredient, let us fix a countable set $\mathcal{Y} = \{y_1, y_2 \ldots\}$ whose elements are called *parameters.*

Definition 3.2.6 *A* Singleton Tree Grammar (STG) *G is a 4-tuple $\langle \mathcal{N}, \Sigma, R, S\rangle$, where*

- *$\mathcal{N}$ is a ranked alphabet whose elements are called nonterminals.*
- *Σ is a ranked alphabet called signature.*
- *R is a finite set of rules of the form $N \to t$ where $N \in \mathcal{N}$, $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N} \cup \{y_1, \ldots, y_{\mathtt{ar}(N)}\})$, $t \notin \mathcal{Y}$, and each of the parameters $\{y_1, \ldots, y_{\mathtt{ar}(N)}\}$ appears in t exactly once.*
- *S is the initial nonterminal of arity* 0.

The sets $\mathcal{N}$ and Σ must be disjoint, each nonterminal N appears as a left-hand side of just one rule of R. Let $N_1 >_G N_2$ for two nonterminals N_1, N_2, iff $(N_1 \to \alpha) \in R$, and N_2 occurs in α. The STG must be nonrecursive, i.e., the transitive closure $>_G^+$ must be terminating. The depth *of G is the* *maximal length of a chain in* $>_G^+$*. We define the derivation relation* $\Rightarrow_G$ *on* $\mathcal{T}(\mathcal{F} \cup \mathcal{N} \cup \mathcal{Y})$ *as follows:* $t \Rightarrow_G t'$ *iff there exists* $(A \to s) \in R$ *with* $\mathtt{ar}(A) = n, t = c(A(t_1, \ldots, t_n))$ *and* $t' = c(\sigma(s))$*, where* $\sigma = \{y_1 \to t_1, \ldots, y_n \to t_n\}$ *and* c *is a context in* $\mathcal{C}(\mathcal{F} \cup \mathcal{N} \cup \mathcal{Y})$*. The* term pattern *generated by a nonterminal* N *of* G*, denoted* w_N*, is the term* $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Y})$ *such that* $N \Rightarrow_G^* t$*.*

The following example shows how, in contrast to the DAG representation that is based on sharing of repeated subterms, STGs allow to compress terms by reusing repeated tree patterns.

Example 3.2.7 *The STG with the two rules rules* $S \to A(A(a, b), A(b, a))$ *and* $A(y_1, y_2) \to f(h(y_1), h(y_2))$ *generates the term* $f(f(h(a), h(b)), f(h(b), h(a)))$ *from* S*.*

Definition 3.2.8 *Let* $G = \langle \mathcal{N}, \Sigma, R, S\rangle$ *be an STG.* G *is called* k*-bounded if* k *is the maximum arity of the nonterminals in* $\mathcal{N}$*.*

Definition 3.2.9 *Let* $G = \langle \mathcal{N}, \Sigma, R, S\rangle$ *be an STG. The size of* G*, denoted* $|G|$*, is defined as* $\sum_{(A \to t_A) \in R} |t_A|$*.*

Definition 3.2.10 *The* depth *within* G *of a nonterminal* N *is defined recursively as* $\mathtt{depth}(N) := 1 + \mathtt{max}\{\mathtt{depth}(N') \mid N'$ *is a nonterminal in* u *where* $N \to u \in G\}$ *and the maximum of an empty set is assumed to be* 0*.*

The depth of a grammar G *is the maximum of the depths of all nonterminals of* G*, and it is denoted as* $\mathtt{depth}(G)$*.*

In [LMSS12], it is shown how to transform a given STG into an equivalent 1-bounded STG. The transformation takes linear time and space. This result allows us to consider, without loss of generality, a simpler definition of STG:

Definition 3.2.11 *A* Singleton Tree Grammar (STG) *is a 4-tuple* $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$*, where* $\mathcal{TN}$ *is a set of tree/term nonterminals, or nonterminals of arity* 0*,* $\mathcal{CN}$ *is a set of context nonterminals, or nonterminals of arity* 1*, and* Σ *is a ranked alphabet whose elements are called terminals, such that the sets* $\mathcal{TN}$*,* $\mathcal{CN}$*, and* Σ *are pairwise disjoint. The set of nonterminals* $\mathcal{N}$ *is defined as* $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$*. The rules in* R *may be of the form:*

- $A \to \alpha(A_1, \ldots, A_m)$*, where* $A, A_i \in \mathcal{TN}$*, and* $\alpha \in \Sigma$ *is an* m*-ary terminal symbol.*
- $A \to C_1 A_2$ *where* $A, A_2 \in \mathcal{TN}$*, and* $C_1 \in \mathcal{CN}$*.*

- *$C \to \bullet$ where $C \in \mathcal{CN}$.*
- *$C \to C_1C_2$, where $C, C_i \in \mathcal{CN}$.*
- *$C \to \alpha(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)$, where $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m \in \mathcal{TN}$, $C, C_i \in \mathcal{CN}$, and $\alpha \in \Sigma$ is an m-ary terminal symbol.*
- *$A \to A_1$, (λ-rule) where A and A_1 are term nonterminals.*

Let $N_1 >_G N_2$ for two nonterminals N_1, N_2, iff $(N_1 \to t) \in R$, and N_2 occurs in t. The STG must be nonrecursive, i.e. the transitive closure $>_G^+$ must be terminating. Furthermore, for every nonterminal N of G there is exactly one rule having N as left-hand side.

From now on, unless explicitly stated otherwise, we consider the previous definition of STG. As mentioned above, the two definitions of STG given here are equivalent up to a linear time and space transformation. Note that we have chosen Σ instead of $\mathcal{F}$ to denote the set of terminals of the grammar, although it is also a signature. The justification for that choice is as follows. In this work, STG's are used for representing terms with occurrences of context and first-order variables. In particular, a terminal A of an STG G generates a term. If Σ was $\mathcal{F}$ we would be able to represent just ground terms. Thus, Σ must also contain first-order variables as terminals of arity 0 and context variables as terminals of arity 1.

A Directed Acyclic Graph can be defined as a particular case of an STG (in fact, this representation is in direct correspondence with the classic implementation of graphs using adjacency lists).

Definition 3.2.12 *A* DAG *is a* 0*-bounded STG.*

Example 3.2.13 *Given the set of equations $\{t_1 = f(a,b), t_2 = f(t_1,t_1), \ldots, t_n = f(t_{n-1}, t_{n-1})\}$, using an STG to represent t_n provides an efficient encoding (as shown in example 3.1.2 for the case of the DAG representation).*

$T_n \to f(T_{n-1}, T_{n-1})$
$\vdots$
$T_2 \to f(T_1, T_1)$
$T_1 \to f(A, B)$
$A \to a$
$B \to b$

Analogously to the case of SCFGs, STG-represented terms may have exponential height with respect to the size of the grammar in contrast to DAGs, which only allow for a linear height in the (notational) size of the DAGs as shown in the following example.

Example 3.2.14 *The term $s = f^{2^n}(a)$ described by the following grammar would have exponential height in a term or DAG representation.*

$S \to C_n A_a$
$A_a \to a$
$C_\bullet \to \bullet$
$C_0 \to f(C_\bullet)$
$C_1 \to C_0 C_0$
$C_2 \to C_1 C_1$
$C_3 \to C_2 C_2$
$\vdots$
$C_n \to C_{n-1} C_{n-1}$

A DAG G is called *optimally compressed* if equal terms are represented by the same term nonterminal. Transforming a DAG into optimally compressed form can be performed in time $\mathcal{O}(n \log n)$ [DST80].

At this point, let us define a notion of size and depth of an STG (considering Definition 3.2.11), a DAG, and an SCFG.

Definition 3.2.15 *The* size $|G|$ *of an STG G is the number of non-terminals in G The* depth *within G of a nonterminal N is defined recursively as* $\mathtt{depth}(N) := 1 + \mathtt{max}\{\mathtt{depth}(N') \mid N'$ *is a nonterminal in u where* $N \to u \in G\}$ *and the maximum of an empty set is assumed to be* 0.

The depth of an STG G *is the maximum of the depths of all nonterminals of G, and it is denoted as* $\mathtt{depth}(G)$.

In the case of DAGs and SCFGs, size and depth are defined analogously.

Plandowski [Pla94, Pla95] proved decidability in polynomial time of the word problem for SCFGs, i.e., given an SCFG P and two nonterminals A and B, deciding whether $w_A = w_B$. The best complexity for this problem has been obtained recently by Lifshits [Lif07] with time $\mathcal{O}(|P|^3)$. In [BLM08] Plandowski's result is generalized to STGs. Since the result in [BLM08] is based on a linear reduction from terms to words and a direct application of Plandowski's result, it also holds for Lifshits result. Hence, we have the following.

Theorem 3.2.16 *([Lif07, BLM08]) Given an STG G, and two tree nonterminals A, B from G, it is decidable in time $\mathcal{O}(|G|^3)$ whether $w_A = w_B$.*

Several properties of STGs are efficiently decidable. The following lemmas will be used all along this thesis.

Lemma 3.2.17 *Let G be an STG. The number $|w_N|$, for every nonterminal N of G, is computable in time $\mathcal{O}(|G|)$.*

Proof. We give an alternative definition of $|w_N|$ recursively as follows.

- if $(N \to \bullet) \in G$ then $|w_N| = 1$.
- if $(N \to f(N_1, \ldots, N_m)) \in G$ then $|w_N| = 1 + |w_{N_1}| + \ldots + |w_{N_m}|$, where $N_1, \ldots, N_m$ are nonterminals of G and f is a function symbol with $\texttt{ar}(f) = m$.
- if $(N \to C_1 N_2) \in G$ then $|w_N| = |w_{C_1}| + |w_{N_2}| - 1$, where C_1 is a context nonterminal and N_2 is a nonterminal of G.

The correctness of the above definition can be shown by induction on the size of w_N. Moreover, since the recursive calls in the definition of $|w_N|$ will be done, at most, over all the nonterminals of G, $|w_N|$ is computable in linear time with respect to $|G|$ using a dynamic programming scheme. □

Lemma 3.2.18 *Given an STG G, a terminal α, and a nonterminal N of G, it is decidable in time $\mathcal{O}(|G|)$ whether α occurs in w_N.*

Proof. Whether α occurs in w_N can be computed efficiently again using a dynamic programming scheme: note that α occurs in w_N iff either $w_N \to \alpha \in G$, or α occurs in $w_{N'}$ for some nonterminal N' occurring in the right-hand side of the rule for N. □

3.2.1 Grammar Constructions

In this section we describe the operations on STGs involved in our algorithms for different variants of compressed term unification and matching.

For example, in [BLM08] it was shown how to succinctly represent the preorder traversal word of a term generated by an STG using an SCFG. We describe that construction in Section 3.2.1. More concretely, given STG-compressed terms s, t, we show how to efficiently compute an SCFG Pre_G with nonterminals $\mathcal{P}_s$ and $\mathcal{P}_t$ generating $\mathrm{Pre}(s)$ and $\mathrm{Pre}(t)$, respectively.

We also need to compute, given Pre_G, the smallest index k in which the compressed strings $\mathrm{Pre}(s)$ and $\mathrm{Pre}(t)$ differ. In Section 3.2.1 we show how to perform this task efficiently. Our approach is based on a recent result on

$$\begin{array}{rcl}
A \to f(A_1, \ldots, A_m) & \Rightarrow & \mathcal{P}_A \to f\mathcal{P}_{A_1} \ldots \mathcal{P}_{A_m} \\
A \to C_1 A_2 & \Rightarrow & \mathcal{P}_A \to \mathcal{L}_{C_1}\mathcal{P}_{A_2}\mathcal{R}_{C_1} \\
A \to A_1 & \Rightarrow & \mathcal{P}_A \to \mathcal{P}_{A_1} \\
C \to C_1 C_2 & \Rightarrow & \begin{cases} \mathcal{L}_C \to \mathcal{L}_{C_1}\mathcal{L}_{C_2} \\ \mathcal{R}_C \to \mathcal{R}_{C_2}\mathcal{R}_{C_1} \end{cases} \\
C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m) & \Rightarrow & \begin{cases} \mathcal{L}_C \to f\mathcal{P}_{A_1} \ldots \mathcal{P}_{A_{i-1}}\mathcal{L}_{C_i} \\ \mathcal{R}_C \to \mathcal{R}_{C_i}\mathcal{P}_{A_{i+1}} \ldots \mathcal{P}_{A_n} \end{cases} \\
C \to \bullet & \Rightarrow & \begin{cases} \mathcal{L}_C \to \lambda \\ \mathcal{R}_C \to \lambda \end{cases}
\end{array}$$

Figure 3.2: Generating the Preorder Traversal

compressed string processing [Lif07]. As commented in Section 2, k corresponds to a unique position $p \in \mathsf{Pos}(s) \cap \mathsf{Pos}(t)$. In Section 3.2.1, we present the procedure, given G and k, to extend G such that a new nonterminal generates $t|_p$. Avoiding the explicit calculation of p refines the approach presented in previous work in STG-compressed first-order unification [GGSS09] in order to obtain a faster algorithm.

We also need to apply substitutions once a variable is isolated. Performing a replacement of a first-order variable x (context variable F) by a term u (context c) is easily representable with STGs by simply transforming x (F) into a nonterminal x (F) of the grammar and adding rules such that x (F) generates u (c).

Finally, we also present operations of STG to adapt the subcontext operation presented in Section 2.1 to the compressed setting.

Computing the preorder traversal of a term.

In [BLM08] it is shown how to construct, from a given STG G, an SCFG Pre_G representing the preorder traversals of the terms and contexts generated by G. We reproduce that construction here, presented in Figure 3.2 as a set of rules indicating, for each term nonterminal A and its rule $A \to \alpha$ of G, which rule $\mathcal{P}_A \to \alpha'$ of Pre_G is required in order to make a nonterminal $\mathcal{P}_A$ of Pre_G satisfy $w_{\mathrm{Pre}_G, \mathcal{P}_A} = \mathrm{Pre}(w_{G,A})$. To this end, for each context nonterminal C of G we also need nonterminals of Pre_G generating the preorder traversal to the left of the hole ($\mathcal{L}_C$), and the preorder traversal to the right of the hole ($\mathcal{R}_C$).

It is straightforward to verify by induction on the depth of G that, for every term nonterminal A of G, the corresponding newly generated nonterminal $\mathcal{P}_A$ of Pre_G generates $\mathrm{Pre}(w_A)$.

Lemma 3.2.19 *Let G be an STG. An SCFG Pre_G of size $\mathcal{O}(|G|)$ can be constructed in time $\mathcal{O}(|G|)$ such that, for each nonterminal N of G, there exists a nonterminal $\mathcal{P}_N$ in Pre_G satisfying $w_{Pre_G,\mathcal{P}_N} = Pre(w_{G,N})$.*

Proof. This follows trivially from the construction in Figure 3.2. □

Computing the first different position of two words

Given two nonterminals p_1 and p_2 of an SCFG P, we want to find the smallest index k such that $w_{p_1}[k]$ and $w_{p_2}[k]$ are different. In order to solve this problem, a linear search over the generated words w_{p_1} and w_{p_2} is not a good idea, since their sizes may be exponentially big with respect to the size of P. Hence, one may be tempted to apply a binary search since prefixes are efficiently computable with SCFGs and equality is checkable in time $\mathcal{O}(|P|^3)$, which would lead to $\mathcal{O}(|P|^4)$ time complexity. However, we will use more specific information from Lifshits' work [Lif07] to obtain $\mathcal{O}(|P|^3)$ time complexity.

Lemma 3.2.20 *[Lif07] Let G be an SCFG. Then a data structure can be computed in time $\mathcal{O}(|G|^3)$ which allows to answer to the following question in time $\mathcal{O}(|G|)$: given two nonterminals N_1 and N_2 of G and an integer value k, does w_{N_1} occur in w_{N_2} at position k?*

Thus, assume that the precomputation of Lemma 3.2.20 has been done (in time $\mathcal{O}(|P|^3)$), and hence we can answer whether a given w_{p_1} occurs in a given w_{p_2} at a certain position in time $\mathcal{O}(|P|)$.

For finding the first different position between p_1 and p_2, we can assume $|w_{p_1}| \leq |w_{p_2}|$ without loss of generality. Moreover, we also assume $w_{p_1} \neq w_{p_2}[1..|w_{p_1}|]$, i.e w_{p_1} is not a prefix of w_{p_2}. Note that this condition is necessary for the existence of a different position between w_{p_1} and w_{p_2}, and that this will be the case when p_1 and p_2 generate the preorder traversals of different trees. Finally, we can assume that P is in Chomsky Normal Form i.e. every rule in P is either of the form $X \to YZ$ or $X \to a$, where a is a terminal and X, Y, Z are nonterminals of P. Note that, if this was not the case, we can force this assumption with a linear time and space transformation.

We generalize our problem to the following question: given two nonterminals p_1 and p_2 of P and an integer k' satisfying $k' + |w_{p_1}| \leq |w_{p_2}|$ and $w_{p_1} \neq w_{p_2}[(k'+1)..(k'+|w_{p_1}|)]$, which is the smallest $k \geq 1$ such that $w_{p_1}[k]$ is different from $w_{p_2}[k'+k]$? (Note that we recover the original question by fixing $k' = 0$).

This generalization is solved efficiently by the recursive algorithm given in Figure 3.3, as can be shown inductively on the depth of p_1. By Lemma 3.2.20,

$$\texttt{index}(p_1,p_2,k',P) = \begin{cases} 1 & \text{, if } |w_{p_1}| = 1 \\ \texttt{index}(p_{11},p_2,k',P) & \text{, if } (p_1 \to p_{11}p_{12}) \in P \wedge \\ & \quad w_{p_{11}} \neq w_{p_2}[(k'+1)\dots(k'+|w_{p_{11}}|)] \\ |w_{p_{11}}| + & \text{, if } (p_1 \to p_{11}p_{12}) \in P \wedge \\ \texttt{index}(p_{12},p_2,k'+|w_{p_{11}}|,P) & \quad w_{p_{11}} = w_{p_2}[(k'+1)\dots(k'+|w_{p_{11}}|)] \end{cases}$$

Figure 3.3: Algorithm for the Index of the First Difference

each call takes time $\mathcal{O}(|P|)$, and at most $\texttt{depth}(P)$ calls are executed. Thus, the most expensive part of computing the first different position of w_{p_1} and w_{p_2} is the pre-computation given by Lemma 3.2.20, that is, $\mathcal{O}(|P|^3)$.

Lemma 3.2.21 *Let P be an SCFG, and let p_1, p_2 be nonterminals of P such that $w_{p_1} \neq w_{p_2}$. The first position k where w_{p_1} and w_{p_2} differ is computable in time $\mathcal{O}(|P|^3)$.*

Isolating variables

As commented in Section 2, every index k in the preorder traversal word of a term t corresponds to the position $p = \texttt{iPos}(t,k)$. We show how to efficiently compute, given G and k, an extension of an STG G with a nonterminal generating $t|_p$. We use the SCFG Pre_G presented in Figure 3.2.

Definition 3.2.22 *Let G be an STG. Let N a nonterminal of G, and let k be a natural number satisfying $k \leq Pre(w_{G,N})|$. We recursively define $\texttt{kExt}(G,N,k)$ as an extension of G as follows:*

- *If $k = 1$ then $\texttt{kExt}(G,N,k) = G$. In the next cases we assume $k > 1$.*
- *If $(N \to f(N_1,\dots,N_{i-1},N_i,\dots,N_m)) \in G$ and $1 + |w_{N_1}| + \dots + |w_{N_{i-1}}| = k' < k \leq k' + |w_{N_i}|$ then $\texttt{kExt}(G,N,k) = \texttt{kExt}(G,N_i,k-k')$.*
- *If $(N \to C_1A_2) \in G$ and $k \leq |w_{\texttt{Pre}_G,\mathcal{L}_{C_1}}|$ then $\texttt{kExt}(G,N,k)$ includes $\texttt{kExt}(G,C_1,k)$, which contains a nonterminal N' generating the subterm of w_{G,C_1} at position $\texttt{iPos}(w_{G,C_1},k)$. If N' is a context nonterminal then $\texttt{kExt}(G,N,k)$ additionaly contains the rule $A \to N'A_2$, where A is a new term nonterminal.*
- *If $(N \to C_1C_2) \in G$ and $k \leq |w_{\texttt{Pre}_G,\mathcal{L}_{C_1}}|$ then $\texttt{kExt}(G,N,k)$ includes $\texttt{kExt}(G,C_1,k)$, which contains a nonterminal N' generating the subterm of w_{G,C_1} at position $\texttt{iPos}(w_{G,C_1},k)$. If N' is a context nonterminal then $\texttt{kExt}(G,N,k)$ additionaly contains the rule $C \to N'C_2$, where C is a new context nonterminal.*

- *If* $(N \to C_1 N_2) \in G$ *and* $k' = |w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}| < k \leq |w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}| + |w_{N_2}|$ *then* $\mathtt{kExt}(G, N, k) = \mathtt{kExt}(G, N_2, k - k')$.

- *If* $(N \to C_1 N_2) \in G$ *and* $|w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}| + |w_{N_2}| < k$ *then* $\mathtt{kExt}(G, N, k) = \mathtt{kExt}(G, C_1, k - |w_{N_2}| + 1)$.

- *If* $(N \to A_2) \in G$ *then* $\mathtt{kExt}(G, N, k) = \mathtt{kExt}(G, A_2, k)$.

- *In any other case* $\mathtt{kExt}(G, N, k)$ *is* undefined*.*

The following lemma states the correctness of the previous definition.

Lemma 3.2.23 *Let* G *be an STG. Let* N *a nonterminal of* G*, and let* k *be a natural number such that* $k \leq |\mathtt{Pre}(w_{G,N})|$*. Then* G *can be extended to an STG* G' *in time* $\mathcal{O}(|G|)$ *with* $\mathcal{O}(\mathtt{depth}(G))$ *new nonterminals such that one of them generates the subterm of* $w_{G,N}$ *at position* $\mathtt{iPos}(w_{G,N}, k)$*.*

Proof.

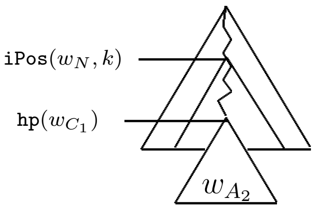
The fact that $\mathtt{kExt}(G, N, k)$ is an extension of G satisfying the statements of the lemma follows by induction on $\mathtt{depth}(N)$:

For the base case we assume $\mathtt{depth}(N) = 1$, then $|\mathtt{Pre}(w_N)| = 1$ and, since $k \leq |\mathtt{Pre}(w_N)|$, $k = 1$. Hence, $w_N|_{\mathtt{iPos}(w_N,k)} = w_N|_{\mathtt{iPos}(w_N,1)} = w_N|_\lambda = w_N$ by definition of $\mathtt{iPos}$, and definition of subterm of w_N. Thus, since N is a nonterminal of G then $\mathtt{kExt}(G, N, k) = G$ generates w_N. For the induction step we distinguish cases according to the definition of $\mathtt{kExt}(G, N, k)$:

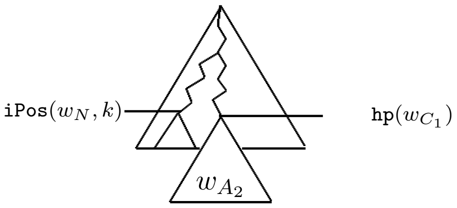
- Assume that $(N \to f(N_1, \ldots, N_{i-1}, N_i, \ldots, N_m)) \in G$ and $1 + |w_{N_1}| + \ldots + |w_{N_{i-1}}| = k' < k \leq k' + |w_{N_i}|$. By definition of $\mathtt{iPos}(w_N, k)$, it holds that $\mathtt{iPos}(w_N, k) = i \cdot \mathtt{iPos}(w_{N_i}, k - k')$. Hence, $w_N|_{\mathtt{iPos}(w_N,k)} = w_{N_i}|_{\mathtt{iPos}(w_{N_i},k-k')}$ by definition of subterm of w_N. Moreover, since in this case $\mathtt{kExt}(G, N, k) = \mathtt{kExt}(G, N_i, k - k')$, the fact that $\mathtt{kExt}(G, N, k)$ generates $w_N|_{\mathtt{iPos}(w_N,k)}$ follows by induction hypothesis.

- If $(N \to C_1 A_2) \in G$ and $k \leq |w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}|$ then either $\mathtt{iPos}(w_N, k) \preceq \mathtt{hp}(w_{C_1})$ or $\mathtt{iPos}(w_N, k)$ and $\mathtt{hp}(w_{C_1})$ are disjoint. Both situations are ilustrated by the following figure:

  In the former case, $w_{C_1}|_{\mathtt{iPos}(w_{C_1},k)}$ is a prefix of w_{C_1} and $w_N|_{\mathtt{iPos}(w_N,k)} = w_{C_1}|_{\mathtt{iPos}(w_{C_1},k)} w_{A_2}$. In this case $\mathtt{kExt}(G, N, k)$ is constructed using $\mathtt{kExt}(G, C_1, k)$, which constains a new nonterminal N' generating $w_{C_1}|_{\mathtt{iPos}(w_{C_1},k)}$ by induction hypothesis, plus the rule $A \to N' A_2$, where A is a new term nonterminal. Hence, it holds that $w_A = w_{N'} w_{A_2} = w_{C_1}|_{\mathtt{iPos}(w_N,k)} w_{A_2} = w_N|_{\mathtt{iPos}(w_N,k)}$ and thus $\mathtt{kExt}(G, N, k)$ generates $w_N|_{\mathtt{iPos}(w_N,k)}$. In the latter case, $w_N|_{\mathtt{iPos}(w_N,k)} = w_{C_1}|_{\mathtt{iPos}(w_{C_1},k)}$.



iPos$(w_N, k) \prec$ hp(w_{C_1})



iPos(w_N, k) and hp(w_{C_1}) disjoint

By induction hyphotesis, $\texttt{kExt}(G, C_1, k)$ generates $w_{C_1}|_{\texttt{iPos}(w_N,k)}$ and, since $w_{C_1}|_{\texttt{iPos}(w_{C_1},k)}$ is a term, $\texttt{kExt}(G, N, k) = \texttt{kExt}(G, C_1, k)$ and $\texttt{kExt}(G, N, k)$ generates $w_N|_{\texttt{iPos}(w_N,k)}$, again by induction hypothesis.

- The case where $(N \to C_1C_2) \in G$ and $k \leq |w_{\texttt{Pre}_G, \mathcal{L}_{C_1}}|$ is solved analogously to the previous one.

- If $(N \to C_1N_2) \in G$ and $k' = |w_{\texttt{Pre}_G, \mathcal{L}_{C_1}}| < k \leq |w_{\texttt{Pre}_G, \mathcal{L}_{C_1}}| + |w_{N_2}|$ then $\texttt{hp}(w_{C_1}) \preceq \texttt{iPos}(w_N, k)$ and hence $\texttt{iPos}(w_N, k)$ is of the form $\texttt{hp}(w_{C_1}) \cdot \texttt{iPos}(w_{N_2}, k - k')$. Moreover, by definition of subterm of w_N, it holds that $w_N|_{\texttt{iPos}(w_N,k)} = w_{N_2}|_{\texttt{iPos}(w_{N_2},k-k')}$ and thus, since in this case $\texttt{kExt}(G, N, k) = \texttt{kExt}(G, N_2, k - k')$, the fact that $\texttt{kExt}(G, N, k)$ generates $w_N|_{\texttt{iPos}(w_N,k)}$ follows from induction hyphotesis.

- Assume that $(N \to C_1N_2) \in G$ and $|w_{\texttt{Pre}_G, \mathcal{L}_{C_1}}| + |w_{N_2}| < k$. Since w_N is of the form $w_{C_1}w_{N_2}$, $\texttt{iPos}(w_N, k) = \texttt{iPos}(w_{C_1}, k - |w_{N_2}| + 1)$ (recall that the hole is considered an special constant of size 1). Furthermore, $w_N|_{\texttt{iPos}(w_N,k)} = w_{C_1}|_{\texttt{iPos}(w_N,k)}$. All together implies that $w_N|_{\texttt{iPos}(w_N,k)} = w_{C_1}|_{\texttt{iPos}(w_{C_1},k-|w_{N_2}|+1)}$, and hence, the fact that $\texttt{kExt}(G, N, k) = \texttt{kExt}(G, C_1, k - |w_{N_2}| + 1)$ generates $w_N|_{\texttt{iPos}(w_N,k)}$ follows from induction hyphotesis.

- If $(N \to A_2) \in G$ *(λ-rule)* then the fact that $\texttt{kExt}(G, N, k)$ generates $w_N|_{\texttt{iPos}(w_N,k)}$ directly follows by induction hypothesis.

To show that $\texttt{kExt}(G, N, k)$ contains $\mathcal{O}(\texttt{depth}(G))$ new nonterminals not in G it suffices to remark that the number of recursive calls in the computation of $\texttt{kExt}(G, N, k)$ is bounded by $\texttt{depth}(G)$ and each of them extends G with at most one new nonterminal.

To compute $\texttt{kExt}(G, N, k)$ in linear time we first build the SCFG $\texttt{Pre}_G$ generating the preorder traversals of the terms generated by G and precompute the size of the term/word generated by each nonterminal in G and $\texttt{Pre}_G$. Both operations can be done in linear time. Once this precomputations are done, $\texttt{kExt}(G, N, k)$ can be computed by a single run over the rules of G, which leads to the desired time complexity. □

Computing subcontexts

In this section we present operations to, given an STG G, a term nonterminal A of G, and a position $p \in \mathsf{Pos}(w_A)$, extend G with new nonterminals to generate the prefix context of A with hole path p.

As a first ingredient, we show how to succinctly represent the hole path of a compressed context.

Definition 3.2.24 *Let G be an STG. We define the SCFG H_G representing the hole paths of w_C for all context nonterminals C as follows. For each context nonterminal C of G we construct a nonterminal H_C of H_G. For each natural number i between 1 and the maximum arity of the signature Σ, we construct a nonterminal H_i representing the position i. For each rule with a context nonterminal C as left-hand side, we construct one rule of H_G, depending on the form of the rule of C in G, as follows.*

- *If $(C \to \bullet) \in G$, then H_G contains the rule $H_C \to \lambda$.*
- *If $(C \to C_1C_2) \in G$, then H_G contains the rule $H_C \to H_{C_1}H_{C_2}$.*
- *If $(C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)) \in G$, then H_G contains the rules $H_C \to H_iH_{C_i}$.*

Moreover, for each H_i, H_G contains the rule $H_i \to i$.

Lemma 3.2.25 *The SCFG H_G can be computed for an STG G in time $\mathcal{O}(|G|)$. Let $C \in G$ be any context nonterminal. Then the corresponding nonterminal $H_C \in H_G$ generates $\mathtt{hp}(w_C)$. Moreover, $|H_G| \leq |G| + M$ and $\mathtt{depth}(H_C) \leq \mathtt{depth}(C)$, where M is the maximum arity of the signature.*

Proof. It is easy to prove that $w_{H_C} = \mathtt{hp}(w_C)$ as well as $\mathtt{depth}(H_C) \leq \mathtt{depth}(C)$ using induction on $\mathtt{depth}(C)$. Moreover, from every rule of G we produce one rule of H_G, and for every i between 1 and M we produce one rule of H_G, which leads to a linear time algorithm with respect to $|G|$. □

The following two definitions and lemmas present efficient algorithms to compute prefixes and suffixes of a compressed context.

Definition 3.2.26 *Let G be an STG describing first-order terms and contexts, let C be a context nonterminal of G, and let l be a natural number such that $l \leq |\mathtt{hp}(w_C)|$. We define the extension $\mathtt{Pref}(G, C, l)$ of G representing a prefix of w_C recursively as follows, where C' will be a context nonterminal representing the prefix of w_C with hole depth l.*

- *If $l = 0$, then $\mathtt{Pref}(G, C, l)$ contains G plus the rule $C' \to \bullet$, where C' is a new context nonterminal. In the next cases we assume $l > 0$.*

- *If $l = |\mathtt{hp}(w_C)|$, then $\mathtt{Pref}(G, C, l) = G$ and $C' = C$. In the next cases we assume $l < |\mathtt{hp}(w_C)|$.*

- *If $(C \to C_1C_2) \in G$ and $l \geq |\mathtt{hp}(w_{C_1})|$. Then $\mathtt{Pref}(G, C, l)$ includes $\mathtt{Pref}(G, C_2, l - |\mathtt{hp}(w_{C_1})|)$, which contains a nonterminal C_2' generating the prefix of w_{C_2} with $|\mathtt{hp}(w_{C_2'})| = l - |\mathtt{hp}(w_{C_1})|$, plus the rule $C' \to C_1C_2'$, where C' is a new context nonterminal.*

- *If $(C \to C_1C_2) \in G$ and $l < |\mathtt{hp}(w_{C_1})|$, then, we define $\mathtt{Pref}(G, C, l)$ as $\mathtt{Pref}(G, C_1, l)$.*

- *If $(C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)) \in G$, then $\mathtt{Pref}(G, C, l)$ includes $\mathtt{Pref}(G, C_i, l - 1)$, which contains a nonterminal C_i' generating the prefix of w_{C_i} with $|\mathtt{hp}(w_{C_i'})| = l - 1$, plus the rule $C' \to f(A_1, \ldots, A_{i-1}, C_i', A_{i+1}, \ldots, A_m)) \in G$, where C' is a new context nonterminal.*

Lemma 3.2.27 *Let G be an STG describing first-order terms and contexts, let C be a context nonterminal of G, and let l be a natural number such that $l \leq |\mathtt{hp}(w_C)|$. Then, $\mathtt{Pref}(G, C, l)$ is an extension of G computable in time $\mathcal{O}(|G|)$. It adds at most $\mathtt{depth}(C)$ nonterminals such that one of them, called C', generates the prefix of w_C satisfying $|\mathtt{hp}(w_{C'})| = l$. Moreover, $\mathtt{depth}(C') \leq \mathtt{depth}(C)$ and $\mathtt{depth}(\mathtt{Pref}(G, C, l)) = \mathtt{depth}(G)$.*

Proof. The correctness of the definition of $\mathtt{Pref}(G, C, l)$, as well as $\mathtt{depth}(C') \leq \mathtt{depth}(C)$ and $\mathtt{depth}(\mathtt{Pref}(G, C, l)) = \mathtt{depth}(G)$ can be easily shown by induction on $\mathtt{depth}(C)$. With respect to time complexity, we first precompute $|\mathtt{hp}(w_C)|$ for each context nonterminal of G, which can be done in linear time thanks to Lemma 3.2.17 and Lemma 3.2.25. Time complexity $\mathcal{O}(|G|)$ follows from the fact that the recursive definition decreases the depth of the involved nonterminal. □

Definition 3.2.28 *Let G be an STG describing first-order terms and contexts, let C be a context nonterminal of G, and l a natural number such that $l \leq |\mathtt{hp}(w_C)|$. We define the extension $\mathtt{Suff}(G, C, l)$ of G representing the suffix of w_C starting at hole depth l by the nonterminal C' as follows:*

- *If $l = 0$, then $\mathtt{Suff}(G, C, l) = G$. In the next cases we assume $l > 0$.*

- *If $l = |\mathtt{hp}(w_C)|$ then $\mathtt{Suff}(G, C, l)$ contains G plus the rule $C' \to \bullet$, where C' is a new context nonterminal. In the next cases we assume $l < |\mathtt{hp}(w_C)|$.*

- *If* $(C \to C_1C_2) \in G$ *and* $l < |\mathtt{hp}(w_{C_1})|$. *Then* $\mathtt{Suff}(G, C, l)$ *includes* $\mathtt{Suff}(G, C_1, l)$, *which contains a context nonterminal* C_1' *generating the suffix of* w_{C_1} *with* $|\mathtt{hp}(w_{C_1'})| = |\mathtt{hp}(w_{C_1})| - l$, *plus the rule* $C' \to C_1'C_2$, *where* C' *is a new context nonterminal.*

- *If* $(C \to C_1C_2) \in G$ *and* $l \geq |\mathtt{hp}(w_{C_1})|$, *then, with* $l' = l - |\mathtt{hp}(w_{C_1})|$, *we define* $\mathtt{Suff}(G, C, l)$ *as* $\mathtt{Suff}(G, C_2, l')$.

- *If* $(C \to f(A_1, \dots, A_{i-1}, C_i, A_{i+1}, \dots, A_m)) \in G$, *then we define* $\mathtt{Suff}(G, C, l)$ *as* $\mathtt{Suff}(G, C_i, l-1)$.

Lemma 3.2.29 *Let* G *be an STG describing first-order terms and contexts. Let* C *be a context nonterminal of* G, *and* l *a natural number such that* $l \leq |\mathtt{hp}(w_C)|$. *Then,* $\mathtt{Suff}(G, C, l)$ *is an extension of* G *computable in time* $\mathcal{O}(|G|)$. *It adds at most* $\mathtt{depth}(C)$ *nonterminals such that one of them, called* C', *generates the suffix of* w_C *satisfying* $|\mathtt{hp}(w_{C'})| = |\mathtt{hp}(w_C)| - l$. *Moreover,* $\mathtt{depth}(C') \leq \mathtt{depth}(C)$ *and* $\mathtt{depth}(\mathtt{Pref}(G, C, l)) = \mathtt{depth}(G)$.

Proof. The proof is analogous to the one of Lemma 3.2.27. □

Finally, we are ready to present our construction to compute a subcontext of a context in the compressed setting.

Definition 3.2.30 *Let* G *be an STG generating terms and contexts, let* A *be a term nonterminal of* G, *and let* p *be a position in* w_A. *Then, we recursively define* $\mathtt{pCon}(G, A, p)$ *as an extension of* G *representing the prefix context of* A *with hole path* p *as follows.*

- *if* $A \to \alpha(A_1, \dots, A_m) \in G$ *and* $p = i \cdot p'$ *then* $\mathtt{pCon}(G, A, p)$ *includes* $\mathtt{pCon}(G, A_i, p')$, *which contains a nonterminal* C_i *generating the context prefix of* w_{A_i} *with* $\mathtt{hp}(w_{C_i}) = p'$, *plus the rule* $C' \to \alpha(A_1, \dots, C_i, \dots, A_m)$, *where* C' *is a new context nonterminal.*

- *If* $A \to A'$, *then* $\mathtt{pCon}(G, A, p) = \mathtt{pCon}(G, A', p)$.

- *if* $A \to C_1A_2 \in G$ *then* $p = p_1 \cdot p_2$ *where* p_1 *is the maximal common prefix of* p *and* $\mathtt{hp}(C_1)$. *We distinguish three cases:*

  - *If* $p_1 = \mathtt{hp}(C_1)$ *then* $\mathtt{pCon}(G, A, p)$ *includes* $\mathtt{pCon}(G, A_2, p_2)$, *which contains a nonterminal* C_2 *generating the context prefix of* w_{A_2} *with* $\mathtt{hp}(w_{C_2}) = p_2$, *plus the rule* $C' \to C_1C_2$, *where* C' *is a new context nonterminal.*

  - *If* $p_1 \prec \mathtt{hp}(C_1)$ *and* $p_2 = \lambda$ *then* $\mathtt{pCon}(G, A, p)$ *is defined as* $\mathtt{Pref}(C_1, G, |p_1|)$.

- *(see Fig. 3.4) If* $p_1 \prec \mathtt{hp}(C_1)$ *and* $p_2 \neq \lambda$ *then* p *is of the form* $p_1 \cdot i \cdot p_3$ *and* $\mathtt{hp}(C_1)$ *is of the form* $p_1 \cdot k \cdot p_4$*, for some positions* p_3 *and* p_4*, and some integers* i *and* k *satisfying* $i \neq k$*. We assume* $i < k$*, without loss of generality. Let* l_1 *and* l_4 *be* $|p_1|$ *and* $|p_4|$*, respectively.*

  Let G_1 *be* $\mathtt{Pref}(G, C_1, l_1)$*. The STG* G_1 *contains a context nonterminal* C_{11} *generating the prefix of* w_{C_1} *such that* $|\mathtt{hp}(C_{11})| = l_1$*. Let* G_2 *be* $\mathtt{Suff}(G_1, C_1, |\mathtt{hp(C_1)}| - l_4)$*. The STG* G_2 *contains a context nonterminal* C_{12} *generating the suffix of* w_{C_1} *such that* $|\mathtt{hp}(C_{12})| = l_4$*.*

  Let G'_1 *be* $\mathtt{Suff}(G, C_1, l_1)$*. The STG* G'_1 *contains a context nonterminal* C'_{12} *generating the suffix of* w_{C_1} *such that* $|\mathtt{hp}(C'_{12})| = |\mathtt{hp(C_1)}| - l_1 = l_4 + 1$*. Let* G'_2 *be* $\mathtt{Pref}(G'_1, C'_{12}, 1)$*. The STG* G'_2 *contains a context nonterminal* C'_{11} *generating the prefix of* $w_{C'_{12}}$ *such that* $|\mathtt{hp}(C'_{11})| = 1$*.*

  At this point, note that $w_{G,C_1} = w_{G_2,C_{11}} w_{G'_2,C'_{11}} w_{G_2,C_{12}}$*. Moreover, the rule of* C'_{11} *in* G'_2 *is of the form* $C'_{11} \to \alpha(A'_1, \ldots, A'_{k-1}, C''_{11}, A'_{k+1}, \ldots, A'_m)$*, where all the* A'_i *are term nonterminals of the original* G*, and generating the same terms as in* G*. Moreover, the rule of* C''_{11} *in* G'_2 *is necessarily* $C''_{11} \to \bullet$*.*

  We define $\mathtt{pCon}(G, A, p)$ *as* $\mathtt{pCon}(G_2, A'_i, p_3)$*, which contains a context nonterminal* C_3 *generating the prefix context of* $w_{A'_i}$ *with hole at position* p_3 *plus the rules* $C' \to C_{11}C_4$*,* $C_4 \to \alpha(A'_1, \ldots, A'_{i-1}, C_3, A'_{i+1}, \ldots, A'_{k-1}, A'_k, A'_{k+1}, \ldots, A_m)$*,* $A'_k \to C_{12}A_2$*, where* C', C_4, A'_k *are new nonterminals.*

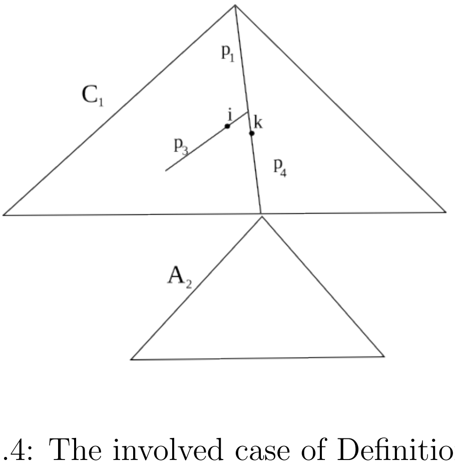


Figure 3.4: The involved case of Definition 3.2.30

Lemma 3.2.31 *Let G be an STG describing terms and contexts, let A be a nonterminal of G, and let p be a position in w_A. Then,* $\texttt{pCon}(G, A, p)$ *contains at most* $\texttt{depth}(A) * (2\texttt{depth}(A) + 1)$ *new nonterminals such that one of them, called C', generates the context prefix of w_A with* $\texttt{hp}(w_{C'}) = p$*. Moreover,* $\texttt{depth}(C') \leq \texttt{depth}(A)$*.*

Proof. The fact that $\texttt{pCon}(G, A, p)$ contains a context nonterminal C' generating the context prefix of w_A with $\texttt{hp}(w_{C'}) = p$ can be verified by induction on $\texttt{depth}(A)$ and distinguishing cases according to the definition of $\texttt{pCon}(G, A, p)$. As in previous constructions, $\texttt{pCon}(G, A, p)$ can be computed in a single run over the rules of the G. To show the upper bound on the size of the computed extension, it suffices to note that the worst case in this sense is when the rule of A is of the form $A \to C_1 A_2$ and $\texttt{hp}(w_{C_1})$ and p are disjoint. In such a case, we add 3 new rules plus the new rules in $\texttt{Pref}(G, C_1, |p_1|)$ and $\texttt{Suff}(G, C_1, |p_1| + 1)$, where p_1 is the maximal common prefix between $\texttt{hp}(C_1)$ and p. The number of added nonterminals is bounded by $\texttt{depth}(A) - 1$ for both the $\texttt{Pref}$ and the $\texttt{Suff}$ constructions by Lemma 3.2.27, and Lemma 3.2.29, respectively. The fact $\texttt{depth}(C') \leq \texttt{depth}(A)$ can be verified by induction on $\texttt{depth}(A)$ and using Lemma 3.2.27, and Lemma 3.2.29 as follows: $\texttt{depth}(A'_j) \leq \texttt{depth}(A) - 2$, hence $\texttt{depth}(C_3) \leq \texttt{depth}(A) - 2$. Also, $\texttt{depth}(C_4) \leq \texttt{depth}(A) - 1$ and $\texttt{depth}(C_{11}) \leq \texttt{depth}(A) - 1$, hence $\texttt{depth}(C') \leq \texttt{depth}(A)$. □

Chapter 4

Compressed First-Order Unification and matching

In this chapter we present polynomial time algorithms for the first-order unification and matching problems when their input is represented using STGs. We will use some of the constructions introduced in Section 3.2.1 of the previous chapter. Moreover, at the end of the chapter, we present an experimental evaluation of different implementations of our algorithms.

4.1 First-order Unification and matching with STGs

First of all, let us state our problem formally.

Definition 4.1.1 *An instance of the* first-order unification problem with STGs *is a triple $\langle G, A_s, A_t\rangle$, where G is an STG G representing first-order terms, and A_s, A_t are term nonterminals terms $s = w_{G,A_s}$ and $t = w_{G,A_t}$. The problem consists in deciding whether s and t are unifiable. In the affirmative case, its computational version asks for a representation of the most general unifier.*

Our algorithm generates the most general unifier in polynomial time and represented again with an STG.

4.1.1 Outline of the algorithm

From a high level perspective the structure of our algorithm, described in Figure 4.1, is very simple and rather standard: it is based on Robinson

```
Input:  An STG $G$ and term nonterminals $A_s$ and $A_t$.
          (we write $s$ and $t$ for $w_{A_s}$ and $w_{A_t}$, respectively).
While $s$ and $t$ are different do:
   Look for the first position $k$ such that $\mathtt{pre}(s)[k] \neq \mathtt{pre}(t)[k]$.
   If both $\mathtt{pre}(s)[k]$ and $\mathtt{pre}(t)[k]$ are function symbols, Then
      Halt stating that the initial $s$ and $t$ are not unifiable
   // Here, either $\mathtt{pre}(s)[k]$ or $\mathtt{pre}(t)[k]$, say $\mathtt{pre}(s)[k]$, is a variable $x$.
   If $x$ occurs in $t|_p$, where $p = \mathtt{iPos}(t,k)$, Then
      Halt stating that the initial $s$ and $t$ are not unifiable
   Extend $G$ by the assignment $\{x \mapsto t|_p\}$
EndWhile
Halt stating that the initial $s$ and $t$ are unifiable
```

Figure 4.1: Unification Algorithm of STG-Compressed Terms

unification algorithm [Rob65]. Many algorithms for first-order unification are variants of this scheme. In those algorithms, terms are usually represented with Directed Acyclic Graphs (Dags), implemented somehow, in order to avoid the space explosion due to the repeated instantiation of variables by terms. For example, in the Martelli-Montanari algorithm [MM82, BS01] instantiations are represented by equations. Note that, as commented in Section 2.3, our algorithm is just an adaptation of the algorithm defined in Figure 2.1 to the case where the inputted terms are compressed using STGs. For this reason we do not argue about correctness and just focus in proving that its running time is polynomial. Hence, we need to perform all the operations mentioned in the algorithm of Figure 4.1 on the compressed representation of terms. More concretely, given an STG G as a compressed representation of two terms s and t, we need to compute a smallest index k in which $\mathtt{pre}(s)$ and $\mathtt{pre}(t)$ differ. At this point, if both $\mathtt{pre}(s)[k]$ and $\mathtt{pre}(t)[k]$ are function symbols, we terminate stating nonunifiability. Otherwise, either $\mathtt{pre}(s)$ or $\mathtt{pre}(t)$, say $\mathtt{pre}(s)$, contains a variable x at k. Note that, since the arity of the terminals in G is fixed, the index k corresponds to a unique position $p \in \mathtt{Pos}(s) \cap \mathtt{Pos}(t)$, as explained in Chapter 2. If x properly occurs in the subterm of t at p, then we terminate, again stating nonunifiability. Otherwise, we replace x by the subterm of t at p everywhere, and repeat the process until both s and t become equal, in which case we state unifiability.

In Chapter 3 we showed how to efficiently perform some of the required operations on STGs: Decide whether s and t are equal, generate a compressed representation for $\mathtt{pre}(s)$ and $\mathtt{pre}(t)$, look for the smallest index k such that $\mathtt{pre}(s)[k] \neq \mathtt{pre}(s)[k]$, and construct the term $t|_p$, where $p = \mathtt{iPos}(t,k)$. However, we still need to see how to deal with instantiations of variables in the STG setting. Performing a replacement of a first-order variable x by a term u is easily representable with STGs by simply transforming x into a nonterminal of the grammar and adding rules such that x generates u. However, since successive replacements of variables by subterms modify the initial terms, we have to show that this does not produce an exponential increase of the size of the grammar, since its depth may be doubled after each of these operations. To this end, in the following section, we develop a notion of restricted depth, and show that its value is polynomial with respect to the size of the input grammar, its value is preserved along the execution, and the size increase after each instantiation can be bounded by the value of the restricted depth.

4.1.2 Application of substitutions and a notion of restricted depth

Term unification algorithms usually apply substitutions when one variable is isolated. We need to emulate such applications when the terms are represented with STGs. In an STG, first-order variables are terminals of arity 0. Replacing a first-order variable X can be emulated by transforming X into a term nonterminal and adding the necessary rules for making X generate the replacing value. We define this notion of application of a substitution as follows.

Definition 4.1.2 *Let G be an STG. Let X be a terminal representing a first-order variable and let A be a term nonterminal of G, respectively. Then, $\{X \mapsto A\}(G)$ is defined as the STG obtained by adding the rule $X \to A$ to G, and converting X into a term nonterminal.*

When one or more substitutions of this form are applied, in general the depth of the nonterminals of G might increase. In order to see that the size increase is polynomially bounded along several substitution operations when unifying, we need a new notion of depth called `Vdepth`, which does not increase after an application of a substitution. It allows us to bound the final size increase of G. The notion of `Vdepth` is similar to the notion of `depth`, but it is 0 for the nonterminals N belonging to a special set V satisfying the following condition.

Definition 4.1.3 *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ be an STG, and let V be a subset of $\mathcal{TN} \cup \Sigma$. We say that V is a* λ-set *for G if for each term nonterminal A in V, the rule of G of the form $A \to \alpha$ is a λ-rule.*

Definition 4.1.4 *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ be an STG and let V be a λ-set for G. For every nonterminal N of G, the value* $\mathtt{Vdepth}_{G,V}(N)$*, denoted also as* $\mathtt{Vdepth}_V(N)$ *or* $\mathtt{Vdepth}(N)$ *when G and/or V are clear from the context, is defined as follows (recall the convention that* $\mathtt{max}(\emptyset) = 0$*).*

$$\begin{array}{lcl} \mathtt{Vdepth}(N) & := & 0 \textit{ for } N \in V \\ \mathtt{Vdepth}(N) & := & 1 + \textit{max}\{\mathtt{Vdepth}(N') \mid N' \textit{ is a nonterminal occurring in } \alpha, \\ & & \qquad \textit{where } N \to \alpha \in G\}, \textit{ otherwise.} \end{array}$$

The $\mathtt{Vdepth}$ *of G is the maximum of the* $\mathtt{Vdepth}$ *of its nonterminals.*

The idea is to make V to contain all first-order variables, before and after converting them into term nonterminals. The following lemma is completely straightforward from the above definitions, and states that a substitution application does not modify the $\mathtt{Vdepth}$ provided $X \in V$ for the substitution $X \mapsto A$.

Lemma 4.1.5 *Let G, V be as in the above definition. Let $X \in V$ be a terminal of G of arity 0, and let A be a term nonterminal of G. Let G' be $\{X \mapsto A\}(G)$. Then, for any nonterminal N of G it holds that* $\mathtt{Vdepth}_{G'}(N) = \mathtt{Vdepth}_G(N)$*.*

We also need the fact that $\mathtt{Vdepth}$ does not increase due to the construction of $\mathtt{kext}(G, A, k)$ from G. However, we first prove a more specific statement.

Lemma 4.1.6 *Let G be an STG, let C be a context nonterminal of G, let V be a λ-set for G, let k be a natural number such that $w_C|_{\mathtt{iPos}(w_C,k)}$ is a context, and let G' be* $\mathtt{kext}(G, C, k)$*.*

Then, for every nonterminal N of G it holds that $\mathtt{Vdepth}_G(N) = \mathtt{Vdepth}_{G'}(N)$*, and for every new nonterminal N' in G' and not in G, it holds that* $\mathtt{Vdepth}_{G'}(N') \leq \mathtt{Vdepth}_G(C)$*. Moreover, the number of new added nonterminals is bounded by* $\mathtt{Vdepth}_G(C)$*.*

Proof. The identity $\mathtt{Vdepth}_G(N) = \mathtt{Vdepth}_{G'}(N)$ for each nonterminal N of G is straightforward from the fact that $\mathtt{kext}(G, C, k)$ does not change the rules for the nonterminals occurring in G. To prove the fact that $\mathtt{Vdepth}_{G'}(N') \leq \mathtt{Vdepth}_G(C)$ for each new nonterminal N' in G' and not in G, plus the fact that at most $\mathtt{Vdepth}_G(C)$ new nonterminals have been added, we will use induction on $\mathtt{Vdepth}(C)$. The base case ($\mathtt{Vdepth}(C) = 1$) trivially holds since, in this case, the STG G is not modified. For the induction step we distinguish cases according to the definition of $\mathtt{kExt}(G, C, k)$:

- Assume that $(C \rightarrow f(A_1, \ldots, A_{i-1}, C', \ldots, A_m)) \in G$. Note that, since $w_C|_{\mathtt{iPos}(w_C,k)}$ is a context, it holds that $1 + |w_{A_1}| + \ldots + |w_{A_{i-1}}| = k' < k \leq k' + |w_{C'}|$. In this case, $\mathtt{kext}(G, C, k) = \mathtt{kext}(G, C', k - k')$ and, since $\mathtt{Vdepth}(C') < \mathtt{Vdepth}(C)$, the lemma directly follows by induction hypothesis.

- Assume that $(C \rightarrow C_1C_2) \in G$ and $k \leq |w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}|$. In this case, the construction of $\mathtt{kext}(G, C, k)$ is done by computing $\mathtt{kext}(G, C_1, k)$ and adding the rule $C' \rightarrow C'_1C_2$, where C'_1 is the context nonterminal generating $w_{C_1}|_{\mathtt{iPos}(w_{C_1},k)}$ and C' is an additional new nonterminal. Since $\mathtt{Vdepth}(C_1) < \mathtt{Vdepth}(C)$, by induction hypothesis, it holds that for all the new nonterminals N' in $G' = \mathtt{kext}(G, C_1, k)$, $\mathtt{Vdepth}_{G'}(N') \leq \mathtt{Vdepth}_G(C_1)$ and at most $\mathtt{Vdepth}_G(C_1)$ new nonterminals have been added. It follows that at most $\mathtt{Vdepth}_G(C)$ new nonterminals have been added in the construction of $\mathtt{kext}(G, C, k)$, and $\mathtt{Vdepth}_{G'}(C'_1) \leq \mathtt{Vdepth}_G(C_1)$. Moreover, since $\mathtt{Vdepth}_G(C) = 1 + \mathtt{max}(\mathtt{Vdepth}_G(C_1), \mathtt{Vdepth}_G(C_2))$, $\mathtt{Vdepth}_{G'}(C') = 1 + \mathtt{max}(\mathtt{Vdepth}_{G'}(C'_1), \mathtt{Vdepth}_{G'}(C_2))$ and $\mathtt{Vdepth}_G(C_2) = \mathtt{Vdepth}_{G'}(C_2)$, it also holds that $\mathtt{Vdepth}_{G'}(C') \leq \mathtt{Vdepth}_G(C)$.

- Assume that $(C \rightarrow C_1C_2) \in G$ and $k' = |w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}| < k \leq |w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}| + |w_{C_2}|$. In this case, $\mathtt{kext}(G, C, k) = \mathtt{kext}(G, C_2, k - k')$ and, since $\mathtt{Vdepth}(C_2) < \mathtt{Vdepth}(C)$, the lemma directly follows by induction hypothesis.

  Finally, note that the case $(C \rightarrow C_1C_2) \in G$ and $|w_{\mathtt{Pre}_{G,\mathcal{L}_{C_1}}}| + |w_{C_2}| < k$ is not possible due to the assumption that $w_C|_{\mathtt{iPos}(w_C,k)}$ is a context.

□

Lemma 4.1.7 *Let G be an STG, let N be a nonterminal of G, let V be a λ-set for G, let k be a natural number satisfying $k \leq |\mathtt{Pre}(w_{G,N})|$, and let G' be $\mathtt{kext}(G, N, k)$.*

Then, for every nonterminal N' of G it holds that $\mathtt{Vdepth}_G(N') = \mathtt{Vdepth}_{G'}(N')$, and for every new nonterminal N'' in G' and not in G, it holds that $\mathtt{Vdepth}_{G'}(N'') \leq \mathtt{Vdepth}(G)$. Moreover, the number of new added nonterminals is bounded by $\mathtt{Vdepth}(G)$.

Proof. The identity $\mathtt{Vdepth}_G(N'') = \mathtt{Vdepth}_{G'}(N'')$ for each nonterminal N'' of G is straightforward from the fact that $\mathtt{kext}(G, N, k)$ does not change the rules for the nonterminals occurring in G. We will prove the fact that $\mathtt{Vdepth}_{G'}(N'') \leq \mathtt{Vdepth}(G)$ for each new nonterminal N'' in G' and not in G, plus the fact that at most $\mathtt{Vdepth}(G)$ new nonterminals have been added by induction on $\mathtt{depth}_G(N)$. The base case ($\mathtt{depth}(N) = 1$) trivially holds since, in this case, the STG G is not modified. For the induction step we distinguish cases according to the definition of $\mathtt{kExt}(G, N, k)$. The only interesting cases are either when $(N \to C_1A_2) \in G$ and $k \leq |w_{\mathtt{Pre}_G,\mathcal{L}_{C_1}}|$ or $(N \to C_1C_2) \in G$ and $k \leq |w_{\mathtt{Pre}_G,\mathcal{L}_{C_1}}|$. Note that these are the only cases in which the grammar might be extended with new nonterminals after the recursive call. We will solve the first one, the other is solved analogously.

Hence, assume that $(N \to C_1A_2) \in G$ and $k \leq |w_{\mathtt{Pre}_G,\mathcal{L}_{C_1}}|$. In this case the nonterminal N' in $\mathtt{kext}(G, C_1, k)$ generating the subterm of w_{G,C_1} at position $\mathtt{iPos}(w_{G,C_1}, k)$ is a either a term nonterminal or a context nonterminal. We will solve the two cases separately. First assume that N' is a term nonterminal. In this case $\mathtt{kext}(G, N, k)$ is constructed as $\mathtt{kext}(G, C_1, k)$. Since $\mathtt{Vdepth}(C_1) < \mathtt{Vdepth}(N)$, the lemma holds by induction hypothesis in this case. On the other hand, if N' is a context nonterminal, the construction of $\mathtt{kext}(G, N, k)$ is done by computing $\mathtt{kext}(G, C_1, k)$ and adding the rule $A \to N'A_2$, where A is an additional new term nonterminal. By Lemma 4.1.6, for all the new nonterminals N'' in $\mathtt{kext}(G, C_1, k)$ and not in G, $\mathtt{Vdepth}_{G'}(N'') \leq \mathtt{Vdepth}_G(C_1)$. Moreover, the number of new added nonterminals is bounded by $\mathtt{Vdepth}_G(C_1)$. Hence, $\mathtt{Vdepth}_{G'}(N') \leq \mathtt{Vdepth}_G(C_1)$ and, since $\mathtt{Vdepth}(C_1) < \mathtt{Vdepth}(N)$, at most $\mathtt{Vdepth}_G(N) \leq \mathtt{Vdepth}(G)$ new nonterminals have been added in the construction of $\mathtt{kext}(G, N, k)$. Furthermore, since $\mathtt{Vdepth}_G(N) = 1 + \mathtt{max}(\mathtt{Vdepth}_G(C_1), \mathtt{Vdepth}_G(A_2))$, $\mathtt{Vdepth}_{G'}(A) = 1 + \mathtt{max}(\mathtt{Vdepth}_{G'}(N'), \mathtt{Vdepth}_{G'}(A_2))$ and $\mathtt{Vdepth}_G(A_2) = \mathtt{Vdepth}_{G'}(A_2)$, it also holds that $\mathtt{Vdepth}_{G'}(A) \leq \mathtt{Vdepth}_G(N) \leq \mathtt{Vdepth}(G)$. □

Now our algorithm is completely defined. Before arguing about its running time, let us show a complete execution example.

Example of execution

Let $G = (\{A_t, A_s, A, B_1, B_2, A_x, B_y\}, \{C_0, C_1, C_2, C_3, C_4, C_., D\}$, $\{g, f, a, x\}, R)$, where $R = \{A_t \to g(B_1, A), A_s \to g(B_2, A), A \to C_4A_a, C_4 \to C_3C_3, C_3 \to C_2C_2, C_2 \to C_1C_1, C_1 \to C_0C_0, C_0 \to f(C_.), C_. \to \bullet, A_a \to a, D \to C_3C_2, B_1 \to DB_x, B_2 \to C_4B_y, B_x \to x, B_y \to y\}$, be an STG. Note that $w_{G,A_t} = g(f^{12}(x), f^{16}(a))$, and $w_{G,A_s} = g(f^{16}(y), f^{16}(a))$. Hence, $\langle G, A_s \doteq A_t\rangle$ is an instance of first-order unification with STG. The goal is to find a substitution σ such that $\sigma(w_{G,A_s}) = \sigma(w_{G,A_t})$.

The set of rules of the SCFG $\mathtt{Pre}_G$ obtained by applying the rules of Figure 3.2. to G is

$\{\mathcal{P}_{A_t} \to g\mathcal{P}_{B_1}\mathcal{P}_A, \mathcal{P}_{A_s} \to g\mathcal{P}_{B_2}\mathcal{P}_A, \mathcal{P}_A \to \mathcal{L}_{C_4}\mathcal{P}_{A_a}\mathcal{R}_{C_4}, \mathcal{P}_{A_a} \to$

$a, \mathcal{P}_{B_1} \rightarrow \mathcal{L}_D\mathcal{P}_{B_x}\mathcal{R}_D, \mathcal{L}_D \rightarrow \mathcal{L}_{C_3}\mathcal{L}_{C_2}, \mathcal{R}_D \rightarrow \mathcal{R}_{C_2}\mathcal{R}_{C_3}, \mathcal{P}_{B_x} \rightarrow x, \mathcal{P}_{B_2} \rightarrow \mathcal{L}_{C_4}\mathcal{P}_{B_y}\mathcal{R}_{C_4}, \mathcal{L}_{C_4} \rightarrow \mathcal{L}_{C_3}\mathcal{L}_{C_3}, \mathcal{L}_{C_3} \rightarrow \mathcal{L}_{C_2}\mathcal{L}_{C_2}, \mathcal{L}_{C_2} \rightarrow \mathcal{L}_{C_1}\mathcal{L}_{C_1}, \mathcal{L}_{C_1} \rightarrow \mathcal{L}_{C_0}\mathcal{L}_{C_0}, \mathcal{L}_{C_0} \rightarrow f\mathcal{L}_{C_.}, \mathcal{L}_{C_.} \rightarrow \lambda, \mathcal{R}_{C_.} \rightarrow \lambda, \mathcal{R}_{C_0} \rightarrow \mathcal{R}_{C_.}, \mathcal{R}_{C_1} \rightarrow \mathcal{R}_{C_0}\mathcal{R}_{C_0}, \mathcal{R}_{C_2} \rightarrow \mathcal{R}_{C_1}\mathcal{R}_{C_1}, \mathcal{R}_{C_3} \rightarrow \mathcal{R}_{C_2}\mathcal{R}_{C_2}, \mathcal{R}_{C_4} \rightarrow \mathcal{R}_{C_3}\mathcal{R}_{C_3}, \mathcal{P}_{B_y} \rightarrow y, \}$. Note that $w_{\mathcal{P}_{A_t}} = gf^{12}xf^{16}a$ and $w_{\mathcal{P}_{A_s}} = gf^{16}af^{16}a$.

The SCFG $\texttt{Pre}_G$ is not in Chomsky Normal Form, but it is easy to adapt the algorithm of Figure 3.3 to this case. Thus, if we execute an adapted version of $\texttt{index}(\mathcal{P}_{A_t}$,$\mathcal{P}_{A_s}$,0,$\texttt{Pre}_G)$, the following sequence of calls is produced: $\texttt{index}(\mathcal{P}_{A_t}, \mathcal{P}_{A_s}, 0, \texttt{Pre}_G)$, $\texttt{index}(\mathcal{P}_{B_1}, \mathcal{P}_{A_s}, 1, \texttt{Pre}_G)$, $\texttt{index}(\mathcal{P}_{B_x}, \mathcal{P}_{A_s}, 13, \texttt{Pre}_G)$. The third call returns 1, the second one returns 13, and the first one returns 14, which corresponds to the first different position of $w_{\mathcal{P}_{A_s}}$ and $w_{\mathcal{P}_{A_t}}$.

Note that $\texttt{iPos}(w_{G,A_s}, 14) = 1^{13}$. We compute now and extension $\texttt{kExt}(G, A_s, 14)$ of G, as described in Definition 3.2.22, such that a new term nonterminal A'_s generates $w_{A_s}|_{1^{13}}$. We obtain the following set of rules, where rules in bold correspond to the added nonterminals due to the $\texttt{kext}$ constructions w.r.t to the STG G given as input: $\{\mathbf{A'_s} \rightarrow \mathbf{C_2B_y}, A_t \rightarrow g(B_1, A), A_s \rightarrow g(B_2, A), A \rightarrow C_4A_a, C_4 \rightarrow C_3C_3, C_3 \rightarrow C_2C_2, C_2 \rightarrow C_1C_1, C_1 \rightarrow C_0C_0, C_0 \rightarrow f(C_.), C_. \rightarrow \bullet, A_a \rightarrow a, D \rightarrow C_3C_2, B_1 \rightarrow DB_x, B_2 \rightarrow C_4B_y, B_x \rightarrow x, B_y \rightarrow y\}$.

Note that, in the extended grammar, $w_{A'_s} = w_{A_s}|_{\texttt{iPos}(w_{G,A_s},14)} = w_{A_s}|_{1^{13}} = f^4(y)$. Then, we need to check that the variable x does not occur in $w_{A'_s}$, which can be done in linear time as shown in Lemma 3.2.18. Finally, we perform the substitution $\{x \mapsto A'_s\}(G)$ by converting x into a nonterminal of the grammar generating $w_{A'_s}$ as stated in Definition 4.1.2. The set of rules of the obtained grammar G' after the $\texttt{kext}$ construction and this assignment is $\{\mathbf{x} \rightarrow \mathbf{A'_s}, \mathbf{A'_s} \rightarrow \mathbf{C_2B_y}, A_t \rightarrow g(B_1, A), A_s \rightarrow g(B_2, A), A \rightarrow C_4A_a, C_4 \rightarrow C_3C_3, C_3 \rightarrow C_2C_2, C_2 \rightarrow C_1C_1, C_1 \rightarrow C_0C_0, C_0 \rightarrow f(C_.), C_. \rightarrow \bullet, A_a \rightarrow a, D \rightarrow C_3C_2, B_1 \rightarrow DB_x, B_2 \rightarrow C_4B_y, B_x \rightarrow x, B_y \rightarrow y\}$.

Note that $w_{G',A'_s} = w_{G,A_s}|_{\texttt{iPos}(w_{G,A_s},14)} = w_{G,A_s}|_{1^{13}} = f^4(y)$, and thus, $w_{G',A_t} = g(f^{12}(w_{G',x}), f^{16}(a)) = g(f^{12}(w_{G',A'_s}), f^{16}(a)) = g(f^{12}f^4(y), f^{16}(a)) = g(f^{16}(y), f^{16}(a)) = w_{G',A_s}$. Hence, we state unifiability. The solution σ is represented in the STG G' as $\sigma(x) = w_{G',x}$.

4.1.3 Run time analysis

The algorithm runs in polynomial time due to the following observations. Let n and m be the initial value of $\texttt{depth}(G)$ and $|G|$, respectively. We define V to be the set of all the first-order variables at the start of the execution (before any of them has been converted into a nonterminal). Hence, at this point $\texttt{Vdepth}(G) = n$. The value $\texttt{Vdepth}(G)$ is preserved to be n along the execution of the algorithm thanks to Lemmas 4.1.5 and 4.1.7. Moreover, by Lemma 4.1.7, at most n new nonterminals are added at each step. Since at most $|V|$ steps are executed, the final size of G is bounded by $m+|V|n$. Each execution step takes time at most $\mathcal{O}(|G|^3)$. Thus we have proved:

Theorem 4.1.8 *First-order unification of two terms represented by an STG can be done in polynomial time ($\mathcal{O}(|V|(m+|V|n)^3)$, where m represents the size of the input STG, n represents the depth, and V represents the set of different first-order variables occurring in the input terms). This holds for the decision question, as well as for the computation of the most general unifier, whose components are represented by the final STG.*

4.2 First-order matching with STGs

Theorem 4.1.8 already provides a polynomial matching algorithm on STG-compressed terms. However, we will describe a matching algorithm that exploits the specific restrictions of matching and show that its worst case running time is better than the one of the algorithm for the unification case. First of all, let us define the problem properly.

Definition 4.2.1 *An instance of the* first-order matching problem with STG's *is a triple $\langle G, A_s, A_t\rangle$, where G is an STG G representing first-order terms, and A_s, A_t are term nonterminals representing terms $s = w_{G,A_s}$ and $t = w_{G,A_t}$, where t is ground. The problem consists in deciding whether s and t are unifiable. In the affirmative case, its computational version asks for a representation of the most general unifier.*

First-order matching is a particular case of first-order unification. However, taking advantage of the fact that one of the terms is ground leads to a faster algorithm with respect to the one presented in the previous section. In [GGSS08], an algorithm for first-order matching with STGs running in time $\mathcal{O}(m+n)^4$ was presented, where m represents the size of the input STG, n represents its depth, and V represents the set of different first-order variables occurring in the input terms. The construction presented in Definition 3.2.22 is the key idea to obtain a faster algorithm.

4.2.1 Outline of the algorithm

The structure of our algorithm is sketched in Figure 4.2. Note that, as commented in Section 2.3, our algorithm is just an adaptation of the algorithm defined in Figure 2.2 to the case where the inputted terms are compressed using STGs. hence, the input of the problem consists of an STG G as a compressed representation of two terms s and t. As in the first-order unification case, the algorithm works with compressed representations of the preorder traversal words of the terms s and t to be matched. Hence, we first compute a representation of $\texttt{pre}(s)$ and $\texttt{pre}(t)$. Then we find the index k of the first occurrence of a variable x in $\texttt{pre}(s)$, and, given G and k, compute $t' = t|_{\texttt{iPos}(t,k)}$. If t' is undefined we halt giving a negative answer. Otherwise we apply the substitution $\{x \to t'\}(s)$ and restart the process until all variables are replaced. Finally, let s' be the term obtained from s after all replacements are done. We check whether s' and t are syntactically equal and answer accordingly. Note that, in contrast to unification algorithm, we look for the first occurrence of a variable in $\texttt{pre}(s)$ instead of looking for the first difference between $\texttt{pre}(s)$ and $\texttt{pre}(t)$. This refines the approach used in the previous section for the general case of first-order unification and improves time complexity results in previous work on first-order matching with STGs [GGSS08].

```
Input:  An STG $G$ and term nonterminals $A_s$ and $A_t$.
         (we write $s$ and $t$ for $w_{A_s}$ and $w_{A_t}$
         and $\mathcal{X}$ for the set of variables in $s$).
Repeat $|\mathcal{X}|$ times:
   Look for the smallest index $k$ such that $\texttt{pre}(s)[k] = x \in \mathcal{X}$.
   If $\texttt{iPos}(t,k)$ is undefined Then Halt stating that the initial $s$ and $t$ do not match.
   Extend $G$ by the assignment $\{x \mapsto t|_p\}$, where $p = \texttt{iPos}(t,k)$.
EndRepeat
   If $s = t$ Then Halt stating that the initial $s$ and $t$ match.
   Else Halt stating that the initial $s$ and $t$ do not match.
```

Figure 4.2: Matching Algorithm for STG-Compressed Terms

In the previous section we already showed how to compute a succinct representation of $\texttt{pre}(s)$ and $\texttt{pre}(t)$, to compute, given a natural number k, the subterm of a term t at position $\texttt{iPos}(t,k)$, and to apply a substitution. Hence, it only remains to show how to compute k, the index of the first occurrence of a variable in $\texttt{pre}(s)$.

4.2.2 Finding the first occurrence of a variable

The task of finding the index of the first occurrence of a variable in a compressed word can be performed efficiently as stated in the following Lemma.

Lemma 4.2.2 *Let P be an SCFG, and let p be a nonterminal of P representing the preorder traversal word of a first-order term. Then, the smallest index k such that $w_p[k]$ is a terminal and a variable can be computed in time $\mathcal{O}(|P|)$.*

Proof. Let $\mathcal{X}$ denote the set of first-order variables. We define $k = \mathtt{index}(p, P)$ as follows:

$$\mathtt{index}(p,P) = \begin{cases} 1 & \text{, if } p \to \alpha \in P \wedge \alpha \in \mathcal{X} \\ \mathtt{index}(p_1,P) & \text{, if } (p \to p_1p_2) \in P \wedge \\ & \quad \exists x \in \mathcal{X} : x \text{ occurs in } w_{P,p_1} \\ |w_{P,p_1}| + \mathtt{index}(p_2,P) & \text{, otherwise.} \end{cases}$$

Note that we assumed that P is in Chomsky Normal Form. If this was not the case, we can force this assumption with a linear time and space transformation. The fact that $\mathtt{index}(p, P)$ computes the smallest index k such that $w_p[k]$ is a variable can be shown by induction on $\mathtt{depth}(p)$. With respect to the time complexity, for each nonterminal p of an SCFG P, both the number $|w_p|$ and whether w_p contains a variable can be precomputed in linear time as stated in Lemmas 3.2.17 and 3.2.18, respectively. When these pre-computations are done, $\mathtt{index}(p, P)$ can be computed by a single run over the rules of P and hence, it runs also in linear time. □

4.2.3 Run time analysis

The algorithm presented in the previous subsection runs in polynomial time due to the following observations. Let n and m be the initial value of $\mathtt{depth}(G)$ and $|G|$, respectively. We define $V := \mathcal{X}$ to be the set of all the first-order variables at the start of the execution (before any of them has been converted into a nonterminal). As in the unification case, the final size of the grammar is bounded by $m + |V|n$ thanks to Lemmas 4.1.5 and 4.1.7. Our algorithms iterates at most V times. By Lemmas 3.2.23, and 4.2.2 each iteration takes linear time. Finally we check equality of two words generated by an SCFG P, which takes time $\mathcal{O}(|P|^3)$ thanks to Theorem 3.2.16. Hence, we have the following:

Theorem 4.2.3 *First-order matching of two terms represented by an STG can be done in polynomial time $\mathcal{O}((m + |V|n)^3)$, where m represents the size of the input STG, n represents its depth, and V represents the set of different first-order variables occurring in the input terms). This holds for the decision question, as well as for the computation of the unifier, whose components are represented by the final STG.*

Since $|V|, m, n$ are bounded by $|G|$, a rough estimation of the upper bound on the execution time is $O(|G|^6)$, which improves the $O(|G|^7)$-bound for the particular case of matching.

4.3 An implementation

Here, we present an implementation of the first-order unification algorithm presented in Section 4.1. As explained above, the algorithm runs a variant of Robinson's standard unification algorithm [Rob65] over two given STGs; it builds SCFGs for the preorder traversals of the input STGs, and then applies equivalence checking for SCFGs, while instantiating the encountered variables. For the equivalence check we implemented two competing algorithms:

(1) the exact algorithm due to Lifshits [Lif07], and

(2) the recent randomized algorithms by Schmidt-Schauß and Schnitger [SSS12].

Our tool, to which we refer as Unif-STG in the sequel, is integrated with the STG compressor TreeRePair [LMM13]: it takes as input two terms represented in XML syntax and runs TreeRePair to build STGs. It then runs the unification algorithm. For the version using (2), we implemented two variants, one using large integers and one using prime numbers.

Through experiments we evaluate the performance of the resulting three unification algorithms and compare them to an implementation of a classical unification algorithm over uncompressed terms. Roughly speaking, unification over STGs is more efficient than over uncompressed terms, whenever the terms are well-compressible and are larger than 100,000 nodes. Our system can be tested online at `www.lsi.upc.edu/~agascon/unif-stg`. All our code is open source and will be available over the same web page.

4.3.1 Equality testing

Given an STG $G = (\mathcal{N}, \Sigma, R)$ and two non-terminals A, B, equality testing consists of deciding whether $w_A = w_B$. Let us assume that $|w_A| = |w_B|$, since otherwise inequality is easily stated in linear time. As commented above, the fastest known exact algorithm for equality testing for SCFGs is Lifshits' algorithm [Lif07]. In Unif-STG we implemented, in addition to Lifshits' algorithm, two new algorithms [SSS12]. These algorithms run faster than Lifshits' by using a randomized approach. They work by considering an SCFG to represent a natural number, in addition to a word. The number coded by a word $w \in \Sigma^*$ is defined in terms a fixed mapping $f : \Sigma \to \{0, \ldots, |\Sigma| - 1\}$, as $\mathit{code}(w_A) = \mathit{code}(w') * |\Sigma| + f(a)$, if $w = w'.a$, and $\mathit{code}(w) = 0$ if $w = \lambda$. The number $\mathit{code}(w_A)$, for every non-terminal A can be computed in linear time w.r.t. $|G|$. See [SSS12] for further details.

The main idea of the algorithm is very simple. If we want to check whether A and B represent the same word, we choose a natural number m satisfying certain properties, and compute $\alpha = \mathit{code}(w_A)$ and $\beta = \mathit{code}(w_B)$ modulo m. If $\alpha \neq \beta$ then the words are obviously different. Otherwise, it is possible that $w_A \neq w_B$, but $a \equiv b \bmod m$. In this case we do not detect inequality. In [SSS12], two upper bounds for the choice of the m that guarantee that we detect inequality with a probability $\geq \frac{1}{2}$ for any pair of words are given: either $m \leq |w_A|^2 * c$ or $m \leq |w_A| * c$ if m is prime, for a certain constant c. We implemented both options in Unif-STG. By repeating the test k times the probability of not detecting inequality is $< \frac{1}{2^k}$. In Unif-STG k is set to 10 by default.

In order to assure that the chosen m is prime we implement a very simple algorithm: generate a random number, and test primality. If the number is not prime, then generate another number, and so on. We test primality with the Fermat primality test, checking if $a^{p-1} \equiv 1 \bmod p$ for $a \in \{2, 3, 5, 7\}$. Due to the Prime number theorem, the average number of times we generate a number until getting a prime is the logarithm of m, and hence linear in —G—, and the Fermat primality test is also performed in logarithmic time.

We also need an algorithm to compute if w_A is a prefix of w_B in order to find the first difference between two words represented with STGs (see Figure 4.1). This problem can be reduced to computing $\mathit{code}(w_B[1..|w_A|])$ and applying the probabilistic algorithm. To perform this task in linear time it is enough to precompute, for each non-terminal A of the grammar, the numbers $\mathit{code}(w_A)$ and $|w_A|$ and compute $\mathit{code}(w_B[1..|w_A|])$ recursively.

Finally, it is important to remark a certain peculiarity of the version of the probabilistic algorithms implemented in Unif-STG. They run in linear time thanks to the fact that $|w_A|$ is limited by default to $\sqrt{L}$ where $L = 2^{64}$, the maximum value for a long long int, in the case of the algorithm using primes; and to $\sqrt[4]{L}$, in the algorithm using natural numbers. Otherwise, computing $\mathit{code}(w_A)$ module m is not guaranteed to run in linear time.

The current implementation allows bigger values, but then does not guarantee an error probability of less than 0.5 for every possible instance of the problem. In our experiments we never encountered a false reply by the probabilistic algorithm.

Nevertheless, Unif-STG have been built to work with arbitrary arithmetic, the size limitation has been added just for efficiency reasons and can be removed at any time.

4.3.2 Unif-STG

Unif-STG is written in C++ using the standard template library. The system can be tested online at `www.lsi.upc.edu/~agascon/unif-stg` and the sources are also available as commented in the previous section, The system implements three algorithms for solving the equality testing with SCFGs: Lifshits' exact algorithm, plus two versions of the randomized algorithm by Schmidt-Schauß and Schnitger (one with integers and one with primes). We refer to the corresponding three versions of the unification algorithm for STG grammars by STG-exact, STG-rand, and STG-rand-prime. Our implementation of unification over uncompressed terms is denoted "tUnif". As commented before, this is a variant of Corbin-Bidoit algorithm. We refer the reader to Chapter 8, Section 2.3 of [BS01] for the details of this algorithm. Note that Unif-STG outputs a compressed representation of the solution (again compressed with STGs).

4.3.3 Experiments

Experimental Setup. All tests are executed on a machine with Intel Xeon Core 2 Duo, 3 Ghz processor, with 4GB of RAM. We use the Ubuntu Linux 9.10 distribution, with kernel 2.6.32 and 64 bits userland. Our implementation was compiled using g++ 4.4.1. We used TreeRePair (build data 01-19-2011) which was kindly made available to us by Roy Mennicke. It is essentially the version available at `http://code.google.com/p/treerepair`, with the only difference that it allows to compress without prior applying a binary tree encoding.

Protocol. Each test is executed three times, and the fastest time of the three runs is reported. We run TreeRePair with the switches "-multiary -bplex -c -nodag -optimize edges". Note that "-optimize edges" gives slightly better times (about 2% faster) for STG unification than "-optimize filesize" We only measure the pure unification time, i.e, we ignore loading time and setup of basic data structures, etc.

Design of the Experiments. Instead of trying to find instances of unification problems with large terms that are realistic and likely to appear in practice, we present results over artificial examples. The idea behind these examples is to test the behaviour of our algorithms in the different extreme corner cases. The main aspects that make up these cases are

(a) are the terms well-compressible by TreeRePair?

(b) do they unify or not?

(c) how many variables?

(d) is it only matching; how much copying of variables?

For question (a) we need to distinguish further: (a1) is the "top-matching part", i.e., parts where both terms do not have variables (and therefore must match exactly) well-compressible? And (a2) is the "binding part", that is, parts that will be bound to variables during unification well compressible? We constructed a family of instances that allows to test many of these aspects. First we show two simple examples which compress well. **Bin and Mon.** For a natural number n, let $f^n(a, b)$ denote a full binary tree with leaf sequence $ababab\ldots$. Given a natural number n, Bin(n) consists of the pair of trees

$$\mathrm{Bin}(n) = (g(g(f^n(a,b)), g(f^n(a,b))), \quad g(g(X,X), g(X, f^n(a,Y)))).$$

Similarly, $f^n(a)$ denotes a monadic tree of height n with internal nodes labeled f and leaf a. The second example, called Mon(n) consists of this pair of trees

$$\mathrm{Mon}(n) = (h(f^n(X), f^n(Y), Y), \quad h(T, Z, T)).$$

Clearly, both Bin and Mon are unifiable for every n. Moreover, they are well compressible with TreeRePair. To see this, consider the right part of Table 4.1 which shows the compression time, the number of edges in the original tree and in the STG grammar, plus the file sizes of the original tree (in XML format) and of the grammar (in text format). It also shows the file size of the grammar in CNF in the special format that our unification program uses. For both Bin and Mon, the TreeRePair algorithms achieves exponential compression rates. As can be seen, for $n > 20000$, the STG-rand algorithm is the fastest. Interestingly, for such small grammars we are punished for using prime numbers and STG-rand-prime is slower than STG-rand. This is different for larger grammars as the later examples show. Note

	Runtime (in ms)				Input			
$n/1000$	tUnif	STG- randp	STG- rand	STG- exact	edges	compr. time	STG edges	CNF file
5	**2**	8	8	24	10008 (69K)	55ms	38 (388B)	1K
10	**5**	10	8	28	20008 (137K)	62ms	40 (398B)	1.1K
20	11	11	**9**	30	40008 (157K)	140ms	42 (420B)	1.1K
50	44	11	**9**	30	100T (684K)	341ms	45 (434B)	1.2M
100	107	12	**10**	31	200T (1.4M)	681ms	47 (457B)	1.3M
200	232	13	**10**	32	400T (2.7M)	1387ms	49 (467B)	1.3M

Table 4.1: The example Mon(n)

	Runtime (in ms)				Input		
randSize	tUnif	STG-rp	STG-r	STG-e	edges	STG edges	CNF file
10	**3**	18	19	78	20484 (111K)	62 (578B)	1.9K
11	**7**	20	20	96	40964 (221K)	66 (596B)	2.1K
12	**16**	22	22	108	81924 (441K)	70 (638B)	2.2K
13	35	**23**	23	131	163844 (881K)	74 (656B)	2.3K
14	72	26	**25**	146	327684 (1.8M)	78 (698B)	2.4K
16	290	30	**28**	(*)	1310724 (6.9M)	86 (758B)	2.7K

(*) STG-exact ran out of (int) bounds.

Table 4.2: The example Bin(n)

that here the exact algorithm still shows reasonable performance. This will not be the case for larger grammars. Note that, in terms of XML, Mon is actually quite relevant: a long list of items usually becomes a long list of siblings in XML. Using the common "first-child/next-sibling"-encoding of unranked into binary trees, such a list becomes a long path, similar to Mon.

Bad Instances for STG-Unif. Here consider instances where the STG-based unification algorithm does *not* perform well. In general, this is the case when the terms are *not well compressible* (see below). But, there are even simpler reasons for this to happen. Consider unifying the trees $f(t)$ and $g(t')$ for large (arbitrary) terms t and t'. The run time of tree-based unification is only 0.005ms for this instance. While, even for highly compressible $t = t'$, STG-Unif will take > 15ms. This is due to the fact that STG-Unif always needs to traverse the whole grammar to find the position of the first difference between $f(t)$ and $g(t')$ and tUnif traverses the input tree *only* until the position of the first difference is reached.

Meta. We now define a highly configurable example instance. Consider the pair of trees (t_1, t_2), where both t_1 and t_2 are full binary trees (with internal nodes labeled 'f) of height n. At the leaves of t_1 and t_2 appear monadic trees of random height h, with $minHeight \leq h \leq maxHeight$. These monadic trees are identical in t_1 and t_2. Now, t_1 contains variables as leaves of the monadic trees, randomly chosen from a given set *Vars* of variables. While t_2 contains random trees at those leaf positions, chosen over a given signature Σ, and maximal size of up to *randSize*. Moreover, a Boolean determines whether at variable copies we force the random trees in t_2 to be equal (which will guarantee that the instance is unifiable). Thus, the specification of an instance is as follows: Meta(n, *minHeight*, *maxHeight*, *Vars*, *randSize*, Σ, Bool U).

Number of Variables. Using Meta, we experimented with the number of different unification variables. The results were convoluted and no clear trends were observable; both algorithms seemed similarly impacted by the number of variables. For instance, for $n = 4$, $maxHeigh = minHeight = 1000$, $randSize = 1$ and $|\Sigma| = 3$ we obtain, for 3 variables: 3ms/18ms (tUnif/STG-rand-prime), for 5 variables: 7ms/32ms, and for 10 variables: 10ms/44ms.

Incompressible Terms. An interesting case is if large incompressible terms appear at positions that will instantiate variables. In terms of Meta, it suffices to take $n = 1$, and to use large random trees. For the other parameters we use $\mathit{minHeight} = \mathit{maxHeight} = 0$, $\mathit{Vars} = \{X, Y\}$, $\Sigma = \{g^{(2)}, f^{(1)}, a^{(0)}\}$, and Boolean U set to true. As Table 4.3 shows, STG-Unif is indeed about 100-times slower than tree-based unification. The difference in speed seems to get slightly smaller for very large inputs. As comparison, if we add a larger binary tree on top of t_1, i.e., use a larger n, then the tree becomes more compressible and therefore STG-based unification becomes efficient. This is shown in the right of Figure 4.3, where we pick $\mathit{randSize} = 20000$, but now use monadic trees of size 0–1000. With respect to *unifiability*, we observed that changing a few nodes to make the input non-unifiable, causes STG-rand to take ca. twice the time given in Table 4.3, while tUnif gets slightly faster.

	Runtime (in ms)				Input		
randSize	tUnif	STG-rp	STG-r	STG-e	edges	STG edges	CNF file
1000	**0.1**	21	34	222	981 (5.9K)	214 (1.5K)	6.6K
5000	**0.6**	78	116	2430	4778 (29K)	774 (15K)	25K
20000	**2.4**	405	598	43632	26114 (157K)	3308 (21K)	107K
50000	**12**	1396	2074	(*)	94280 (564K)	9975 (63K)	327K
200000	**43**	5464	8036	(*)	334586 (2M)	30740 (196K)	1.1M

(*) STG-exact ran out of memory.

Table 4.3: Incompressible Terms in Substitution Positions

There are also examples where the solution consists of deeper trees than the input. This works well for the uncompressed algorithm too. But, we can see the effect of compression: consider $t_1 = h(X, Y, Z)$ and $t_2 = h(s_1, s_2, s_3)$, where s_1 is a full binary tree (over f's) of height n with all leaves labeled Y, s_1 is a full binary tree (over f's) of height n with all leaves labeled Z, and s_3 the same but with leaves labeled X. Note that X will be assigned to s_1. Hence, all the X's in s_3 will be replaced by s_1. Then, Y will be replaced by s_2 everywhere (also in s_3). So finally $t' = Y \to s_2(X \to s_1(s_3))$ will be compared to Z. Note that t' is the complete tree of depth $3 * 20$ whose leaves are all labeled Z. Since Z occurs in t' unification fails. This example is called "3-Stack" and timings are shown in Table 4.3

4.3.4 Operations on STGs

In this section we present a contruction that, given an STG G, a non-terminal A of G and an natural number k, extends G with new non-terminals such

n	tUnif	STG-rp	STG-r	STG-e
18	99	**7**	9	35
19	200	**8**	9	38
20	401	**8**	10	(*)

n	tUnif	STG-rp	STG-r	STG-e
7	**369**	1694	2130	(*)
8	**688**	1726	2149	(*)
9	**1730**	2391	2982	(*)

(*) STG-exact ran out of memory.

Figure 4.3: The example 3-Stack (left) and randSize=20000 of Table 4.3 (right)

that one of them generates $t_{G,A}|_{\mathrm{iPos}(w_{G,B},k)}$. This construction is analogous to the one presented in Definition 3.2.22 for monadic grammars. We extended this result to allow arbitrarily many holes in our implementation. The main reason to do that is that TreeRepair works with grammars with parameters.

First of all we present some basic results and definitions.

Definition 4.3.1 *Given a signature Σ and a set of parameters $\mathcal{Y}$, we define $|t|$, the size of a tree pattern $t \in \mathcal{T}(\Sigma, \mathcal{Y})$, as $|t| = 1 + |t_1| \dots |t_n|$ if $t = f(t_1, \dots, t_n)$ with $n > 0$, $|t| = 1$ if t is a constant, and $|t| = 0$ if $t \in \mathcal{Y}$.*

We define $\mathrm{Pre}(t)$ to be the preorder traversal or a tree pattern t. Note that the only difference with a preorder traversal of a term is that we skip parameters.

Definition 4.3.2 *Given a signature Σ and a set of parameters $\mathcal{Y}$, we define $Pre(t)$, the preorder traversal of a tree pattern $t \in \mathcal{T}(\Sigma, \mathcal{Y})$, as $Pre(t) = f\,Pre(t_1) \dots Pre(t_n)$ if $t = f(t_1, \dots, t_n)$ and $n > 0$, $Pre(t) = c$ if t is a constant, and $Pre(t) = \lambda$ if $t \in \mathcal{Y}$, where λ denotes the empty word.*

Definition 4.3.3 ([LMSS12]) *A linear STG $G = \mathcal{N}, \Sigma, R, S)$ is in* Chomsky Normal Form (CNF) *if for every $(N \to t) \in R$ with $\mathtt{ar}(N) = n$, the term t is of one of the following forms:*

- *$f(y_1, \dots, y_n)$ with $f \in \Sigma$.*
- *$B(y_1, \dots, y_{i-1}, C(y_i, \dots, y_{j-1}), y_j, \dots, y_n)$ with $B, C \in \mathcal{N}$, $1 \leq i \leq j \leq n$.*
- *$A \to B$ (λ rule).*

We assume that STGs are in CNF. This is not a loss of generality since this assumption can be forced by a linear time transformation as shown in [LMSS12]. In the rest of this section, for clarity, we represent rules rule of the form $f(y_1, \dots, y_n)$ as $A \to f$, rules of the form $A \to B(y_1, \dots, y_{i-1}, C(y_i, \dots, y_{j-1}), y_j, \dots, y_n)$ with $B, C \in \mathcal{N}$, $1 \leq i \leq j \leq n$

as $A \to (B, i, C)$, and λ rules as $A \to B$. Actually, this is the internal representation used by Unif-STG.

Let t be a tree pattern with parameters $\{y_1, \dots, y_n\}$. We define the first chunk of t as the word obtained by traversing t in preorder from the root position to the position labeled with y_1, the second chunk as the word obtained by traversing t in preorder from the position labeled with y_1 to the position labeled with y_2, and so on. Finally, the $(n+1)$th chunk is the word obtained by traversing t in preorder starting at the position labeled with y_n.

Definition 4.3.4 (chunks) *Let G be an STG. Let A be a non-terminal of G with $\mathtt{ar}(A) > 0$. For every $1 \leq j \leq \mathtt{ar}(A) + 1$, we recursively define the jth chunk of A, denoted (A, j), as follows.*

- *If $A \to f \in G$ and $j = 1$ then $(A, j) = f$.*
- *If $A \to f \in G$ and $j > 1$ then $(A, j) = \lambda$, where λ denotes the empty word.*
- *If $A \to (B, i, C) \in G$ and $j < i$ then $(A, j) = (B, j)$.*
- *If $A \to (B, i, C) \in G$, $j = i$, and $\mathtt{ar}(C) = 0$, then $(A, j) = (B, i)\mathit{Pre}(t_C)(B, i + 1)$.*
- *If $A \to (B, i, C) \in G$, $j = i$, and $\mathtt{ar}(C) > 0$ then $(A, j) = (B, i)(C, 1)$*
- *If $A \to (B, i, C) \in G$, $\mathtt{ar}(C) > 0$, and $i < j < i + \mathtt{ar}(C)$ then $(A, j) = (C, j - i + 1)$*
- *If $A \to (B, i, C) \in G$, $\mathtt{ar}(C) > 0$, and $i < j = i + \mathtt{ar}(C)$, then $(A, j) = (C, \mathtt{ar}(C) + 1)(B, i + 1)$*
- *If $A \to (B, i, C) \in G$, $\mathtt{ar}(C) > 0$, and $j > i + \mathtt{ar}(C)$ then $(A, j) = (B, j - \mathtt{ar}(C) + 1)$.*
- *If $A \to B$ then $(A, j) = (B, j)$.*

The following lemma can be easily proven by induction on $\mathtt{depth}(G)$ and distinguishing cases according to definition 4.3.4.

Lemma 4.3.5 *Let G be a k-bounded STG of size n. An SCFG G_c can be computed in time $\mathcal{O}(nk)$ such that, for every non-terminal N of G with $\mathtt{ar}(N) > 0$ and natural number $1 \leq j \leq \mathtt{ar}(N) + 1$, there exists a nonterminal N_j of G_c that generates the jth chunk of N.*

Now we have all the ingredients to define how subterms of terms represented by STGs are computed by Unif-STG. Recall that the following operation is needed in order to obtain a non-terminal that generates the term to with a certain first-order variable is assigned in the unification process (see Figure 4.1). Analogously as done in Definition 3.2.22, we recursively define the grammar extension `kext` as follows.

Definition 4.3.6 *Let G be an STG. Let A be a non-terminal of G, and let k be a natural number satisfying $k \leq |Pre(t_{G,A})|$. We recursively define* $\texttt{kext}(G, A, k)$ *as an extension of G as follows:*

- *If $k = 1$ then $\texttt{kext}(G, A, k) = G$. In the next cases we assume $k > 1$.*
- *If $A \to (B, i, C) \in G$, and there exists $j \leq i$ such that $\sum_{l=1}^{j} |(B, l)| \leq k < \sum_{l=1}^{j+1} |(B, l)|$ then $\texttt{kext}(G, A, k)$ includes $\texttt{kext}(G, B, k)$, which contains a non-terminal N' generating the subterm of $t_{G,B}$ at position $iPos(w_{G,B}, k)$. If $\texttt{ar}(N') \geq i - j$ then $\texttt{kext}(G, A, k)$ additionally contains the rule $A' \to (N', i - j, C)$, where A' is a new term non-terminal with $\texttt{ar}(A) = \texttt{ar}(N') + \texttt{ar}(C) - 1$.*
- *If $A \to (B, i, C) \in G$, and it holds that $k' = \sum_{l=1}^{i} |(B, l)| < k \leq k' + |C|$ then $\texttt{kext}(G, A, k) = \texttt{kext}(G, C, k - k')$.*
- *If $A \to (B, i, C) \in G$, and there exists $i < j \leq \texttt{ar}(B)$ such that $\sum_{l=1}^{j} |(B, l)| < k \leq \sum_{l=0}^{j+1} |(B, l)| + |C|$ then $\texttt{kext}(G, A, k) = \texttt{kext}(G, B, k - |C|)$.*
- *If $(A \to A_1) \in G$ then $\texttt{kext}(G, A, k) = \texttt{kext}(G, A_1, k)$.*

The following Lemma states the correctness of the previous definition and is analogous to Lemma 3.2.23.

Lemma 4.3.7 *Let G be an STG. Let A be a non-terminal of G, and let k be a natural number satisfying $k \leq |Pre(t_{G,A})|$. Then, $G' = \texttt{kext}(G, A, k)$ contains a non-terminal N generating $t_{G,A}|_{iPos(w_{G,B},k)}$. Moreover, G' can be computed in time $\mathcal{O}(nk)$ and $|G'| \leq |G| + \texttt{depth}(G)$.*

Proof. The correctness of the construction can be checked by induction on $\texttt{depth}(A)$ and distinguishing cases according to the definition of $kext(G, N, k)$. For the size bound it's is enough to notice that the number of recursive calls in the construction of $kext(G, A, k)$ is bounded by $\texttt{depth}(A) \leq \texttt{depth}(G)$ and at most one non-terminal is added at each recursive call. The time bound holds by Lemmas 4.3.5 the fact that the sizes of the words represented by each non-terminal of an STG can be computed in linear time. in time $\mathcal{O}(nk)$. □

It rests to prove the analogous of Lemma 4.1.7 to non-monadic STG's, i.e. that the number of added non-terminals due to the `kext` construction is bounded by the `Vdepth` of the STG and that the `Vdepth` of the STG does not increase due to the `kext` extension. Since our construction is analogous to the one presented for monadic grammars, this facts can be proved using the same scheme that in the proof of Lemma 4.1.7.

Chapter 5

One Context Unification

In this chapter we tackle the *one context unification* problem, the particular case of context unification (Definition 2.3.3 in Chapter 2,) in which only one context variable may occur in the input terms, possibly with many occurrences. Let us state again the definition of the one context unification problem.

Definition 5.0.8 *For a fixed ranked alphabet $\mathcal{F}$, we define an* instance *of the one context unification problem as a triple $\langle \Delta, \mathcal{X}, F \rangle$, where $\mathcal{X}$ is a set of first-order variables, F is a context variable, and Δ is a set of equations, i.e. a set of unordered pairs, of the form $s \doteq t$, with $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \{F\})$. A solution of $\langle \Delta, \mathcal{X}, F \rangle$ is a substitution $\sigma : \mathcal{X} \cup \{F\} \to \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \{F\}) \cup \mathcal{C}(\mathcal{F}, \mathcal{X} \cup \{F\})$ such that $\forall (s \doteq t) \in \Delta \ : \ \sigma(s) = \sigma(t)$.*

Note that, after applying the standard decomposition and variable elimination rules of first-order unification (see [BS01]), we can assume that an instance of one context unification is of the form $\{F(s_1) \doteq t_1, \ldots, F(s_n) \doteq t_n\}$. This assumption can be forced by a polynomial time an space transformation.

Example 5.0.9 *Consider the set of equations $\{F(a) \doteq g(a, x), F(b) \doteq g(x, b)\}$. It can be unified by the substitution $\{F \mapsto g(\bullet, b), x \mapsto b\}$. Now consider the equation $F(g(x, b)) \doteq g(a, F(y))$. It has infinitely many solutions such as $\{F \mapsto g(a, \bullet), x \mapsto a, y \mapsto b\}$ and $\{F \mapsto g(a, g(a, \bullet)), x \mapsto a, y \mapsto b\}$. Finally, consider the set of equations $\{F(a) \doteq g(x, y), F(b) \doteq g(x, g(g(y, y), g(a, b)))\}$, which has no solution.*

5.1 The General Scheme

Given a set Δ of equations over terms containing some first-order variables and at most one context variable F, we present an algorithm that determines if the set Δ has a solution. We will argue that our procedure is in NP.

The algorithm is described, analogously to some classical unification algorithms, as an inference system that manipulates the set Δ. The inference system runs in two phases. In a first phase, the inference rules deal with the context variable F. The first phase ends when F is eliminated. At the end of the first phase, we obtain either a first-order term unification problem, or a term unification problem containing subexpressions of the form $C^{|p|e}$, where c is a context, p is the position of the hole in c, and e is a variable ranging over non negative integers. The inference rules in the second phase deal with this new kind of expressions and eliminate them.

Let c be a context. We use $\mathtt{sp}(c)$ to denote the list of function symbols occurring on the path from the root to $\mathtt{hp}(c)$. More formally, if $\mathtt{hp}(c)$ is of the form $i_1.i_2.\cdots.i_{|\mathtt{hp}(c)|}$, then $\mathtt{sp}(c)$ is $\mathtt{root}(c), \mathtt{root}(c|_{i_1}), \mathtt{root}(c|_{i_1.i_2}), \ldots, \mathtt{root}(c|_{i_1.i_2.\cdots.i_{|\mathtt{hp}(c)|-1}})$.

The following is an alternative definition of the context exponentiation operation defined in the preliminaries.

Definition 5.1.1 (Rotation and Context with exponent) *Let* $c = f(u_1, \ldots, u_{i-1}, d, u_{i+1}, \ldots, u_m)$ *be a non empty context, where* d *is a context and* u_i *is a term, for every* $i \in \{1, \ldots, i-1, \ldots, i+1, \ldots, m\}$*. For any non negative integer* $n \geq 0$*, we define* $\mathtt{rot}(c,n)$ *and* c^n *recursively as follows:*

$$\begin{aligned}
\mathtt{rot}(c,0) &:= c \\
\mathtt{rot}(c,n) &:= \mathtt{rot}(d[f(u_1, \ldots, u_{i-1}, \bullet, u_{i+1}, \ldots, u_m)], n-1) \\
c^0[\bullet] &:= \bullet \\
c^n[\bullet] &:= f(u_1, \ldots, \mathtt{rot}(c,1)^{n-1}, \ldots, u_m)
\end{aligned}$$

For any nonempty context c and any $n \geq 0$, note that $\mathtt{rot}(c,n) = \mathtt{rot}(c, n \bmod |\mathtt{hp}(c)|)$ and $|\mathtt{hp}(c)| = |\mathtt{hp}(\mathtt{rot}(c,n))|$. Also note that c^1 is not necessarily equal to c, whereas $c^{|\mathtt{hp}(C)|} = c$ always holds.

Example 5.1.2 *Let* $c = f(g(\bullet))$ *be a context. Then,* $\mathtt{rot}(c,0) = \mathtt{rot}(c,2) = \mathtt{rot}(c,4) = c$ *and* $\mathtt{rot}(c,1) = \mathtt{rot}(c,3) = g(f(\bullet))$*. Moreover,* $c^1[a]$ *represents* $f(a)$*, the notation* $c^5[a]$ *succinctly represents the term* $f(g(f(g(f(a)))))$*.*

Let us extend the notion of unifier to allow variables to be mapped to terms and contexts containing exponent expressions.

Definition 5.1.3 *Given an instance* Δ *of one context unification, a* unifier σ *is a substitution mapping each first order variable* x *to a term* t_x *and* *the context variable F to a context c_F, such that t_x and c_F possibly contain expressions of the form $c_p^{|p|N}$ for some fixed single integer variable N, such that for every substitution δ that instantiates N by a non negative number and for every equation $s \doteq t \in \Delta$, $\delta(\sigma(s)) = \delta(\sigma(t))$ holds. We speak of a* ground unifier *or a* solution*, if the unifier maps all variables to ground terms, in which case no exponent expressions are required.*

A set S of unifiers, where we assume that only a single integer variable N is used, is a complete set of unifiers *of Δ, iff for every ground unifier γ of Δ (i.e. for every solution), there is a $\sigma \in S$, and a ground substitution ρ such that for all variables x occurring in Δ, we have $\gamma(x) = \rho(\sigma(x))$. By abuse of notations we will refer as* most general unifiers *to the elements of S.*

5.2 PHASE1 Inference System: Eliminating the Context Variable

The PHASE1 inference rules are given in Figure 5.1. These inference rules are applied nondeterministically to transform the current set Δ into one of the finitely many possible sets $\Delta_1, \dots, \Delta_k$. The first phase ends when either a contradiction ($\perp$) is reached, or the context variable disappears. For efficiency reasons we will use a DAG representation of terms and contexts, but use a notation as for terms.

The first two rules – `Decompose` and `Var-Elim` – are the standard rules to simplify a unification problem and eliminate a first-order variable [BS01]. The rule `Var-Elim2` partially guesses the context variable F in terms of a new context variable F' and eliminates a first-order variable x. The rule `CVar-Elim` eliminates F by (nondeterministically) guessing a context c to be the value of F in the solution. Note that c here means a *first-order* context, thus it cannot contain a context variable. The rule `CVar-Elim2` eliminates F by again (nondeterministically) guessing a position of length $0 \leq k < |p|$ that determines the value of F using the exponent notation (Definition 5.1.1). Note that the instantiation $F \mapsto c^{|p|N+k}$ introduces exponent expressions containing a variable N ranging over the non negative integers. The term $c^{|p|N+k}$ is actually represented as $c^{|p|N}[c^k]$, that is, *only exponent expressions of the form $c^{|p|N}$ are introduced*, and the term C^k is expanded (into a regular term) according to Definition 5.1.1.

Example 5.2.1 *Let $\Delta = \{f(F(f(y,z)),z) \doteq F(f(f(y,z),z))\}$. Rule* `CVar-Elim2` *is applicable to Δ. Here, $C = f(\bullet, z)$ and hence $|p| = 1$. Therefore, k can only be 0 and we get the substitution $\{F \mapsto f(\bullet, z)^N\}$. Apply-*

`Decompose`: $$\frac{\Delta \cup \{\alpha(t_1,\ldots,t_n) \doteq \alpha(u_1,\ldots,u_n)\}}{\Delta \cup \{t_1 \doteq u_1,\ldots,t_n \doteq u_n\}}$$

where α is either a function symbol ($n = \mathtt{arity}(\alpha)$) or a first-order variable ($n = 0$) or a context variable ($n = 1$). One of $\alpha(t_1,\ldots,t_n)$ or $\alpha(u_1,\ldots,u_n)$ must have maximal depth among all terms in $\Delta \cup \{\alpha(t_1,\ldots,t_n) \doteq \alpha(u_1,\ldots,u_n)\}$.

`Var-Elim`: $$\frac{\Delta \cup \{x \doteq t\}}{\{x \mapsto t\}(\Delta)}$$

where x is a first order variable that does not occur in t.

`Var-Elim2`: $$\frac{\Delta \cup \{F(t) \doteq c[x]\}}{(\{x \mapsto F'(\rho(t))\} \circ \rho)(\Delta)}$$

where $c \neq \bullet$ is guessed (note that c does not contain F) and $\rho = \{F \mapsto c[F'(\bullet)]\}$, and where x is not contained in t, and if F occurs in t, then x must not occur in c.

`CVar-Elim`: $$\frac{\Delta \cup \{F(t) \doteq c[u]\}}{\{F \mapsto c\}(\Delta \cup \{t \doteq u\})}$$

where c is guessed (note that c does not contain F).

`CVar-Elim2`: $$\frac{\Delta \cup \{F(t) \doteq c[F(u)]_p\}}{\{F \mapsto c^{|p|N+k}(\bullet)\}(\Delta \cup \{t \doteq \mathtt{rot}(c,k)[u]\})}$$

where N is an integer variable and k is guessed such that $0 \leq k < |p|$ and $p = \mathtt{hp}(c)$ (note that c does not contain F).

`Occurs-Check`: $$\frac{\Delta \cup \{x \doteq t\}}{\perp}$$

where x occurs in t and $t \neq F(\ldots F(x)\ldots)$, i.e. t is not a term consisting only of F's and x.

`Fail`: $$\frac{\Delta \cup \{f(t_1,\ldots,t_n) \doteq g(u_1,\ldots,u_m)\}}{\perp}$$

where $f \neq g$.

Figure 5.1: PHASE1 inference system for eliminating a context variable. The inference rules are applicable only when the side-condition holds.

ing `CVar-Elim2` *using this substitution, we get the new set* $\{f(f(y,z),z) \doteq \mathtt{rot}(f(\bullet,z),0)[f(y,z)]\}$*, which is simply* $\{f(f(y,z),z) \doteq f(f(y,z),z)\}$*. Figure 5.4 contains a more involved example.*

The following correctness statements and their proofs provide further intuition for the inference rules. Note that in each inference step, the new set Δ' is obtained from the old set Δ by removing some equations, adding some new equations, and applying a substitution to all the terms.

Lemma 5.2.2 (Soundness) *Let* $\Delta_1 \vdash \Delta_2$ *be a* PHASE1 *inference step and let* σ_1 *be the substitution used in this inference step. If* σ *is a solution for* Δ_2*, then* $\sigma \circ \sigma_1$ *is a solution for* Δ_1*.*

Proof.

For each PHASE1 inference rule, we can write Δ_2 as $\sigma_1(\Delta_1 \setminus \Delta_d \cup \Delta_a)$, where Δ_d are the deleted equations and Δ_a are the added equations. If $\Delta_a \neq \emptyset$, it is easily verified that if σ is a solution for $\sigma_1(\Delta_a)$, then $\sigma \circ \sigma_1$ is a solution of Δ_d. If $\Delta_a = \emptyset$, as in the rules `Var-Elim` and `Var-Elim2`, it is easily verified that σ_1 is a solution of Δ_d, hence also $\sigma \circ \sigma_1$ is a solution of Δ_d.

Consider, for example, the rule `CVar-Elim2`. Let $\sigma_1 = \{F \mapsto c^{|p|N+k}\}$, where c is a first-order context, and suppose σ is a solution of $\sigma_1(\Delta \cup \{t \doteq \mathtt{rot}(c,k)[u]\})$, for some term u. Obviously, $\sigma \circ \sigma_1$ solves Δ. We show that $\sigma \circ \sigma_1$ also solves $F(t) \doteq c[F(u)]_p$ as follows, where $\theta = \sigma \circ \sigma_1$:

$$\begin{array}{rclcl}
\theta(F(t)) & = & \sigma(c^{|p|N+k})[\theta(t)] & = & \sigma(c^{|p|N+k})[\theta(\mathtt{rot}(C,k)[u])] \\
 & = & \sigma(c^{|p|N+k})[\sigma(\mathtt{rot}(c,k)[\sigma_1(u)])] & = & \sigma(c^{|p|N+k}[\mathtt{rot}(c,k)[\sigma_1(u)]]) \\
 & = & \sigma(c[c^{|p|N+k}[\sigma_1(u)]]) & = & \theta(c[F(u)])
\end{array}$$

Apart from the definitions of σ and σ_1, we also use the fact that $c^{|p|N+k}[\mathtt{rot}(c,k)] = c[c^{|p|N+k}]$ above. Soundness of all other rules can be argued similarly. □

Lemma 5.2.3 (Completeness) *Suppose* Δ *can be transformed to one of* $\Delta_1, \ldots, \Delta_m$ *by the* PHASE1 *inference system using substitutions* $\sigma_1, \ldots, \sigma_m$ *respectively. If* σ *is a solution of* Δ*, then there exists a solution* θ *of some* Δ_i *such that* $\sigma(x) = \theta \circ \sigma_i(x)$*,* $\sigma(F) = \theta \circ \sigma_i(F)$ *and* $\sigma(N) = \theta \circ \sigma_i(N)$*, for all variables* x, F, N *occurring in* Δ*.*

Proof. Suppose that the solution σ for Δ instantiates F by a context d, and let p be $\mathtt{hp}(d)$. We consider different cases based on the form of the equations in Δ. The interesting case is when Δ contains an equation $F(t) \doteq s$. The choice of which inference rule to apply to Δ to identify the required Δ_i can be guided by d and the form of s.

- Assume that p is a position of s. Then p is a prefix of all positions labeled with F in s, because, if not, then the context d will properly contain d, which leads to a contradiction. Thus, s can be written as $d'[u]$ for a context d' not containing F, and such that $\sigma(d') = d$. In this case, the rule `CVar-Elim` can be used to get the desired Δ_i from Δ.

- Assume that p is not a position of s. Then s contains a unique maximal position q that is a proper prefix of p, such that the roots of $s|_{q'}$ and $d|_{q'}$ are identical signature symbols for all proper prefixes q' of q. We distinguish two cases two exclusive cases: either (a) $s|_q = x$ for some first order variable x and $s = c[x]_q$, or (b) $s|_q = F(s')$ and $s = c[F(s')]_q$, and F does not occur in c (otherwise the context d would properly containing d, a contradiction).

  In case (a), i.e. if $s|_q = x$ is a first-order variable, it follows that s is of the form $c[x]_q$, and d is of the form $\sigma(c)[d'] = \sigma(F)$, for some non empty context d'. Moreover, it must be the case that, $d[\sigma(t)] = \sigma(c)\sigma(x)]$. In this case, the rule `Var-Elim2` can be used: the substitution $\rho = \{F \mapsto c[F'(\bullet)]\}$ replaces the equation $F(t) \doteq c[x]$ by the equation $c[F'(\rho(t))] \doteq c[x]$, which is equivalent to the $F'(\rho(t)) \doteq x$. Since the system is unifiable, x is not contained in $F'(\rho(t))$, hence the second part of the conditions of `Var-Elim2` is satisfied. The second part of the substitution is then $\{x \mapsto F'(\rho(t))\}$, where $x \notin \mathsf{Vars}(F'(\rho(t)))$. We obtain the required Δ_i where the required θ is the same as σ except that $\theta(F') = d'$.
  Now, suppose $s|_q$ is $F(s')$. We know that F does not occur in c. Unifiability of the equations and the fact that d cannot be a proper subcontext of itself enforce that p has to be of the form $q^e q'$ for a prefix q' of q and a non negative integer e. Let us write s as $c[F(s')]_q$. Then, d has to be of the form $\sigma(c)^{|q|e+|q'|}$. This means that the rule `CVar-Elim2` can be used to obtain the required Δ_i.

For other choices of the solution σ and the set Δ, we can argue similarly to complete the proof. □

Note that if Δ contains a context variable, then at least one of the PHASE1 rules can be used. Hence, when no more rules can be applied, we are guaranteed to have eliminated all context variables.

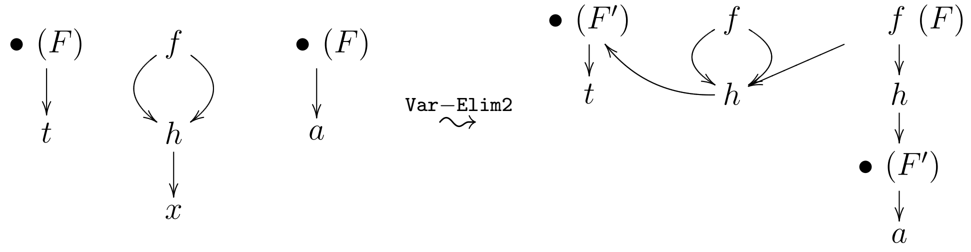
We still need to argue that (a) the size of the representation of the set of equations resulting from the first phase is polynomially bounded, and that (b) the first phase terminates in a polynomial number of (nondeterministic) steps.

We assume the DAG representation for terms introduced in Chapter 3. In particular, in this section we refer to a DAG as a a graph, without using its STG representation. Hence, we will refer to nodes of the DAG, and not to its nonterminals. Contexts are represented like terms (in the DAG) with the hole $\bullet$ as a constant (do not confuse this with the context nonterminals of an STG). A slight exception is $c^{|p|N}$ which is represented using a single node labelled with the triple $(c, |p|, N)$. Recall that $c^{|p|N+k}$ is always represented as $c^{|p|N}[c^k]$, where c^k is represented as a regular term.

The only rules that may increase the number of nodes in the DAG are rules that instantiate the context variable F, namely `Var-Elim2`, `CVar-Elim`, and `CVar-Elim2`. The rules `CVar-Elim` and `CVar-Elim2` can be applied at most once, since these rules eliminate all occurrences of context variables. The number of additional nodes to create a context c is at most $\mathtt{hp}(c)$, which is bounded by the current number of nodes in the DAGs. This implies that the application of `CVar-Elim` and `CVar-Elim2` may cause at most a quadratic blowup. Now consider the application of the substitution $\{F \mapsto c[F'(\bullet)]_p\}$ in `Var-Elim2`. This causes an addition of $l * |p|$ new nodes in the DAG – l copies of each node in $\mathtt{sp}(c)$ – where l is the number of occurrences of F in Δ.

Note that an application of the rule `Var-Elim2` implies (by the definition of contexts) that there is no occurrence of F in $c[x]_p$. Hence, an application of `Var-Elim2` makes a copy of a node only when it has no context variable F below it. However, each newly added node will necessarily have a context variable (F') below itself. The property that a node has a context variable below it is preserved by the rules `Decompose`, `Var-Elim`, and `Var-Elim2` for all nodes. Therefore, a newly created node will never be copied again. Thus, the number of new nodes that can be added by `Var-Elim2` is bounded by $l * n$, where n is the node count of the original DAG. It is easy to see that the current number of nodes labeled with the context variable is also bounded above by n, since the sum of the number of first-order variable nodes and of the nodes labeled with the context variable is not increased by `Var-Elim2`. This proves that the size of the DAG created in the first phase is polynomially bounded.

Example 5.2.4 *Consider* $\Delta = \{F(t) \doteq f(h(x), h(x)),\ F(a) \doteq x\}$*. Applying* `Var-Elim2` *with* $\langle x \mapsto F'(t)\rangle(\langle F \mapsto f(h(x), h(F'))\rangle)$ *gives* $\Delta' = \{f(h(F'(t)), h(F'(a))) \doteq F'(t)\}$*. The DAG representations of (all terms in)* Δ *and* Δ' *are shown below. Note that the node representing* $F(a)$ *in the original DAG causes* copying *and the creation of two new nodes, marked with* f *and* h*. However, these new nodes will never get copied in the future as they will continue to have a context variable* F' *below them.*



Lemma 5.2.5 (Termination) *Using DAGs to represent terms, any* PHASE1 *derivation terminates in a polynomial number of steps.*

Proof. Any PHASE1 derivation is immediately terminated if we apply `CVar-Elim`, `CVar-Elim2`, `Var-Elim2` where the second part is `Occurs-Check`, `Occurs-Check`, or `Fail`. Single rule applications together with their applicability checks can be done in polynomial time on DAGs using standard techniques. The rules `Var-Elim` and `Var-Elim2` eliminate a first-order variable, and hence they can be applied at most a linear number of times. The rule `Decompose` preserves the number of variables. Note that the maximal number of possible equations is quadratic in the number of nodes in the DAG. Sequences of applications that consist only of `Decompose` have an at most polynomial length, since in every step at least one equation is processed that cannot occur again in this sequence. Since the number of nodes in the DAG is polynomially bounded, termination is guaranteed in (nondeterministic) polynomial time. □

Lemma 5.2.5 and the polynomial bound on the node count of the DAG show that the first phase runs in nondeterministic polynomial time. We remark here that the rules `Decompose` and `Var-Elim` can be applied eagerly (since they correspond to "don't care" nondeterminism). The rules `Var-Elim2`, `CVar-Elim`, and `CVar-Elim2` involve "don't know" nondeterministic guesses.

Lemma 5.2.6 (Result of Phase 1) *The first phase terminates either with Fail (i.e. $\perp$), or with success and the output is an empty set of equations or a set of equations including exponent expressions. If the output is an empty set, then the unifier σ is the composition of the substitutions of the rule applications. The unifier can be represented in polynomial space if the substitution is applied using DAGs as representation for term. If the output is a set of equations including exponent expressions, then the combined partial solution can be represented in polynomial space, as well as the set of equations.*

Proof. The lemma follows from Lemma 5.2.5 and the arguments on the polynomial number of nodes in the DAG. □

5.3 PHASE2 Inference System: Solving Exponent Equations

In this section we solve the unification problem for sets of equations containing terms constructed over a ranked alphabet $\mathcal{F}$, a set of first-order variables, and a special expression $c^{|p|N}$, where c is a context, $p = \mathtt{hp}(c)$, and N is a variable taking values in the set of non negative integers. We assume that terms are represented using DAGs. For the purposes of this section, we assume that c is fixed. Thus p is also fixed. The expression $c^{|p|N}$ may occur several times, but it is unique. Equations over such terms are called *initial exponent equations*. This form of equations corresponds to the output of the first phase, but the output of the first phase has to be preprocessed to ensure efficiency, as explained below.

We prove that the unification problem for initial exponent equations is solvable in polynomial time, and moreover, that an explicit description of a complete set of unifiers can be computed also in polynomial time. This is done by splitting the solutions into small ones and big ones, depending on the instantiation for N. We prove the following interesting property: A set of initial exponent equations Δ has a solution if and only if it has a solution where N is replaced by a non negative integer bounded by $\mathtt{nf}(\Delta) + 2$, where $\mathtt{nf}(\Delta)$ is just the number of occurrences of function symbols in Δ, without counting the ones in expressions $C^{|p|N}$. Thus, an efficient decision algorithm is directly obtained by considering all these possible replacements and solving each of them with a fast algorithm for first-order unification.

In order to prove this bound for N, we solve a set of initial exponent equations Δ using an inference system. While the initial set Δ only contains exponent terms of the form $c^{|p|N}$, the sets derived using the inference rules can, in general, contain special expressions (of arity 1) of the form $d^{|p|N-k}$, where d is a rotation of the original context c (after flattening, see the preprocessing step below), and k is a non negative integer. The interpretation of these expressions under substitution is analogous to before, but now, $|p|N - k \geq 0$ is an implicit condition for N. Hence, when N is replaced by a non negative integer n, it should be the case that $|p|n-k \geq 0$ and $d^{|p|n-k}$ is then a context according to Definition 5.1.1. A set of equations containing this kind of expressions is called *a set of exponent equations*.

Our inference system uses multiequations instead of just equations. A *multiequation* M is a set of terms denoted as $s_1 \doteq s_2 \doteq \ldots \doteq s_{n-1} \doteq s_n$. It has the same meaning as the set of equations $s_1 \doteq s_2, s_2 \doteq s_3, \ldots, s_{n-1} \doteq s_n$. But having multiequations has some advantages from a computational point of view such as avoiding duplication of terms. In particular they avoid the substitution of non variable terms t by x, when an equation $x \doteq t$ occurs.

Our inference system deals just with *flattened terms*, that is, terms with depth at most one, and such that all the expressions of the form $d^{|p|N-k}$ satisfy that D is a *flattened context*, i.e. a context whose subterms not containing the hole have depth 0. Since our original set of equations is not necessarily flattened, we need to transform it into a flattened one, while preserving its set of solutions.

Definition 5.3.1 (Flattening) *Let Δ be a set of equations with first-order terms including also expressions of the form $d^{|p|e-k}$. We define* $\mathtt{flatten}(\Delta)$ *to be the set of equations resulting from applying the following transformation process to Δ as many times as possible.*
Flattening step*: Let t be either a proper (non variable) subterm of a term in Δ or a proper (non variable) subterm of a context d. In the latter case we assume that t does not contain the hole. Then, we create a new variable z, replace t by z everywhere in Δ, including the occurrences of t in any expression of the form $d^{|p|N-k}$, and add the equation $z \doteq t$ to Δ.*

We say that a set of equations is flattened *when the flattening step cannot be applied to it.*

The output of the first phase is a set of equations, where the terms are represented by DAGs. Applying the flattening to DAGs is exactly the same as for terms, and the results are terms, where every nonterminal in the DAG is represented by a first order variable. Flattening of contexts produces contexts where every subterm not containing the hole is a first order variable. Flattening causes an at most linear space increase.

In order to describe the inference rules, we need the following concepts of compatible contexts and of expansion of two compatible contexts.

Definition 5.3.2 (Compatible Contexts and Expansion) *Two contexts c_1 and c_2 are* compatible *if* $\mathtt{hp}(c_1) = \mathtt{hp}(c_2)$ *and* $\mathtt{sp}(c_1) = \mathtt{sp}(c_2)$.

For compatible contexts c_1 and c_2, we recursively define the set $\mathtt{expand}(c_1, c_2)$ *as follows:*

- *If $c_1 = \bullet$ and $c_2 = \bullet$, then* $\mathtt{expand}(c_1, c_2) = \emptyset$.
- *If c_1 is of the form $f(u_1, \ldots, u_{i-1}, d_1[\bullet], u_{i+1}, \ldots, u_n)$ and c_2 is of the form $f(v_1, \ldots, v_{i-1}, d_2[\bullet], v_{i+1}, \ldots, v_n)$, then* $\mathtt{expand}(c_1, c_2) = \{u_1 \doteq v_1, \ldots, u_{i-1} \doteq v_{i-1}, u_{i+1} \doteq v_{i+1}, \ldots, u_n \doteq v_n\} \cup \mathtt{expand}(d_1, d_2)$.

The PHASE2 Inference System

The idea behind the PHASE2 inference system, shown in Figure 5.2, is to simulate the usual decomposition rules for term unification, but applied to terms that may also contain expressions of the form $d^{|p|N-k}$ of arity 1. For convenience, we define the inference system on $\Delta_s; \Delta_u$, where the solved part Δ_s is a set of equations, and the unsolved part Δ_u is a set of multiequations. Initially, the solved part Δ_s is empty. The inference system operates essentially only on the unsolved part, except that certain rules may move equations from the unsolved part into the solved part.

Example 5.3.3 *Let Δ be $\{f(\bullet, y)^{N-2}[z_2] \doteq f(\bullet, z)^N[x_4]\}$. The inference rule* `NN` *is applicable with $c_1 = f(\bullet, y)$, $c_2 = f(\bullet, z)$, $|p| = 1$, $k_1 = 2$ and $k_2 = 0$. The contexts c_1 and c_2 are compatible, and* $\texttt{expand}(c_1, c_2) = \{y \doteq z\}$. *Thus, applying* `NN` *to Δ gives* $\{y \doteq z,\ \texttt{flatten}(z_2 \doteq f(\bullet, y)^2[x_4])\}$. *Figure 5.4 contains a more involved example.*

The PHASE2 inference system is intended to compute only "big" unifiers, i.e. unifiers for which the replacement for N is "big". The solutions with a small instantiation for N are computed by scanning over all small instantiations of N and then using first-order unification for each. Recall that, the goal of the PHASE2 inference system is not proving solvability of a set of exponent equations in polynomial time. This inference system is just an artifact in the proof of a periodicity lemma that leads to a polynomial time algorithm

For a set of multiequations Δ we define some measures on the unsolved part that will help in formally defining "big":

- $\texttt{nf}(\Delta)$ denotes the number of occurrences of function symbols in Δ_u without counting the ones in expressions $d^{|p|N-k}$,
- $\texttt{nx}(\Delta)$ denotes the number of occurrences of first-order variables in Δ_u,
- $\texttt{nk}(\Delta)$ denotes the sum of all the k's for all the occurrences of expressions $d^{p|N|-k}$ in Δ_u, and
- $\texttt{nc}(\Delta)$ denotes the number of occurrences of expressions $d^{|p|N-k}$ in Δ_u.

Let Δ_0 be some fixed set of initial exponent multiequations. Recall that c and $p = \texttt{hp}(c)$ are fixed. Define $B(\Delta_0) := 2 + \dfrac{\texttt{nf}(\Delta_0)}{|p|}$.

Definition 5.3.4 (Big Unifiers) *Let Δ be a set of exponent equations derived from the (implicitly assumed) set Δ_0. We say σ is a* big unifier *of Δ if σ is a solution of Δ and $\sigma(N) \geq B(\Delta_0)$.*

$$\text{(xx)} \quad \frac{\Delta_s; \Delta \cup \{x \doteq M_1, x \doteq M_2\}}{\Delta_s; \Delta \cup \{x \doteq M_1 \doteq M_2\}}$$

$$\text{(xy)} \quad \frac{\Delta_s; \Delta \cup \{x \doteq y \doteq M\}}{\Delta_s \cup \{x \doteq y\}; \langle x \mapsto y\rangle(\Delta \cup \{y \doteq M\})}$$

if $x \neq y$

$$\text{(xM)} \quad \frac{\Delta_s; \Delta \cup \{x \doteq t \doteq M\}}{\Delta_s \cup \{x \doteq t\}; \Delta \cup \{t \doteq M\}}$$

if x does not occur in Δ, M, t

$$\text{(alone)} \quad \frac{\Delta_s; \Delta \cup \{t\}}{\Delta_s; \Delta}$$

$$\text{(ff)} \quad \frac{\Delta_s; \Delta \cup \{f(x_1, \ldots, x_m) \doteq f(y_1, \ldots, y_m) \doteq M\}}{\Delta_s; \Delta \cup \{f(y_1, \ldots, y_m) \doteq M, x_1 \doteq y_1, \ldots, x_m \doteq y_m\}}$$

$$\text{(NN)} \quad \frac{\Delta_s; \Delta \cup \{c_1^{|p|N-k_1}(x_1) \doteq c_2^{|p|N-k_2}(x_2) \doteq M\}}{\begin{array}{l}\Delta_s; \Delta \cup \{c_2^{|p|N-k_2}(x_2) \doteq M\} \cup \texttt{expand}(c_1, c_2) \cup \\ \quad \texttt{flatten}(\{x_1 \doteq (\texttt{rot}(c_2, k_1'))^{k_1-k_2}(x_2)\})\end{array}}$$

if c_1,c_2 are compatible, where $k_1 \geq k_2$, and $k_1' = |p| - k_1 \texttt{ mod } |p|$.

$$\text{(Nf)} \quad \frac{\Delta_s; \Delta \cup \{f(x_1, \ldots, x_m) \doteq (f(y_1, \ldots, y_{i-1}, c_1, y_{i+1}, \ldots, y_m))^{|p|N-k}(x)\}}{\begin{array}{l}\Delta_s; \Delta \cup \{x_1 \doteq y_1, \ldots, x_{i-1} \doteq y_{i-1}, x_{i+1} \doteq y_{i+1}, \ldots, x_m \doteq y_m\} \cup \\ \{x_i = (c_1[f(y_1, \ldots, \bullet, \ldots, y_m)])^{|p|N-k-1}(x)\}\end{array}}$$

Figure 5.2: The PHASE2 unification rules for multiequations

Example 5.3.5 *Let* Δ *be* $\{f(x) \doteq y, (g(f(\bullet)))^{2N-2}(y) \doteq g(z) \doteq (g(f(\bullet)))^{2N-5}(w)\}$*, where* x, y, z, w *are first-order variables. Then,* $\texttt{nf}(\Delta) = 2$*,* $\texttt{nc}(\Delta) = 2$*,* $\texttt{nk}(\Delta) = 7$ *and* $\texttt{nx}(\Delta) = 5$*. Here,* $c = g(f(\bullet))$ *and* $|p| = 2$*.*

We shall sometimes use the measures `nf`, `nx`, `nk` and `nc` without explicitly showing their argument (Δ), which either means the measures are being used as functions, or the argument is clear by the context.

(cycle) $$\frac{\Delta_s; \Delta \cup \{x_1 \doteq t_1 \doteq M_1\} \cup \ldots \cup \{x_n \doteq t_n \doteq M_n\}}{\bot}$$
if $t_n \neq x_1 \in \mathtt{V}(t_n), t_1 \neq x_2 \in \mathtt{V}(t_1), \ldots, t_{n-1} \neq x_n \in \mathtt{V}(t_{n-1})$.

(clash) $$\frac{\Delta_s; \Delta \cup \{s \doteq t \doteq M\}}{\bot}$$
if $\mathtt{root}(s), \mathtt{root}(t) \in \mathcal{F}$ and $\mathtt{root}(s) \neq \mathtt{root}(t)$

(clashfC) $$\frac{\Delta_s; \Delta \cup \{f(\ldots) \doteq (g(\ldots, c_1, \ldots))^{|p|N-k}(x) \doteq M\}}{\bot}$$
if $f \neq g$

(clashCC) $$\frac{\Delta_s; \Delta \cup \{c_1^{|p|N-k_1}(x_1) \doteq c_2^{|p|N-k_2}(x_2) \doteq M\}}{\bot}$$
if c_1, c_2 are not compatible

Figure 5.3: The PHASE2 unification failure rules for multiequations

Termination

We first show that any derivation using the above inference rules terminates in a polynomial number of steps. The following lemma follows by inspecting the inference rules.

Lemma 5.3.6 *Let* $\Delta_1, \Delta_2, \ldots, \Delta_n$ *be a* PHASE2 *derivation. Then,* $\mathtt{nf}(\Delta_n) + \mathtt{nk}(\Delta_n) \leq \mathtt{nf}(\Delta_1) + \mathtt{nk}(\Delta_1)$*.*

Proof. It suffices to see that $\mathtt{nf} + \mathtt{nk}$ is either preserved or decreased after every rule application. This is easy by inspecting the inference rules. Rules (xx), (xy) and (xM) preserve $\mathtt{nf}$ and $\mathtt{nk}$. Rule (alone) either preserves or reduces $\mathtt{nf}$ and $\mathtt{nk}$. Rule (ff) reduces $\mathtt{nf}$ and preserves $\mathtt{nk}$. Rule (Nf) reduces $\mathtt{nf}$ by 1 and increases $\mathtt{nk}$ by 1. For the case of rule (NN), note that $\mathtt{expand}(c_1, c_2)$ does not add any function symbol. Note also that the addition of the term $(\mathtt{rot}(c_2, k_1'))^{k_1-k_2}(t_2)$ increases $\mathtt{nf}$ by $k_1 - k_2$, while $\mathtt{nk}$ is reduced by k_1 since the term $c_1^{|p|N-k_1}(x_1)$ is removed. □

Corollary 5.3.7 *Let* $\Delta_0, \Delta_1, \ldots, \Delta_n$ *be a* PHASE2 *derivation starting from the initial set* Δ_0 *(note that* $\mathtt{nk}(\Delta_0)$ *is* 0*). Then,* $\mathtt{nk}(\Delta_n) + \mathtt{nf}(\Delta_n) \leq \mathtt{nf}(\Delta_0)$*,* *and* $2 + \dfrac{(\mathtt{nk}(\Delta_n) + \mathtt{nf}(\Delta_n))}{|p|} \leq B(\Delta_0)$.

Lemma 5.3.8 *The* PHASE2 *inference system terminates. The number of inference steps of a derivation starting from a given starting set of flattened exponent equations* Δ *is bounded by* $(2 * \mathtt{nc} * (\mathtt{nf} + \mathtt{nk}) + \mathtt{nx})(\Delta)$.

Proof. For termination, it suffices to see that the tuple $\langle \mathtt{nc}, \mathtt{nf}, \mathtt{nx} \rangle$ decreases after every rule application, where tuples are compared using the lexicographic extension of the usual ordering on integers. This is trivial by inspecting the inference rules. Rules (xx), (xy) and (xM) preserve $\mathtt{nc}$ and $\mathtt{nf}$, but they reduce $\mathtt{nx}$. Rule (alone) reduces some of $\mathtt{nc}, \mathtt{nf}, \mathtt{nx}$, and either reduces or preserves the rest. Rules (ff) and (Nf) reduce $\mathtt{nf}$ and preserve $\mathtt{nc}$ and $\mathtt{nx}$. For the case of rule (NN), we recall that $(\mathtt{rot}(c_2, k_1'))^{k_1 - k_2}$ is a first order context and not an expression of the form $d^{|p|N-k}$. Hence, this rule always reduces $\mathtt{nc}$.

Now, for the termination measure, we note that the only rule which adds new occurrences of variables is rule (NN), due to the application of the flattening process. In this case, the number of newly added variables is bounded by $\mathtt{nk}$, and by Lemma 5.3.6, this is bounded by $(\mathtt{nf} + \mathtt{nk})(\Delta)$, for the given original set of multiequations Δ. Once this occurs, $\mathtt{nc}$ is reduced by 1. Hence, the total number of newly added variables along the inference process is bounded by $(\mathtt{nc}(\mathtt{nf} + \mathtt{nk}))(\Delta)$. Therefore, $(\mathtt{nx} + \mathtt{nc}(\mathtt{nf} + \mathtt{nk}))(\Delta)$ is a bound for the number of times the third component of $(\mathtt{nc}, \mathtt{nf}, \mathtt{nx})$ is reduced along this process. Again by Lemma 5.3.6, the two first components are reduced $(\mathtt{nc} * (\mathtt{nf} + \mathtt{nk}))(\Delta)$ times at most. This gives the bound $(\mathtt{nc} * (\mathtt{nf} + \mathtt{nk}) + (\mathtt{nx} + \mathtt{nc} * (\mathtt{nf} + \mathtt{nk})))(\Delta)$ for the total number of inference steps. □

Correctness

The following property of big unifiers will allow us to justify compatibility of contexts of certain equations below.

Corollary 5.3.9 *In any* PHASE2 *derivation* $\Delta_0, \Delta_1, \ldots$, *if* $d^{|p|N-k}$ *occurs in any* Δ_i *and* σ *is a big unifier of* Δ_i, *then* $\sigma(|p|N - k) \geq 2|p|$.

Proof. If $d^{|p|N-k}$ occurs in Δ_i, then note that $\sigma(|p|N - k) = |p|\sigma(N) - k \geq |p|(2 + \mathtt{nf}(\Delta_0)/|p|) - k = 2|p| + \mathtt{nf}(\Delta_0) - k \geq 2|p| + \mathtt{nf}(\Delta_i) + \mathtt{nk}(\Delta_i) - k \geq 2|p|$, using Lemma 5.3.6 and Definition 5.3.4. □

The soundness of the PHASE2 inference system follows directly from inspecting the rules and applying Corollary 5.3.9.

Lemma 5.3.10 *(Soundness) Let $\Delta_1 \vdash \Delta_2$ be an inference step, and let σ be a big unifier of Δ_2. Then, σ is also a big unifier of Δ_1.*

Proof. For all rules, a big unifier σ of Δ_2 is also a big unifier of Δ_1, by inspecting the rules. It is crucial to take into account the solved part of Δ_2 for the rules (xy) and (xM). The detailed proof is straightforward, but tedious. We illustrate it for the rule (NN): Let σ be a big unifier of $\Delta \cup \{c_2^{|p|N-k_2}(x_2) \doteq M\} \cup \texttt{expand}(c_1, c_2) \ \cup \texttt{flatten}(\{x_1 \doteq (\texttt{rot}(C_2, k'_1))^{k_1-k_2}(x_2)\})$ where c_1,c_2 are compatible, $k_1 \geq k_2$ and $k'_1 = |p| - k_1 \ \texttt{mod}\ |p|$. We only need to prove that σ is a unifier of $c_1^{|p|N-k_1}(x_1) \doteq c_2^{|p|N-k_2}(x_2)$. Since σ is big, it follows from Corollary 5.3.9 that $\sigma(|p|N - k_1) > 0$ and $\sigma(|p|N - k_2) > 0$. Since σ unifies $\texttt{expand}(c_1, c_2)$ and c_1 and c_2 are compatible, we can conclude that $\sigma(c_1) = \sigma(c_2)$. Since σ is a unifier of $x_1 \doteq (\texttt{rot}(c_2, k'_1))^{k_1-k_2}(x_2)$, we see that σ is also a unifier of $c_1^{|p|N-k_1}(x_1)$ and $c_1^{|p|N-k_1}(\texttt{rot}(c_2, k'_1))^{k_1-k_2}(x_2)$. The term $\sigma(c_1^{|p|N-k_1}(\texttt{rot}(C_2, k'_1))^{k_1-k_2}(x_2))$ is equal to $\sigma(c_2^{|p|N-k_2}(x_2))$ using properties of $\texttt{rot}$ and exponents, and the fact that $\sigma(c_1) = \sigma(c_2)$. □

Completeness of the inference system of Phase 2 depends on the following lemma that shows compatibility of two contexts c_1, c_2 that occur in exponent equations that are, roughly speaking, of the form $c_1[c_1[\ldots]] \doteq c_2[c_2[\ldots]]$.

Lemma 5.3.11 *Let c_1, c_2 be two flattened contexts such that each one is a rotation of the other. Let t_1, t_2 be terms. Let k_1, k_2 be non-negative integers greater than or equal to $2|p|$. If $c_1^{k_1}(t_1) \doteq c_2^{k_2}(t_2)$ has a solution, then c_1 and C_2 are compatible.*

Proof. We prove this by contradiction, i.e. under the assumption of incompatibility of c_1 and c_2 we prove that $c_1^{k_1}(t_1) \doteq c_2^{k_2}(t_2)$ has no solution.

Let q be the maximum position that is a prefix of $\texttt{hp}(c_1)$ and $\texttt{hp}(c_2)$. If $|q| = |\texttt{hp}(c_1)| = |\texttt{hp}(c_2)|$, then $\texttt{hp}(c_1) = \texttt{hp}(c_2)$. In this case, the incompatibility of c_1 and c_2 implies $\texttt{sp}(c_1) \neq \texttt{sp}(c_2)$, from which we conclude that $c_1^{k_1}(t_1) \doteq c_2^{k_2}(t_2)$ has no solution. Hence, assume that $q < \texttt{hp}(c_1)$ and $q < \texttt{hp}(c_2)$. If some $q' \leq q$ satisfies $\texttt{root}(c_1|_{q'}) \neq \texttt{root}(c_2|_{q'})$, then $c_1^{k_1}(t_1) \doteq c_2^{k_2}(t_2)$ has no solution again. Therefore, assume $\texttt{root}(c_1|_{q'}) = \texttt{root}(c_2|_{q'})$ for all $q' \leq q$. The equation $c_1^{k_1}(t_1) \doteq c_2^{k_2}(t_2)$ has solvability of $\texttt{rot}(c_1, |q|)^{k_1-|q|}(t_1) \doteq \texttt{rot}(c_2, |q|)^{k_2-|q|}(t_2)$ as a necessary condition. We write $\texttt{rot}(c_1, |q|)$ and $\texttt{rot}(c_2, |q|)$ more explicitly of the form $f(x_1, \ldots, x_{i-1}, d_1, x_{i+1}, \ldots, x_m)$ and $f(y_1, \ldots, y_{j-1}, d_2, y_{j+1}, \ldots, y_m)$, respectively. Note that, by the maximality of q, the indexes i and j are different. Any solution σ of $c_1^{k_1}(t_1) \doteq c_2^{k_2}(t_2)$, and hence of $\texttt{rot}(c_1, |q|)^{k_1-|q|}(t_1) \doteq \texttt{rot}(c_2, |q|)^{k_2-q}(t_2)$, makes $\sigma(y_i)$ equal to $\sigma(d_1[t_1])$. But since k_1 is greater than or equal to $2|p|$ and c_1 is a rotation of c_2, D_1 properly contains the variable y_i as well, and hence $\sigma(y_i)$ is a proper subterm of itself, a contradiction. □

Lemma 5.3.12 *Let Δ be a set of exponent multiequations with $\Delta_u \neq \emptyset$, such that (cycle), (clash), (clashfC) or (clashCC) can be applied to it. Then Δ does not have any big unifier.*

Proof. Assume that Δ has a big unifier. We show that (cycle) or (clash) cannot be applicable. From Corollary 5.3.9, we know that the existence of a big unifier implies that for every expression $d^{|p|N-k}(s)$, the exponent is at least $2|p|$, and hence the context is at least $d[d[\ldots[\bullet]\ldots]]$.

For all multiequations of the form $s \doteq t \doteq M$ where s, t are rooted by function symbols, we must have $\mathtt{root}(s) = \mathtt{root}(t)$. Hence (clash) is not applicable. However, (cycle) is not applicable, either. The reason is that the existence of equations $x_1 \doteq t_1 \doteq M_1, x_2 \doteq t_2 \doteq M_2, \ldots, x_{n-1} \doteq t_{n-1} \doteq M_{n-1}, x_n \doteq t_n \doteq M_n$ in Δ such that x_2 occurs in t_1,..., x_n occurs in t_{n-1}, and x_1 occurs in t_n together with the existence of a big unifier would imply that $\sigma(x_1)$ is a proper subterm of $\sigma(x_1)$, which is impossible. If there is a multiequation of the form $c_1^{|p|N-k_1}(x_1) \doteq c_2^{|p|N-k_2}(x_2) \doteq M$, then by Lemma 5.3.11 c_1, c_2 are compatible, hence (clashCC) is also not applicable. If there is a multiequation of the form $f(\ldots) \doteq (g(\ldots, D[\bullet], \ldots))^{|p|N-k}(x) \doteq M$, then the root of $\sigma(g(\ldots, d, \ldots))^{|p|N-k}(x)$ is $g = f$ for every big unifier σ, hence (clashfC) is not applicable. From this contradiction, we conclude that Δ cannot have a big unifier. □

Lemma 5.3.13 *(Completeness for big unifiers) Let $\Delta_1 \vdash \Delta_2$ be an inference step, and let σ be a big unifier of Δ_1. Then there is an extension σ_1 of σ that is also a big unifier of Δ_2.*

Proof. For the case of rules (xx), (xy), (xM), (alone), (ff), and (Nf) the same σ serves as a big unifier.

For the case of rule (NN), the same σ serves for $\Delta \cup \{c_2^{|p|N-k_2}(x_2) \doteq M\}$. It also serves for $\mathtt{expand}(c_1, c_2)$, because, by Corollary 5.3.9, the replacement of N by $\sigma(N)$ makes $|p|N - k_1$ greater than or equal to $2p$, and hence, solvability of $\mathtt{expand}(c_1, c_2)$ is a necessary condition for solvability of Δ_1 by Lemma 5.3.11. But σ is not enough for $\mathtt{flatten}(\{x_1 \doteq (\mathtt{rot}(c_2, k_1'))^{k_1-k_2}(x_2)\})$, since it contains new variables. We just need to extend σ in the following way. Whenever the occurrences of a term t are replaced by a new variable z along the flattening process, we extend σ by $\{z \mapsto \sigma(t)\}$. The final extension of σ after the complete flattening process is a big unifier of the resulting set of multiequations. Finally, note that by Lemma 5.3.12, the remaining rules, (cycle), (clash), (clashfC) and (clashCC), are not applicable, and this completes the proof. □

Combining the soundness and completeness results for big unifiers, we conclude that the inference rules can be applied in a "don't care" manner.

Corollary 5.3.14 *If $\Delta_1 \vdash \Delta_2$ then every big unifier σ of Δ_1 can be extended to a big unifier of Δ_2, and every big unifier of Δ_2 is a big unifier of Δ_1.*

We finally show that the inference system is progressive, that is, if there is a solution and $\Delta_u \neq \emptyset$, then we can apply some rule.

Proposition 5.3.15 *Let Δ_0 be a set of initial exponent multiequations. Then the following are equivalent:*
(A) there is a PHASE2 *derivation $\Delta_0 \vdash \Delta_1 \vdash \ldots \vdash \Delta_n$ with $\Delta_{n,u} = \emptyset$,*
(B) Δ_0 has a big unifier, and
(C) for all $n \geq B(\Delta_0)$, Δ_0 has a solution σ with $\sigma(N) = n$.

Proof.
(A)⇒(C): If $\Delta_0 \vdash \Delta_1 \vdash \ldots \vdash \Delta_n$ is a maximal derivation with $\Delta_{n,u} = \emptyset$, then the final Δ_n has a solution: $\Delta_{n,s}$ can be arranged in the form $x_1 \doteq t_1, x_2 \doteq t_2, \ldots, x_m \doteq t_m$, such that all x_i are different and such that x_i is not contained in any t_j with $j \geq i$. Hence a solution can be computed by iterated instantiation. Since none of our inference rules instantiate N, we can set $\sigma(N) := n'$ for any $n' \geq B(\Delta_0)$. By Lemma 5.3.10, each of these solutions for Δ_n are also solutions for Δ_0.
(C)⇒(B): This is straightforward.
(B)⇒(A): If Δ_0 has a big unifier, then let $\Delta_0 \vdash \Delta_1 \vdash \ldots \vdash \Delta_n$ be a maximal derivation. By Lemma 5.3.13, Δ_n has a big unifier. The assumption that (clash), (clashfC) and (clashCC) are not applicable implies that for any $s \doteq t \in \Delta_{n,u}$ such that s, t are not variables, one of the rules (ff), (NN), (Nf) could be applied. Since these rules are not applicable, this implies that multiequations in $\Delta_{n,u}$ must be of the form $x \doteq t$, and the variable x occurs in t or somewhere else in $\Delta_{n,u}$. Since (cycle) is not applicable, this is in conflict with the non-applicability of the rule (xM). Hence the final set $\Delta_{n,u}$ must be empty.

□

Complexity

An important consequence of Proposition 5.3.15 is the following corollary. The following corollary is a direct consequence of Proposition 5.3.15.

Corollary 5.3.16 *A flattened set of initial exponent multiequations Δ_0 has a big unifier if and only if the first-order problem $\{N \mapsto \lceil B(\Delta_0) \rceil\}(\Delta_0)$ is unifiable.*

Therefore, for deciding the existence of big unifiers of a given Δ_0, it suffices to ask for the solution of the first-order unification problem

$\{N \mapsto \lceil(\texttt{nf})(\Delta_0)/p + 2\rceil\}\Delta_0$, which can be solved efficiently. Hence, for deciding the existence of any (big or small) solution we just need to consider several first-order unification problems obtained by substituting N by $0, 1, 2, \ldots, \lceil(\texttt{nf})(\Delta_0)/p + 2\rceil$.

Theorem 5.3.17 *The unification problem for flattened exponent multiequations is solvable in* $\texttt{O}(n^3\texttt{log}(n))$ *time.*

Proof. By iteratively instantiating N by $0, 1, \ldots, \lceil B(\Delta_0)\rceil$, and expanding the exponents in each case, we get $O(n)$ first-order unification problems of size $O(n^2)$ each. Since first-order unifiability of multiequations of size n is decidable in $O(n * \texttt{log}(n))$ time [MM82, PW78], we get the desired result. □

Combining Theorem 5.3.17 and the process of Phase 1, which runs in nondeterministic polynomial time, yields the following result.

Theorem 5.3.18 *One context unification is in NP.*

From the first phase, we know that any set of equations of one context unification containing an equation of the form $F(s) = c[F(t)]$ can be transformed into a unification problem of exponent flattened multiequations by guessing just *once* over a *linear* number of possibilities. Hence, we obtain the following.

Theorem 5.3.19 *The unification problem for one context term equations containing an equation of the form* $F(s) = c[F(t)]$*, where* c *is a non empty context, is solvable in polynomial time.*

We can also construct a complete set of unifiers for Δ_0 following a similar approach. In the second phase, for unifiers σ where $\sigma(N) < B(\Delta_0)$, we simply solve the term unification problems obtained by setting $\langle N \mapsto n\rangle$ for every $n < B(\Delta_0)$; and for unifiers σ where $\sigma(N) \geq B(\Delta_0)$, we use the PHASE2 inference system to compute Δ_n. By Lemma 5.3.8, this runs in polynomial time. If $\Delta_{n,u} = \emptyset$, then $\Delta_{n,s}$ represents exactly *all* big unifiers of Δ_0 since, by Corollary 5.3.14, we know that solutions are not lost. We get the following analogues of Theorem 5.3.17 and Theorem 5.3.18.

Theorem 5.3.20 *Let* Δ *be a set of initial exponent equations. Then, a complete set of unifiers for* Δ *with polynomially many unifiers (representable in polynomial space) can be generated in polynomial time.*

Combining the previous theorem with the results in the previous section we obtain the following.

Theorem 5.3.21 *Given an instance Δ of the one context unification problem, a complete set of most general unifiers for Δ can be generated in exponential time. Each unifier is represented in polynomial space using mappings from variables to exponent-terms.*

We will give a polynomial complexity bound, if the number of first-order variables is fixed. Although all the claims and proofs we have given above are valid by assuming that the input is already represented by a DAG structure, the following result requires the input to be given without any compression, i.e. the terms are represented with a size proportional to the size of such terms (as trees).

Theorem 5.3.22 *If the number of first-order variables is fixed (say k), then one context unification is solvable in polynomial time.*

Proof. Assume for the purposes of this proof that we implement phase 1 without DAGS, i.e. no reuse of equal subterms occurs. Then, an assignment for a variable produces as many copies of the assigned term as the number of occurrences of this variable in the equations. If the number k of first order variables is fixed, then the size increase of the equations and terms by the algorithm in phase 1 without DAGs remains polynomial: Instantiation by rules `Var-Elim` and `Var-Elim2` does at most square the size, which happens at most $k + 1$ times. I.e. if n is the size of the input, then the size (as terms without using DAGs) of the equations and substitutions during the whole phase 1 is at most $n^{2^{k+1}}$. Now it is sufficient to argue that there is a polynomial upper bound for the maximal total number of necessary guessing possibilities in phase 1, since we have already established polynomiality of phase 2, and since phase 1 computation is in NP. The rules `Decompose` and `Var-Elim` are don't care non-deterministic and thus it is not necessary to explore alternatives. The possibilities of the rules `Var-Elim2`, `CVar-Elim`, and `CVar-Elim2` have to be counted. They are applied at most $k+1$ times. Every rule has at most $\left(n^{2^{k+1}}\right)^2$ possibilities, due to the selection of the context C. Thus we have shown that there is a polynomial number of possibilities, and every possibility can be completely checked including phase 2 in polynomial time. In summary, we have shown the claim of the theorem. □

$$\texttt{Var-Elim2:}\quad \frac{\{f(x) \doteq F[f(y)], f(f(x)) \doteq F[f(f(y))], f(f(f(x))) \doteq F[f(f(f(y)))]\}}{\left\langle \begin{array}{c} x \mapsto F'[f(y)], \\ F \mapsto f(F'[\bullet]) \end{array} \right\rangle \left\{ \begin{array}{c} f(f(F'[f(y)])) \doteq f(F'[f(f(y))]), \\ f(f(f(F'[f(y)]))) \doteq f(F'[f(f(f(y)))]) \end{array} \right\}}$$

$$\texttt{Decompose:}\quad \overline{\{f(F'[f(y)]) \doteq F'[f(f(y))], f(f(f(F'[f(y)]))) \doteq f(F'[f(f(f(y)))])\}}$$

$$\texttt{CVar-Elim2:}\quad \overline{\langle F' \mapsto f^N[\bullet]\rangle \;\; \{f(f(f(f^N[f(y)]))) \doteq f(f^N[f(f(f(y)))])\}}$$

$$\texttt{Flatten:}\quad \overline{\left\{ \begin{array}{c} f(x_1) \doteq f(z_1), x_1 \doteq f(x_2), x_2 \doteq f(x_3), x_3 \doteq f^N[x_4], x_4 \doteq f(y), \\ z_1 \doteq f^N[z_2], z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\}}$$

$$\texttt{ff:}\quad \overline{\left\{ \begin{array}{c} x_1 \doteq z_1 \doteq f(x_2) \doteq f^N[z_2], x_2 \doteq f(x_3), x_3 \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\}}$$

$$\texttt{xy,xM:}\quad \overline{\left\langle \begin{array}{c} x_1 \mapsto z_1, \\ z_1 \mapsto f(x_2) \end{array} \right\rangle \left\{ \begin{array}{c} f(x_2) \doteq f^N[z_2], x_2 \doteq f(x_3), x_3 \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\}}$$

$$\texttt{Nf:}\quad \overline{\left\{ \begin{array}{c} x_2 \doteq f^{N-1}[z_2] \doteq f(x_3), x_3 \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\}}$$

$$\texttt{Nf:}\quad \overline{\langle x_2 \mapsto f(x_3)\rangle \left\{ \begin{array}{c} x_3 \doteq f^{N-2}[z_2] \doteq f^N[x_4], \\ x_4 \doteq f(y), z_2 \doteq f(z_3), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\}}$$

$$\texttt{NN:}\quad \overline{\langle x_3 \mapsto f^{N-2}[z_2]\rangle \left\{ \begin{array}{c} z_2 \doteq f^2[x_4] \doteq f(z_3), \\ x_4 \doteq f(y), z_3 \doteq f(z_4), z_4 \doteq f(y) \end{array} \right\}}$$

$$\texttt{xM:}\quad \overline{\langle z_2 \mapsto f^2[x_4]\rangle \;\; \{z_3 \doteq f(x_4) \doteq f(z_4), x_4 \doteq f(y), z_4 \doteq f(y)\}}$$

$$\texttt{xM:}\quad \overline{\langle z_3 \mapsto f(x_4)\rangle \;\; \{x_4 \doteq z_4 \doteq f(y) \doteq f(y)\}}$$

$$\texttt{xM:}\quad \overline{\langle x_4 \mapsto z_4, z_4 \mapsto f(y)\rangle}$$

Figure 5.4: Example illustrating the inference rules of the two phases. Some trivial applications are not shown, and we only show the insertions into the solved part.

Chapter 6

Compressed One Context Unification

In this chapter we extend the result of the previous chapter, one context unification is in NP, to the case where the terms in the given set of equations are represented using STGs.

By considering that the input terms are compressed using STGs, we obtain *one context unification with STGs*. Fixed a ranked alphabet $\mathcal{F}$, an *instance* of this problem is a tuple $\langle \Delta, G, \mathcal{X}, F \rangle$, where $\mathcal{X}$ is a set of first-order variables, F is a context variable, $G = \langle \mathcal{TN}, \mathcal{CN}, \mathcal{F} \cup \mathcal{X} \cup \{F\}, R \rangle$ is an STG, and Δ is a set of equations of the form $\{A_1 \doteq B_1, \ldots, A_n \doteq B_n\}$, where the A_is and the B_is are term nonterminals of G. With this representation, the context variable and first-order variables are initially represented as unary and constant terminal symbols of the grammar, respectively. Given an instance $\langle \{A_1 \doteq B_1, \ldots, A_n \doteq B_n\}, G, \mathcal{X}, F \rangle$, its corresponding uncompressed one context unification instance is $\langle \{w_{G,A_1} \doteq w_{G,B_1}, \ldots, w_{G,A_n} \doteq w_{G,B_n}\}, \mathcal{X}, F \rangle$.

In order to prove that one context unification is also in NP in the STG-compressed case, we adapt the inference system described in Chapter 5. Since subterms, subcontexts, context exponentiation, and variable instantiations are known to be efficiently computable with STGs, as seen in Chapter 3, one may be tempted to simply reproduce the sequence of variable instantiations resulting of the inference system of the previous chapter. However, the size of the grammar may grow after certain operations and, in order to prove that it does not explode, we need to use a different approach. In particular, the difficulties rely on the fact that we do not have a bound on the size of the grammar after successive partial instantiations of the context variable, which require to compute a subcontext and thus increase the size of the grammar.

Our algorithm is built upon some of the constructions described in Section 3.2.1 of Chapter 3 as well as the results from Chapter 5 and Chapter 4. However, in the following section we restate some of the needed results in a more convenient way to ease the presentation

6.1 Known Results

In this section we restate some of the results presented in previous chapters regarding operations on STGs and solvability in nondeterministic polynomial time of the one context unification problem.

6.1.1 Operations on STGs

In this section we describe the operations needed to compute a solution to an STG-compressed one context unification instance. The following definition of *extension of an STG* describes the result of those operations.

Definition 6.1.1 *Let $\mathcal{F}$ be a ranked alphabet. Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$ be an STG such that $\mathcal{F} \subseteq \Sigma$. An $\mathcal{F}$-*extension *of G is an STG $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma', R'\rangle$ such that $\mathcal{TN} \subseteq \mathcal{TN}'$, $\mathcal{CN} \subseteq \mathcal{CN}'$, $R \subseteq R'$, and $\mathcal{F} \subseteq \Sigma'$.*

The goal of the previous definition is to capture operations on STGs that correspond to instantiation of variables. More concretely, let $G = \langle \mathcal{TN}, \mathcal{CN}, \mathcal{F} \cup \mathcal{V}, R\rangle$ be an STG and let $G' = \langle \mathcal{TN}', \mathcal{CN}', \mathcal{F} \cup \mathcal{V}', R'\rangle$ be an $\mathcal{F}$-extension of G. The terms generated by term nonterminals of G and G' are assumed to belong to the sets $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{F}, \mathcal{V}')$, respectively, and analogously for contexts. Hence, by defining a substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V}') \cup \mathcal{C}(\mathcal{F}, \mathcal{V}')$ as $\sigma(\alpha) = w_{G',\alpha}$ it holds that, for each nonterminal N of G, $\sigma(w_{G,N}) = w_{G',N}$.

In Chapter 4, we saw that first-order unification can be solved in polynomial time. Recall that the algorithm described in Figure 4.1 proceeds analogously to the classical unification process: Given a first-order equation $s \doteq t$, the algorithm iteratively finds a position p labeled by a variable x in one of the terms, say s, and replaces all its occurrences by the corresponding subterm $t|_p$. This process ends when either s and t become equal, and thus they are unifiable, or a contradiction is reached. Since each iteration of the algorithm instantiates a variable, the solution σ can be described as an ordered sequence of substitutions on first-order variables. We already proved in Chapter 4 that this behaviour can be efficiently adapted to the STG-compressed setting. Besides checking equality at each iteration, the only operation that the algorithm needs to perform is to apply substitutions of the form $\{x \mapsto t|_p\}$. This corresponds to computing an $\mathcal{F}$-extension of the grammar that adds n new nonterminals $A_1, \ldots, A_n$ such that A_n generates $t|_p$ using $A_1, \ldots, A_{n-1}$ and adds a new rule $x \to A_n$, i.e. converting the terminal symbol x into a nonterminal generating $t|_p$. The crucial result to prove that such an $\mathcal{F}$-extension can be computed in polynomial time is that all such n are linearly bounded by the size of the initial input grammar and not by the size of the current grammar at each step of the process. Note that this implies that the size of the final grammar, i.e. the grammar obtained after all the variables have been replaced, is polynomially bounded by the size of the initial grammar. This technical fact is summarized in the following lemma, which is a consequence of Lemma 4.1.7 proven in Chapter 4.

Lemma 6.1.2 *Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R \rangle$ be an STG describing a one context unification instance. Let A be a term nonterminal of G and let t_0 be $w_{G,A}$. Let $t_1, \ldots, t_n$ be terms, $x_1, \ldots, x_n$ be first-order variables, $p_1, \ldots, p_n$ be positions, and $\sigma_1, \ldots, \sigma_n$ be substitutions satisfying, for $i \in \{1, \ldots, n\}$, $p_i \in \mathsf{Pos}(t_{i-1})$, $x_i \in \mathsf{Vars}(t_{i-1}) \setminus \mathsf{Vars}(t_{i-1}|_{p_i})$, $\sigma_i = \{x_i \mapsto t_{i-1}|_{p_i}\}$, and $t_i = \sigma_i(t_{i-1})$.*

Then, there exists an $\mathcal{F}$-extension $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma \setminus \{x_1, \ldots, x_n\}, R' \rangle$ of G such that $w_{G',A} = t_n$ and $|G'| \leq |G| + n(|G| + 1)$.

In our current setting we also have to deal with the context variable F. Similarly to the first-order case, instantiation of the special context variable F is performed by adding a rule $F \to C$ to the grammar, i.e. turning the terminal unary symbol F into a context nonterminal. This C is a new context nonterminal defined through the concatenation of several subcontexts of the input terms. In Chapter 3, we already saw how to compute subcontexts from a given STG-compressed term. As stated in Lemma 3.2.31, the `pCon` construction guarantees that, given a grammar G, a nonterminal A and a position $p_1.p_2 \in \mathsf{Pos}(w_{G,A})$, G can be extended with, at most, $|G|(2|G|+3)$ new nonterminals such that one of them generates the context $w_{G,A}|_{p_1}[\bullet]_{p_2}$. Moreover, given the context nonterminals $C_1, \ldots, C_n$ of a grammar G, an extended grammar containing a context nonterminal that generates the concatenation $w_{G,C_1} \ldots w_{G,C_n}$ can be easily obtained by adding n new nonterminals to G. These two facts are stated in the following lemma.

Lemma 6.1.3 *Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R \rangle$ be an STG describing a one context unification instance. Let A be a term nonterminal of G and let t be $w_{G,A}$. Let $p_1, \ldots, p_n, \hat{p}_1, \ldots, \hat{p}_n$ be positions and $c_1, \ldots, c_n$ be contexts satisfying, for $i \in \{1, \ldots, n\}$, $p_i.\hat{p}_i \in \mathsf{Pos}(t)$ and $c_i = t|_{p_i}[\bullet]_{\hat{p}_i}$.*

Then, there exists an $\mathcal{F}$-extension $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma, R' \rangle$ of G with a context nonterminal C such that $w_{G',C} = c_1 \ldots c_n$ and $|G'| \leq |G| + n|G|(2|G| + 3) + n$.

As seen in the previous chapter, in some cases, the context nonterminal C that instantiates F is defined using one last construction: context exponentiation. Computing a succinct grammar that generates the result of this operation is straightforward, as shown in the following illustrative example.

Example 6.1.4 *Let G be an STG with the following set of rules: $\{A_g \to g(A_h), A_h \to h(A_a), A_a \to a\}$. Note that w_{G,A_g} is $g(h(a))$. The context exponentiation $(w_{G,A_g}|_\lambda[\bullet]_{1.1})^{11}$ is $g(h(g(h(g(h(g(h(g(h(g(\bullet)\ldots)$ and can be generated by the STG with the set of rules:*

$$\begin{aligned} &\{C \to g(C_h), C_h \to h(C_\bullet), C_\bullet \to \bullet\} \\ \cup\ &\{C_{\mathsf{exp4}} \to CC, C_{\mathsf{exp8}} \to C_{\mathsf{exp4}}C_{\mathsf{exp4}}, C_{\mathsf{exp10}} \to C_{\mathsf{exp8}}C\} \\ \cup\ &\{C_{\mathsf{pref}} \to g(C_\bullet)\} \\ \cup\ &\{C_{\mathsf{exp11}} \to C_{\mathsf{exp10}}C_{\mathsf{pref}}\} \end{aligned}$$

Note that we use the nonterminals C and C_{pref} to generate two different subcontexts, $g(h(\bullet))$ and $g(\bullet)$, respectively. Moreover, the nonterminals C_{exp4}, C_{exp8}, C_{exp10}, and C_{exp11} are used for exponentiation and concatenation, with C_{exp11} generating the desired context.

As seen in the previous example, raising a context to a natural number e requires to (i) compute two different subcontexts c_1 and c_2, (ii) concatenate c_1 with itself several times, and (iii) concatenate the resulting context with c_2. The construction done in (i) adds, at most, $|G|(2|G|+3)$ for each computed subcontext, (ii) can be performed efficiently because the number of new nonterminals to be added is logarithmic with respect to e, and (iii) only adds one extra nonterminal to the grammar. This fact is stated formally in the following lemma.

Lemma 6.1.5 *Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$ be an STG describing a one context unification instance. Let A be a term nonterminal of G and let t be $w_{G,A}$. Let p_1, p_2 be positions such that $p_1.p_2 \in \mathsf{Pos}(t)$ and let $e \geq 0$ be a natural number.*

Then, there exists an $\mathcal{F}$-extension $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma, R'\rangle$ of G with a context nonterminal C such that $w_{G',C} = (t|_{p_1}[\bullet]_{p_2})^e$ and $|G'| \leq |G| + 2|G|(2|G|+3) + \lceil \log_2(e) \rceil + 1$.

6.1.2 Uncompressed One-Context Unification

Consider an instance $\langle \Delta, \mathcal{X}, F\rangle$ of the one context unification problem. In the previous chapter, it is proven that this problem is in NP when the terms in Δ are represented explicitly. The proposed algorithm is presented by means of an inference system that modifies the set of equations until a contradiction is found or the empty set is derived, which implies unifiability. By representing Δ with DAGs, the size of the representation of the terms in the set of equations is guaranteed to stay polynomially bounded by the size of the input at each step of a derivation. For the sake of clarity, we introduce in Figure 6.1 a simplified version of the algorithm of the previous chapter. In this simplified version we have erased the rules used to early detect nonunifiability and some restrictions that eased the proofs by guaranteeing a bound on the length of every derivation. Moreover, the result from Section 5.3 have been incorporated (see rule `CVar-Elim2`). Hence, the new version of the inference system is simpler and still sound. Moreover, since it is less restrictive than the original one, each derivation leading to a solution can still be performed, and thus it is complete.

$$\texttt{Decompose}: \quad \frac{\Delta = \Delta' \uplus \{\alpha(t_1,\ldots,t_n) \doteq \alpha(u_1,\ldots,u_n)\}}{\Delta' \cup \{t_1 \doteq u_1,\ldots,t_n \doteq u_n\}}$$

$$\texttt{Var-Elim}: \quad \frac{x \doteq t \in \Delta}{\{x \mapsto t\}(\Delta)}$$

where $x \notin \mathsf{Vars}(t)$

$$\texttt{Var-Elim2}: \quad \frac{F(u) \doteq c[x] \in \Delta}{\{x \mapsto F(\theta(u))\}(\theta(\Delta))}$$

where $\theta = \{F \mapsto c[F(\bullet)]\}$ and c is guessed such that $F \notin \mathsf{Vars}(c)$, $x \notin \mathsf{Vars}(u)$, and $(F \notin \mathsf{Vars}(u) \vee x \notin \mathsf{Vars}(c))$

$$\texttt{CVar-Elim}: \quad \frac{F(u) \doteq c[t] \in \Delta}{\{F \mapsto c\}(\Delta)}$$

where c is guessed such that $F \notin \mathsf{Vars}(c)$

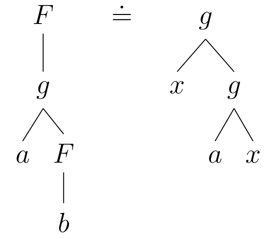
$$\texttt{CVar-Elim2}: \quad \frac{F(u) \doteq c[F(t)] \in \Delta}{\{F \mapsto c^e\}(\Delta)}$$

where c is guessed such that $F \notin \mathsf{Vars}(c)$ and e is guessed such that $0 \leq e \leq 3\sum_{s \doteq t \in \Delta}(|s| + |t|)$

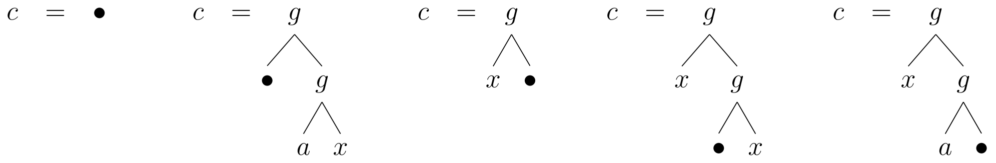
Figure 6.1: Inference system for one-context unification.

The most basic rule of the inference system is the rule `Decompose`, which is just used to simplify the unification problem. Note that it does not introduce any new subterms in the set of equations. The remaining rules modify the current set of equations by replacing variables by subterms and subcontexts constructed from the terms in the set of equations. In particular, rule `Var-Elim` replaces a first-order variable by a term, rule `Var-Elim2` partially guesses the initial part of F and instantiates a first-order variable, and rules `CVar-Elim` and `CVar-Elim2` replace F by a context. As a technical detail, note that the substitution applied due to the application of rule `Var-Elim2` can be seen as an instantiation of F in terms of a freshly introduced context variable, which, for clarity, we denote also as F. Finally, the rule `CVar-Elim2` instantiates the context variable by a context raised to a natural number whose value is linearly bounded by the size of the current set of equations. Note that it is a bound on the exponent of periodicity of minimal solutions, i.e. the maximum number of periodic repetitions of a context in a minimal solution.

Example 6.1.6 *Consider the one-context unification instance $\langle \Delta, \{x\}, F \rangle$, where Δ contains only the following equation:*



This instance has no solution. Note that neither `Decompose`*,* `Var-Elim`*, nor* `CVar-Elim2` *can be applied. In the case of* `CVar-Elim`*, we need to choose a position in* $\mathsf{Pos}(g(x, g(a, x)))$ *in order to guess a context c. Hence, there exist five different options for c:*



Note that, in any case, the resulting first-order equation after applying the substitution $\{F \mapsto c\}$ to Δ does not lead to a solution. Finally, `Var-Elim2` *cannot be applied since the left-hand side of the equation is of the form $F(u)$ and F occurs in u, and the right-hand side of the equation has two occurrences of x. In order to understand the conditions of rule* `Var-Elim2`*, note that they allow either x to occur more than once in the right-hand side or F to occur in u. The reason to consider these situations separately is that both facts cannot hold at the same time since it would lead to an instantiation of F in terms of itself, and thus a contradiction.*

By the form of the rules, the following statement bounding the length of the derivations holds trivially.

Lemma 6.1.7 *Let $\langle \Delta, \mathcal{X}, F \rangle$ be a one context unification instance. Any derivation from Δ using the inference system of Figure 6.1 contains at most $|\mathcal{X}|$ occurrences of* `Var-Elim` *and* `Var-Elim2`*, and at most one occurrence of either* `CVar-Elim` *or* `CVar-Elim2`*.*

6.2 Approach

Our approach consists of modifying the sequences of rule applications that describe a solution in order to guarantee that they can be represented in polynomial space using an STG. To simplify reasonings, we introduce in Figure 6.2 a new inference system $\mathfrak{R}$ that generalizes the previous one in Figure 6.1. In $\mathfrak{R}$ we assume without loss of generality that the initial set of equations $\Delta = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ is encoded as a single term. This can be done by extending the alphabet with new symbols $\mathfrak{d}$ and $\mathfrak{e}$ of arity n and 2, respectively, and defining $\mathsf{term}(\Delta) = \mathfrak{d}(\mathfrak{e}(s_1, t_1), \ldots, \mathfrak{e}(s_n, t_n))$. With this notion, the question of whether there exists a substitution σ, the solution for Δ, such that $\sigma(s_1) = \sigma(t_1), \ldots, \sigma(s_n) = \sigma(t_n)$ corresponds to check whether there exists a substitution σ, the solution for $\mathsf{term}(\Delta)$, such that $\sigma(\mathsf{term}(\Delta))$ is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_n, u_n))$. This change in notation is useful to refer to subterms of both sides of the equations in Δ indistinctly as subterms of $\mathsf{term}(\Delta)$.

$$\mathrm{R_x}: \quad \frac{t}{\{x \mapsto t|_p\}(t)}$$

where $p \in \mathsf{Pos}(t)$ and $x \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_p)$

$$\mathrm{R_{FF}}: \quad \frac{t}{\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t)}$$

where $p_1.p_2 \in \mathsf{Pos}(t)$ and $F \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_1})$

$$\mathrm{R_{FC}}: \quad \frac{t}{\{F \mapsto (t|_{p_1}[\bullet]_{p_2})^e\}(t)}$$

where $p_1.p_2 \in \mathsf{Pos}(t)$, $F \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_1}[\bullet]_{p_2})$, and $e \in \{0, \ldots, 3|t|\}$

Figure 6.2: The adapted inference system $\mathfrak{R}$.

It is easy to see that an application of `Var-Elim`, `Var-Elim2`, `CVar-Elim`, or `CVar-Elim2` corresponds to the application of at most two of the rules of $\mathfrak{R}$. In particular, an application of `Var-Elim` is emulated by an application of $\mathrm{R_x}$. In the rest of this chapter, we use $\mathrm{R_{xF}}$ and $\mathrm{R_{x\neg F}}$ to refer to applications of $\mathrm{R_x}$ in which the involved subterm $t|_p$ has an occurrence of F or not, respectively.

An application of `Var-Elim2` corresponds to an application of R_{FF} followed by an application of R_{xF}. Finally, applications of `CVar-Elim` and `CVar-Elim2` are emulated by R_{FC}. The remaining original rule, `Decompose`, was only used to simplify the problem in order to apply other rules. Its behaviour is implicitly emulated in $\mathfrak{R}$ by defining its rules by means of subterms and subcontexts of t.

With $\rightarrow_{\mathrm{R}_{\mathrm{xF}},x,p}$ and $\rightarrow_{\mathrm{R}_{\mathrm{x}\neg\mathrm{F}},x,p}$ we denote an application of rules R_{xF} and $\mathrm{R}_{\mathrm{x}\neg\mathrm{F}}$, respectively, making explicit the position p and the first-order variable x involved in the rule application. Analogously, with $\rightarrow_{\mathrm{R}_{\mathrm{FF}},p_1,p_2}$ and $\rightarrow_{\mathrm{R}_{\mathrm{FC}},p_1,p_2}$ we denote an application of R_{FF} and R_{FC}, making explicit the positions p_1 and p_2 involved in the rule application. Sometimes, we do not make explicit x, p, p_1, and p_2 when they are clear from the context or not relevant. By $\rightarrow_{\mathfrak{R}}$ we denote the derivational relation using $\mathfrak{R}$ and, as usual, $\rightarrow_{\mathfrak{R}}^{+}$ denotes its transitive closure and $\rightarrow_{\mathfrak{R}}^{*}$ denotes its reflexive-transitive closure. Additionally, by $\rightarrow_r$ and $\rightarrow_r^*$, we denote the derivational relation using the rule r of $\mathfrak{R}$ and its reflexive-transitive closure, respectively.

The following example illustrates the fact that $\mathfrak{R}$ can emulate derivations done with the inference system in Figure 6.1.

Example 6.2.1 *Consider the one context unification instance $\langle\{F(a) \doteq g(x_0, x_0),\ F(b) \doteq g(g(x_1, x_1), g(a, b))\}, \{x_0, x_1\}, F\rangle$. In this example we use $\rightarrow_r^\sigma$ to denote a derivation step using the rule r and applying the substitution σ and $\rightarrow_{\mathtt{Decompose}}^*$ to denote some derivation steps using the rule* `Decompose`. *A possible successful derivation using the rules of Figure 6.1 is the following:*

$$\{F(a) \doteq g(x_0, x_0),\ F(b) \doteq g(g(x_1, x_1), g(a, b))\}$$

$$\rightarrow_{\mathtt{Var-Elim2}}^{\{x_0 \mapsto F(\{F \mapsto g(x_0, F(\bullet))\}(a))\} \circ \{F \mapsto g(x_0, F(\bullet))\}}$$

$$\{g(F(a), F(a)) \doteq g(F(a), F(a)),\ g(F(a), F(b)) \doteq g(g(x_1, x_1), g(a, b))\}$$

$$\rightarrow_{\mathtt{Decompose}}^*$$

$$\{F(a) \doteq g(x_1, x_1),\ F(b) \doteq g(a, b)\}$$

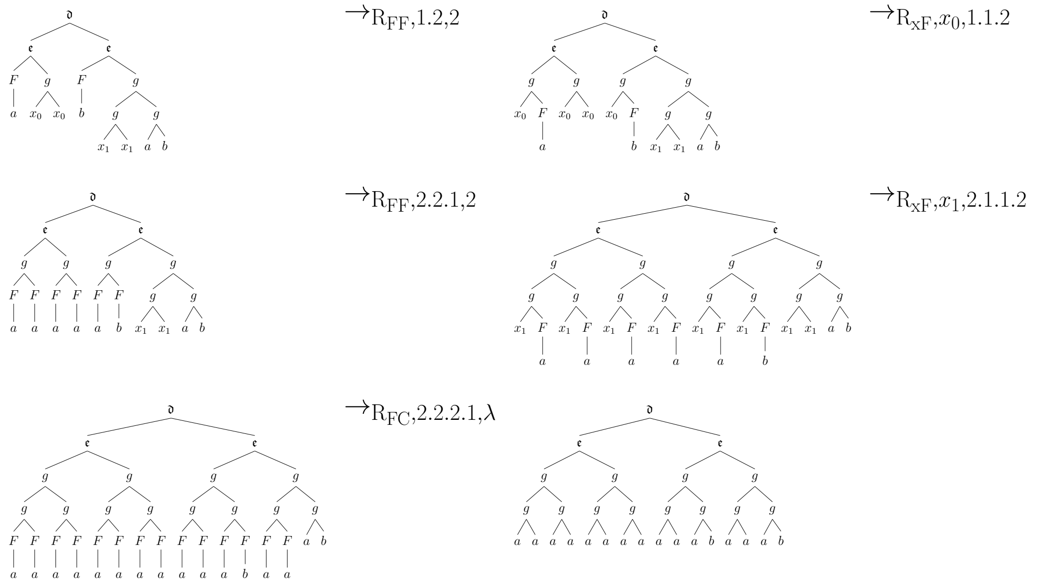
$$\rightarrow_{\mathtt{Var-Elim2}}^{\{x_1 \mapsto F(\{F \mapsto g(x_1, F(\bullet))\}(a))\} \circ \{F \mapsto g(x_1, F(\bullet))\}}$$

$$\{g(F(a), F(a)) \doteq g(F(a), F(a)),\ g(F(a), F(b)) \doteq g(a, b)\}$$

$$\rightarrow_{\mathtt{Decompose}}^* \quad \{F(a) \doteq a,\ F(b) \doteq b\} \quad \rightarrow_{\mathtt{CVar-Elim}}^{\{F \mapsto \bullet\}} \quad \{a \doteq a,\ b \doteq b\} \quad \rightarrow_{\mathtt{Decompose}}^* \quad \emptyset$$

Since we could derive $\emptyset$, the considered instance is indeed unifiable. The solution associated to the derivation is $\{F \mapsto g(g(a, a), g(a, \bullet)), x_0 \mapsto g(a, a), x_1 \mapsto a\}$.

We now show that the same solution can be derived also using $\mathfrak{R}$. The initial set of equations, expressed with the new notation, corresponds to the *term* $t = \mathsf{term}(\Delta) = \mathfrak{d}(\mathfrak{e}(F(a), g(x_0, x_0)), \mathfrak{e}(F(b), g(g(x_1, x_1), g(a, b))))$*. Note that the subterm* $t|_1$ *encodes the equation* $F(a) \doteq g(x_0, x_0)$ *and* $t|_2$ *encodes* $F(b) \doteq g(g(x_1, x_1), g(a, b))$*. Our goal is to check whether there exists a substitution* σ *such that* $\sigma(t)$ *is of the form* $\mathfrak{d}(\mathfrak{e}(t_1, t_1), \mathfrak{e}(t_2, t_2))$*. The previous derivation corresponds to the following one using* $\mathfrak{R}$*:*


$\rightarrow_{\mathrm{R}_{\mathrm{FF}},1.2,2}$
$\rightarrow_{\mathrm{R}_{\mathrm{xF}},x_0,1.1.2}$
$\rightarrow_{\mathrm{R}_{\mathrm{FF}},2.2.1,2}$
$\rightarrow_{\mathrm{R}_{\mathrm{xF}},x_1,2.1.1.2}$
$\rightarrow_{\mathrm{R}_{\mathrm{FC}},2.2.2.1,\lambda}$

Note that, there are several equivalent options for the selection of the first position used in the last rule application. We chose 2.2.2.1 *because this is the position that corresponds to the last steps of the previous derivation.*

It is trivial that $\mathfrak{R}$ is sound. Moreover, since $\mathfrak{R}$ can emulate the original inference system, it follows that it is also complete. The following lemma states that soundness and completeness of $\mathfrak{R}$ also hold when the length of derivations is linearly bounded. This property follows from completeness of the inference system of Figure 6.1, soundness of $\mathfrak{R}$, and Lemma 6.1.7, taking into account that each rule of Figure 6.1 can be emulated with at most two rules of $\mathfrak{R}$.

Lemma 6.2.2 *Let* $\langle \Delta, \mathcal{X}, F\rangle$ *be a one context unification instance.* Δ *has a solution if and only if there exists a derivation* $\mathsf{term}(\Delta) \rightarrow^*_{\mathfrak{R}} t$ *of length at most* $2|\mathcal{X}| + 1$ *such that* t *is of the form* $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_{|\Delta|}, u_{|\Delta|}))$*.*

In the case of one context unification with STGs, $\mathsf{term}(\Delta)$ is represented by a single term nonterminal. More concretely, an instance of one-context unification with STGs $\langle\{A_1 \doteq B_1, \ldots, A_n \doteq B_n\}, \bar{G}, \mathcal{X}, F\rangle$ is represented as $\langle T, G, \mathcal{X}, F\rangle$, where G is an $\mathcal{F}$-extension of $\bar{G}$ and T is a term nonterminal of

G such that $w_{G,T} = \mathfrak{d}(\mathfrak{e}(w_{\bar{G},A_1}, w_{\bar{G},B_1}), \ldots, \mathfrak{e}(w_{\bar{G},A_n}, w_{\bar{G},B_n})) = \mathsf{term}(\{w_{\bar{G},A_1} \doteq w_{\bar{G},B_1}, \ldots, w_{\bar{G},A_n} \doteq w_{\bar{G},B_n}\}) = \mathsf{term}(\Delta)$. With this new representation, the problem consists of deciding whether there exists a substitution σ such that $\sigma(w_{G,T})$ is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_n, u_n))$. Recall that NP can be defined as the set of decisional problems whose positive instances can be verified in polynomial time. Hence, to prove that one context unification with STGs is in NP we need to prove that there exists an $\mathcal{F}$-extension G' of G of polynomial size with respect to $|G|$ that represents σ. More concretely, in the rest of this chapter we prove that, for each derivation of the form $\mathsf{term}(\Delta) \rightarrow^*_{\mathfrak{R}} t$ of length at most $2|\mathcal{X}| + 1$, there exists an $\mathcal{F}$-extension G' of G such that $w_{G',T} = t$ and whose size is polynomial with respect to $|G|$. This is enough for proving that one context unification with STGs is in NP since the fact that t is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_n, u_n))$ can be checked in polynomial time with respect to $|G|$.

6.3 Commutation of Substitutions

As commented in the previous section, our approach consists of modifying the sequences of derivation steps with $\mathfrak{R}$ in such a way that allows to conclude that every unifiable instance $\langle \Delta, \mathcal{X}, F \rangle$ has a solution σ that can be represented in polynomial space using an STG. Our goal is to show that rule applications can always be commuted to obtain an equivalent derivation, i.e. a derivation describing the same substitution σ, that is of the following form:

$$\mathsf{term}(\Delta) \rightarrow^*_{\mathrm{R_{x\neg F}}} \rightarrow^*_{\mathrm{R_{FF}}} \rightarrow^*_{\mathrm{R_{xF}}} \rightarrow^{0,1}_{\mathrm{R_{FC}}} \rightarrow^*_{\mathrm{R_{x\neg F}}} \sigma(\mathsf{term}(\Delta))$$

where $\rightarrow^{0,1}_{\mathrm{R_{FC}}}$ denotes either 0 or 1 applications of the rule $\mathrm{R_{FC}}$. As a first ingredient in this argument, we define a particular notion of composition of substitutions which will be useful to reason in our setting. Next, we present some technical results to, finally, prove how to commute derivation steps until obtaining an equivalent derivation of the desired form.

6.3.1 Strong Composition

Consider two substitutions σ_1, σ_2. The goal of the following notion of composition of σ_1 and σ_2 is to capture how instantiations due to σ_1 are modified by the later application of σ_2.

Definition 6.3.1 *Let σ_1, σ_2 be substitutions. The* strong composition *of σ_1 and σ_2, denoted $\sigma_2 \diamond \sigma_1$, is defined as* $\{\alpha \mapsto \sigma_2(\sigma_1(\alpha)) \mid \alpha \in \mathsf{Dom}(\sigma_1)\}$

The usual and the strong notions of composition are not equivalent in general. Recall that, given substitutions σ_1, σ_2, the usual notion of composition can be defined as $\sigma_2 \circ \sigma_1 = \{\alpha \mapsto \sigma_2(\sigma_1(\alpha)) \mid \alpha \in \mathsf{Dom}(\sigma_1) \cup \mathsf{Dom}(\sigma_2)\}$. In order to stress the difference, consider $\theta_1 = \{y \mapsto b\}$ and $\theta_2 = \{x \mapsto a\}$ and note that $(\theta_2 \diamond \theta_1)(x) = x$, while $(\theta_2 \circ \theta_1)(x) = a$. Moreover, strong composition is not associative, i.e. $(\sigma_3 \diamond \sigma_2) \diamond \sigma_1 = \sigma_3 \diamond (\sigma_2 \diamond \sigma_1)$ does not hold in general: consider $\theta_0 = \{y \mapsto g(x)\}$ and note that $(\theta_2 \diamond \theta_1) \diamond \theta_0 = \{y \mapsto g(x)\}$, while $\theta_2 \diamond (\theta_1 \diamond \theta_0) = \{y \mapsto g(a)\}$. Another property that distinguishes both notions of composition is that, when using strong composition, $\mathsf{Dom}(\sigma_2 \diamond \sigma_1) \subseteq \mathsf{Dom}(\sigma_1)$ holds. This inclusion is strict only in anomalous cases, for example $\mathsf{Dom}(\{y \mapsto x\} \diamond \{x \mapsto y\}) = \emptyset$. In fact, the condition $\mathsf{Dom}(\sigma_2 \diamond \sigma_1) = \mathsf{Dom}(\sigma_1)$ is ensured when $\mathsf{Vars}(\sigma_2) \cap \mathsf{Dom}(\sigma_1) = \emptyset$, which is usually the case in our setting.

The following lemma is straightforward from the definition of strong composition.

Lemma 6.3.2 *Let σ_1, σ_2 be substitutions and let α be a variable in $\mathsf{Dom}(\sigma_1)$. Then, $(\sigma_2 \circ \sigma_1)(\alpha) = (\sigma_2 \diamond \sigma_1)(\alpha)$.*

The following lemma states how two substitutions can be “commuted” using the strong composition.

Lemma 6.3.3 *Let σ_1, σ_2 be substitutions such that $\mathsf{Vars}(\sigma_2) \cap \mathsf{Dom}(\sigma_1) = \emptyset$. Then, $\sigma_2 \circ \sigma_1 = (\sigma_2 \diamond \sigma_1) \circ \sigma_2$.*

Proof. Let $\mathcal{V}$ be a set of variables such that $\mathsf{Dom}(\sigma_1) \cup \mathsf{Dom}(\sigma_2) \subseteq \mathcal{V}$ holds. It suffices to prove that, for all $\alpha \in \mathcal{V}$, $(\sigma_2 \circ \sigma_1)(\alpha) = ((\sigma_2 \diamond \sigma_1) \circ \sigma_2)(\alpha)$ holds.

We distinguish cases depending on whether $\alpha \in \mathsf{Dom}(\sigma_1)$ and $\alpha \in \mathsf{Dom}(\sigma_2)$. If $\alpha \in \mathsf{Dom}(\sigma_1)$ then, by the assumption of the lemma, $\alpha \notin \mathsf{Dom}(\sigma_2)$ holds, and hence $((\sigma_2 \diamond \sigma_1) \circ \sigma_2)(\alpha) = (\sigma_2 \diamond \sigma_1)(\alpha) = (\sigma_2 \circ \sigma_1)(\alpha)$ holds by Lemma 6.3.2. If $\alpha \in \mathsf{Dom}(\sigma_2)$ then, by the assumption of the lemma, $\mathsf{Vars}(\sigma_2(\alpha)) \cap \mathsf{Dom}(\sigma_2 \diamond \sigma_1) = \emptyset$ and $\alpha \notin \mathsf{Dom}(\sigma_1)$ hold, and it follows $((\sigma_2 \diamond \sigma_1) \circ \sigma_2)(\alpha) = \sigma_2(\alpha) = (\sigma_2 \circ \sigma_1)(\alpha)$. Finally, the case where $\alpha \notin \mathsf{Dom}(\sigma_1) \cup \mathsf{Dom}(\sigma_2)$ trivially holds and the case where $\alpha \in \mathsf{Dom}(\sigma_1) \cap \mathsf{Dom}(\sigma_2)$ is not possible by the assumption. □

6.3.2 Subterm Preservation

Let t_1 and t_2 be terms such that t_2 is obtained from t_1 by applying a substitution. The following two lemmas state under which conditions a certain subterm of t_2 exists also as a subterm of t_1. These results will be crucial to argue about the commutation of derivation steps.

Lemma 6.3.4 *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2 be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_2 = \{F \mapsto c[F(\bullet)]\}(t_1)$, where c is a context in $\mathcal{C}(\mathcal{F}, \mathcal{V})$. Let p be a position in $\mathsf{Pos}(t_2)$ such that $F \notin \mathsf{Vars}(t_2|_p)$. Then, either there exists a position $\hat{p} \in \mathsf{Pos}(t_1)$ such that $t_1|_{\hat{p}} = t_2|_p$ or there exists a position $\hat{p} \in \mathsf{Pos}(c)$ such that $c|_{\hat{p}} = t_2|_p$.*

Proof. We prove the lemma by induction on $|p|$. If $p = \lambda$, note that F cannot occur in t_1 since, otherwise, $F \in \mathsf{Vars}(t_2|_p)$. Hence, in this case, $t_1 = t_2$ holds and we are done by defining $\hat{p}$ as p. For the induction step, assume that $|p| > 0$ and consider the following cases.

First, consider that t_1 is of the form $g(u_1, \ldots, u_n)$, where g is a function symbol in $\mathcal{F}$. Note that $n > 0$ holds since, otherwise, $t_2 = g$ and $p = \lambda$, contradicting the assumption. Let p be $i.p'$ more explicitly written, for $i \in \{1, \ldots, n\}$. Note that $t_2|_i = \{F \mapsto c[F(\bullet)]\}(u_i)$, $p' \in \mathsf{Pos}(t_2|_i)$, and $F \notin \mathsf{Vars}(t_2|_i|_{p'})$ hold. By induction hypothesis on $|p'|$, there exists a position $\hat{p}'$ such that either $u_i|_{\hat{p}'} = t_2|_i|_{p'}$ or $c|_{\hat{p}'} = t_2|_i|_{p'}$. The statement follows by defining $\hat{p}$ as $i.\hat{p}'$ in the former case and as $\hat{p}'$ in the latter case.

Second, consider that t_1 is of the form $F(u)$. We distinguish cases depending on whether p and $\mathtt{hp}(c)$ are parallel or not. Note that p cannot be a prefix of $\mathtt{hp}(c)$ since it would lead to a contradiction with the fact that $F \notin \mathsf{Vars}(t_2|_p)$. In the case where p and $\mathtt{hp}(c)$ are parallel, it follows that $c|_p = t_2|_p$, and we are done by defining $\hat{p}$ as p. Otherwise, let p be $\mathtt{hp}(c).1.p'$ more explicitly written. In this case, note that $t_2|_{\mathtt{hp}(c).1} = \{F \mapsto c[F(\bullet)]\}(u)$, $p' \in \mathsf{Pos}(t_2|_{\mathtt{hp}(c).1})$, and $F \notin \mathsf{Vars}(t_2|_{\mathtt{hp}(c).1}|_{p'})$ hold. Hence, by induction hypothesis on $|p'|$, there exists a position $\hat{p}'$ such that either $u|_{\hat{p}'} = t_2|_{\mathtt{hp}(c).1}|_{p'}$ or $c|_{\hat{p}'} = t_2|_{\mathtt{hp}(c).1}|_{p'}$. The statement follows by defining $\hat{p}$ as $1.\hat{p}'$ in the former case and as $\hat{p}'$ in the latter case.

Third, the case where t_1 is a first-order variable is not possible because, in such case, p must be λ, which contradicts the assumption. □

Lemma 6.3.5 *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2 be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that either $t_1 \rightarrow_{\mathrm{R_{xF}}} t_2$ or $t_1 \rightarrow_{\mathrm{R_{FF}}} t_2$. Let p be a position in $\mathsf{Pos}(t_2)$ such that $F \notin \mathsf{Vars}(t_2|_p)$. Then, there exists a position $\hat{p} \in \mathsf{Pos}(t_1)$ such that $t_1|_{\hat{p}} = t_2|_p$.*

Proof. We first consider the case where the applied rule is $\mathrm{R_{FF}}$. Let $t_1 \rightarrow_{\mathrm{R_{FF}}, p_1, p_2} t_2$ be the derivation step of the statement more explicitly written. Hence, $t_2 = \{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\}(t_1)$ and, by Lemma 6.3.4, the statement holds.

Now assume that the applied rule is $\mathrm{R_{xF}}$ with variable x and position q. Let P be the subset of positions of $\mathsf{Pos}(t_1)$ labeled by x in t_1. Note that P is a set of pairwise parallel positions. Moreover, note that p cannot be a prefix of any of the positions in P since, otherwise, $F \in \mathsf{Vars}(t_2|_p)$, contradicting the assumptions of the lemma. In the case where p is parallel with every position in P, it follows that $p \in \mathsf{Pos}(t_1)$ and $t_1|_p = t_2|_p$. Otherwise, exactly one position $p' \in P$ is a proper prefix of p. Hence, p is of the form $p'.q'$ and it follows that $t_1|_{q.q'} = t_2|_p$. □

The following lemma states how the instantiation of a context variable and the computation of a subterm can be commuted.

Lemma 6.3.6 *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t be a term in $\mathcal{T}(\mathcal{F},\mathcal{V})$, let p be a position in $\mathsf{Pos}(t)$, and let $\sigma = \{F \mapsto c\}$ be a substitution, where c is a context in $\mathcal{C}(\mathcal{F},\mathcal{V})$. Then, there exists a position $\hat{p} \in \mathsf{Pos}(\sigma(t))$ such that $\sigma(t|_p) = \sigma(t)|_{\hat{p}}$.*

Proof. We prove the lemma by induction on $|p|$. The base case, i.e. when $p = \lambda$, trivially holds by defining $\hat{p}$ as p. For the induction step, assume that $|p| > 0$. We distinguish cases depending on the form of t.

First, assume that t is of the form $g(u_1, \ldots, u_n)$, where g is a function symbol in $\mathcal{F}$. Note that $n > 0$ necessarily holds since, otherwise, $p = \lambda$, contradicting the assumption. Let p be $i.p'$ more explicitly written, for $i \in \{1, \ldots, n\}$. By induction hypothesis, there exists a position $\hat{p}' \in \mathsf{Pos}(\sigma(u_i))$ such that $\sigma(u_i|_{p'}) = \sigma(u_i)|_{\hat{p}'}$ holds. Hence, the statement holds by defining $\hat{p}$ as $i.\hat{p}'$.

Second, assume that t is of the form $F(u)$. Let p be $1.p'$ more explicitly written. By induction hypothesis, there exists a position $\hat{p}' \in \mathsf{Pos}(\sigma(u))$ such that $\sigma(u|_{p'}) = \sigma(u)|_{\hat{p}'}$ holds. Hence, the statement holds by defining $\hat{p}$ as $\mathtt{hp}(c).\hat{p}'$.

Third, the case where t is a first-order variable is not possible because, in such case, p must be λ, which contradicts the assumption. □

6.3.3 Reordering Derivations

In this section we show how derivations with $\mathfrak{R}$ can be modified in order to guarantee that rules are applied in a specific order. As basic ingredients, the following three technical lemmas show how pairs of derivation steps can be swapped.

Lemma 6.3.7 *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2, t be terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$ such that $t_1 \rightarrow_{\mathrm{R_{xF}},x,p} t \rightarrow_{\mathrm{R_{FF}},p_1,p_2} t_2$. Then, there exists a position $\hat{p}_1 \in \mathsf{Pos}(t_1)$, a term*

$t' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and a position $\hat{p} \in \mathsf{Pos}(t')$ such that $t_1 \to_{\mathrm{R_{FF}},\hat{p}_1,p_2} t' \to_{\mathrm{R_{xF}},x,\hat{p}} t_2$ holds.

Proof. By the conditions on the application of $\mathrm{R_{xF}}$, $x \in \mathsf{Vars}(t_1) \setminus \mathsf{Vars}(t_1|_p)$ and $F \in \mathsf{Vars}(t_1|_p)$ hold. In addition, since $\mathrm{R_{xF}}$ instantiates x, then $x \notin \mathsf{Vars}(t)$ holds. Moreover, by the conditions on the application of $\mathrm{R_{FF}}$, we know that $F \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_1})$. Note that

$$t_2 =\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(\{x \mapsto t_1|_p\}(t_1)) \tag{6.1}$$

$$=(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\} \diamond \{x \mapsto t_1|_p\})(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1)) \tag{6.2}$$

$$=\{x \mapsto \{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1|_p)\}(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1)) \tag{6.3}$$

$$=\{x \mapsto \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1|_p)\}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)) \tag{6.4}$$

$$=\{x \mapsto \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}}\}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)) \tag{6.5}$$

where (1) follows from definition of $\mathrm{R_{xF}}$ and $\mathrm{R_{FF}}$, (2) follows from Lemma 6.3.3, which can be applied because $\mathsf{Vars}(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}) \cap \mathsf{Dom}(\{x \mapsto t_1|_p\}) = \emptyset$ holds since $x \notin \mathsf{Vars}(t)$ and $x \neq F$, (3) follows from Definition 6.3.1, in (4) the implicit definition of $\hat{p}_1$ holding $t_1|_{\hat{p}_1} = t|_{p_1}$ follows from Lemma 6.3.5, which can be applied since $t_1 \to_{\mathrm{R_{xF}},x,p} t$ and $F \notin \mathsf{Vars}(t|_{p_1})$, and in (5) the implicit definition of $\hat{p}$ holding $\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}} = \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1|_p)$ follows from Lemma 6.3.6.

Finally, let the t' of the lemma be defined as $\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)$ and note that $t_2 = \{x \mapsto t'|_{\hat{p}}\}(t')$. It remains to prove that t' can be derived from t_1, and that t_2 can be derived from t', as stated in the lemma. First, note that $t_1 \to_{\mathrm{R_{FF}},\hat{p}_1,p_2} t'$ holds because F does not occur in $t_1|_{\hat{p}_1}$, since $t_1|_{\hat{p}_1} = t|_{p_1}$, and, moreover, $F \in \mathsf{Vars}(t_1)$ holds because $F \in \mathsf{Vars}(t)$ and $\mathsf{Vars}(t) \subsetneq \mathsf{Vars}(t_1)$. Now, note that $t' \to_{\mathrm{R_{xF}},x,\hat{p}} t_2$ holds because:

- $x \in \mathsf{Vars}(t')$, which follows from the facts that $x \in \mathsf{Vars}(t_1)$ and $\mathsf{Vars}(t') = \mathsf{Vars}(t_1)$,
- $x \notin \mathsf{Vars}(t'|_{\hat{p}})$, since $x \notin \mathsf{Vars}(t_1|_p)$, $x \notin \mathsf{Vars}(t)$, and $t'|_{\hat{p}} = \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}} = \{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1|_p)$, and
- $F \in \mathsf{Vars}(t'|_{\hat{p}})$, since $F \in \mathsf{Vars}(t_1|_p)$ holds and thus $F \in \mathsf{Vars}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1|_p)) = \mathsf{Vars}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}}) = \mathsf{Vars}(t'|_{\hat{p}})$ also holds.

Hence, $t_1 \to_{\mathrm{R_{FF}},\hat{p}_1,p_2} t' \to_{\mathrm{R_{xF}},x,\hat{p}} t_2$ holds, which concludes the proof. □

Lemma 6.3.8 *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2, t be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_1 \to_{\mathrm{R_{FF}},p_1,p_2} t \to_{\mathrm{R_{x\neg F}},x,p} t_2$. Then, there exists a position $\hat{p} \in \mathsf{Pos}(t_1)$ and a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_1 \to_{\mathrm{R_{x\neg F}},x,\hat{p}} t' \to_{\mathrm{R_{FF}},p_1,p_2} t_2$ holds.*

Proof. By the conditions on the application of rule $\mathrm{R_{FF}}$, $F \in \mathsf{Vars}(t_1) \setminus \mathsf{Vars}(t_1|_{p_1})$ holds. Moreover, by the conditions on the application of rule $\mathrm{R_{x\neg F}}$, we know that $x \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_p)$ and $F \notin \mathsf{Vars}(t|_p)$. Note that

$$\begin{aligned} t_2 =& \{x \mapsto t|_p\}(\{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\}(t_1)) && (6.1)\\ =& (\{x \mapsto t|_p\} \diamond \{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\})(\{x \mapsto t|_p\}(t_1)) && (6.2)\\ =& \{F \mapsto \{x \mapsto t|_p\}(t_1|_{p_1}[F(\bullet)]_{p_2})\}(\{x \mapsto t|_p\}(t_1)) && (6.3)\\ =& \{F \mapsto \{x \mapsto t_1|_{\hat{p}}\}(t_1|_{p_1}[F(\bullet)]_{p_2})\}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)) && (6.4)\\ =& \{F \mapsto \{x \mapsto t_1|_{\hat{p}}\}(t_1)|_{p_1}[F(\bullet)]_{p_2}\}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)) && (6.5) \end{aligned}$$

where (1) follows from definition of $\mathrm{R_{FF}}$ and $\mathrm{R_{x\neg F}}$, (2) follows from Lemma 6.3.3, which can be applied because $\mathsf{Vars}(\{x \mapsto t|_p\}) \cap \mathsf{Dom}(\{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\}) = \emptyset$ holds since $F \notin \mathsf{Vars}(t|_p)$ and $F \neq x$, (3) follows from Definition 6.3.1, in (4) the implicit definition of $\hat{p}$ holding $t_1|_{\hat{p}} = t|_p$ follows from Lemma 6.3.5, which can be applied since $t_1 \rightarrow_{\mathrm{R_{FF}},p_1,p_2} t$ and $F \notin \mathsf{Vars}(t|_p)$, and (5) holds because replacements of first-order variables and computation of subterms can be commuted in this way.

Finally, let the t' of the lemma be defined as $\{x \mapsto t_1|_{\hat{p}}\}(t_1)$ and note that $t_2 = \{F \mapsto t'|_{p_1}[F(\bullet)]_{p_2}\}(t')$. It remains to prove that t' can be derived from t_1, and that t_2 can be derived from t', as stated in the lemma. First, note that $t_1 \rightarrow_{\mathrm{R_{x\neg F}},x,\hat{p}} t'$ holds because neither F nor x occur in $t_1|_{\hat{p}}$ since $t_1|_{\hat{p}} = t|_p$ and, moreover, $x \in \mathsf{Vars}(t_1)$ holds because $x \in \mathsf{Vars}(t)$ and $\mathsf{Vars}(t) = \mathsf{Vars}(t_1)$. Now, note that $t' \rightarrow_{\mathrm{R_{FF}},p_1,p_2} t_2$ holds because:

- $F \in \mathsf{Vars}(t')$, since $F \in \mathsf{Vars}(t_1) \setminus \{x\} = \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)) = \mathsf{Vars}(t')$, and

- $F \notin \mathsf{Vars}(t'|_{p_1})$ since $F \notin \mathsf{Vars}(t_1|_{p_1}) \cup \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\})$ and thus $F \notin \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\}(t_1|_{p_1})) = \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)|_{p_1}) = \mathsf{Vars}(t'|_{p_1})$.

Hence, $t_1 \rightarrow_{\mathrm{R_{x\neg F}},x,\hat{p}} t' \rightarrow_{\mathrm{R_{FF}},p_1,p_2} t_2$ holds, which concludes the proof. □

Lemma 6.3.9 *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2, t be terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$ such that $t_1 \rightarrow_{\mathrm{R_{xF}},x_1,p_1} t \rightarrow_{\mathrm{R_{x\neg F}},x_2,p_2} t_2$. Then, there exists a position $\hat{p}_2 \in \mathsf{Pos}(t_1)$ and a term $t' \in \mathcal{T}(\mathcal{F},\mathcal{V})$ such that $t_1 \rightarrow_{\mathrm{R_{x\neg F}},x_2,\hat{p}_2} t' \rightarrow_{\mathrm{R_{xF}},x_1,p_1} t_2$ holds.*

Proof. By the conditions on the application of $\mathrm{R_{xF}}$, $x_1 \in \mathsf{Vars}(t_1) \setminus \mathsf{Vars}(t_1|_{p_1})$ and $F \in \mathsf{Vars}(t_1|_{p_1})$ hold. Moreover, by the conditions on the application of $\mathrm{R_{x\neg F}}$, we know that $x_2 \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_2})$ and $F \notin \mathsf{Vars}(t|_{p_2})$.

Note that

$$\begin{aligned} t_2 =& \{x_2 \mapsto t|_{p_2}\}(\{x_1 \mapsto t_1|_{p_1}\}(t_1)) \tag{6.1}\\ =&(\{x_2 \mapsto t|_{p_2}\} \diamond \{x_1 \mapsto t_1|_{p_1}\})(\{x_2 \mapsto t|_{p_2}\}(t_1)) \tag{6.2}\\ =&\{x_1 \mapsto \{x_2 \mapsto t|_{p_2}\}(t_1|_{p_1})\}(\{x_2 \mapsto t|_{p_2}\}(t_1)) \tag{6.3}\\ =&\{x_1 \mapsto \{x_2 \mapsto t_1|_{\hat{p}_2}\}(t_1|_{p_1})\}(\{x_2 \mapsto t_1|_{\hat{p}_2}\}(t_1)) \tag{6.4}\\ =&\{x_1 \mapsto \{x_2 \mapsto t_1|_{\hat{p}_2}\}(t_1)|_{p_1}\}(\{x_2 \mapsto t_1|_{\hat{p}_2}\}(t_1)) \tag{6.5} \end{aligned}$$

where (1) follows from definition of R_{xF} and $\mathrm{R}_{\mathrm{x\neg F}}$, (2) follows from Lemma 6.3.3, which can be applied because $\mathsf{Vars}(\{x_2 \mapsto t|_{p_2}\}) \cap \mathsf{Dom}(\{x_1 \mapsto t_1|_{p_1}\}) = \emptyset$ holds since $x_1 \notin \mathsf{Vars}(t)$ and $x_1 \neq x_2$, (3) follows from Definition 6.3.1, in (4) the implicit definition of $\hat{p}_2$ holding $t_1|_{\hat{p}_2} = t|_{p_2}$ follows from Lemma 6.3.5, which can be applied since $t_1 \rightarrow_{\mathrm{R}_{\mathrm{xF}},x_1,p_1} t$ and $F \notin \mathsf{Vars}(t|_{p_2})$, and (5) holds because replacements of first-order variables and computation of subterms can be commuted in this way.

Finally, let the t' of the lemma be defined as $\{x_2 \mapsto t_1|_{\hat{p}_2}\}(t_1)$ and note that $t_2 = \{x_1 \mapsto t'|_{p_1}\}(t')$. It remains to prove that t' can be derived from t_1, and that t_2 can be derived from t', as stated in the lemma. First, note that $t_1 \rightarrow_{\mathrm{R}_{\mathrm{x\neg F}},x_2,\hat{p}_2} t'$ holds because neither F nor x_2 occur in $t_1|_{\hat{p}_2}$ since $t_1|_{\hat{p}_2} = t|_{p_2}$ and, moreover, $x_2 \in \mathsf{Vars}(t_1)$ holds because $x_2 \in \mathsf{Vars}(t) = \mathsf{Vars}(t_1) \setminus \{x_1\}$. Now, note that $t' \rightarrow_{\mathrm{R}_{\mathrm{xF}},x_1,p_1} t_2$ holds because:

- $F \in \mathsf{Vars}(t'|_{p_1})$ since $F \in \mathsf{Vars}(t_1|_{p_1})$ and $F \neq x_2$, and
- $x_1 \in \mathsf{Vars}(t') \setminus \mathsf{Vars}(t'|_{p_1})$ since $x_1 \in \mathsf{Vars}(t_1) \setminus \mathsf{Vars}(t_1|_{p_1})$, and $x_1 \notin \mathsf{Vars}(\{x_2 \mapsto t_1|_{\hat{p}_2}\})$.

Hence, $t_1 \rightarrow_{\mathrm{R}_{\mathrm{x\neg F}},x_2,\hat{p}_2} t' \rightarrow_{\mathrm{R}_{\mathrm{xF}},x_1,p_1} t_2$ holds, which concludes the proof. □

The following result follows from the three previous lemmas, summarizing the goal of this section.

Lemma 6.3.10 *Let $\langle \Delta, \mathcal{X}, F \rangle$ be a one-context unification instance. Let t be a term such that $\mathsf{term}(\Delta) \rightarrow^*_{\mathfrak{R}} t$ in n derivation steps. Then, there exists a derivation of length n of the form $\mathsf{term}(\Delta) \rightarrow^*_{\mathrm{R}_{\mathrm{x\neg F}}} \rightarrow^*_{\mathrm{R}_{\mathrm{FF}}} \rightarrow^*_{\mathrm{R}_{\mathrm{xF}}} \rightarrow^{0,1}_{\mathrm{R}_{\mathrm{FC}}} \rightarrow^*_{\mathrm{R}_{\mathrm{x\neg F}}} t$.*

6.4 Complexity Analysis

We now have all the ingredients needed to prove that the one-context unification where the input terms are compressed using STGs is in NP. To prove this fact, we show, for each unifiable instance, that there exists a witness of polynomial size verifiable in polynomial time. In our setting, this witness is an STG generating the unified term and the verification consists of checking whether such a term is of a certain form.

Theorem 6.4.1 *One context unification with STGs is in NP.*

Proof. Let $\langle \Delta, \mathcal{X}, F \rangle$ be a one-context unification instance represented by an STG G and a nonterminal T of G such that $w_{G,T} = \mathsf{term}(\Delta)$. By Lemma 6.2.2, Δ has a solution if and only if there exists a derivation $\mathsf{term}(\Delta) \rightarrow^*_{\mathfrak{R}} t$ of length at most $2|\mathcal{X}| + 1$ such that t is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_{|\Delta|}, u_{|\Delta|}))$. Moreover, by Lemma 6.3.10, we assume without loss of generality that this derivation is of the form $\mathsf{term}(\Delta) \rightarrow^*_{\mathrm{R}_{\mathrm{x}\neg\mathrm{F}}} t_1 \rightarrow^*_{\mathrm{R}_{\mathrm{FF}}} t_2 \rightarrow^*_{\mathrm{R}_{\mathrm{xF}}} t_3 \rightarrow^{0,1}_{\mathrm{R}_{\mathrm{FC}}} t_4 \rightarrow^*_{\mathrm{R}_{\mathrm{x}\neg\mathrm{F}}} t$, for some terms t_1, t_2, t_3, t_4.

We first prove that there exists an $\mathcal{F}$-extension G' of G such that $w_{G',T} = t$ and whose size is polynomially bounded by $|G|$. By Lemma 6.1.2, there exists an $\mathcal{F}$-extension G_1 of G such that $w_{G_1,T} = t_1$ and whose size is polynomially bounded by $|G|$. Now we claim that there exists an $\mathcal{F}$-extension G_2 of G_1 such that $w_{G_2,T} = t_2$ and whose size is polynomially bounded by $|G_1|$. By the conditions on the application of R_{FF} and by Lemma 6.3.5, the subcontexts computed in each step of the subderivation $t_1 \rightarrow^*_{\mathrm{R}_{\mathrm{FF}}} t_2$ can be obtained from t_1. Hence, the sequence of applications of rules R_{FF} can be seen as a single application of a substitution of the form $\{F \mapsto c_1 \ldots c_n F(\bullet)\}$, where each c_i is a subcontext of t_1. By Lemma 6.1.3, there exists an $\mathcal{F}$-extension G_1' of G_1 such that $w_{G_1',T} = t_1$ with a context nonterminal C such that $w_{G_1',C} = c_1 \ldots c_n$ and whose size is polynomially bounded by $|G_1|$ and $|\mathcal{X}|$. The STG G_2 mentioned above is defined as the $\mathcal{F}$-extension of G_1' obtained by transforming the context variable into a context nonterminal generating $c_1 \ldots c_n F(\bullet)$, where F is a freshly introduced terminal symbol standing for the context variable. Again by Lemma 6.1.2, there exists an $\mathcal{F}$-extension G_3 of G_2 such that $w_{G_3,T} = t_3$ and whose size is polynomially bounded by $|G_2|$. At this point, note that the exponent e involved in the application of R_{FC} can be at most exponential with respect to $|G_3|$ since it is linear with $|t_3|$. Hence, by Lemma 6.1.5, there exists an $\mathcal{F}$-extension G_4 of G_3 such that $w_{G_4,T} = t_4$ and whose size is polynomially bounded by $|G_3|$. Hence, by definition of G_1, G_2, G_3, G_4, it follows that $|G_4|$ is polynomially bounded by $|G|$. Finally, the existence of the grammar G' mentioned above follows from Lemma 6.1.2 and the fact that $|G_4|$ is polynomially bounded by $|G|$. Note that, as commented in Section 6.1.1, the size of the representation of G' is bounded by $|G'| \cdot (2 + m)$, where m is the maximum arity of the terminals of G'.

To conclude the proof, note that the property whether the term generated by a certain nonterminal, in our case by the nonterminal T of G', is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_{|\Delta|}, u_{|\Delta|}))$ can be checked in polynomial time with respect to the size of the given grammar. To see this, first note that checking whether the symbol labeling $w_{G',T}$ at position p, with $|p| = 0$ or $|p| = 1$, is

$\mathfrak{d}$ or $\mathfrak{e}$, respectively, can be computed in linear time. Finally, nonterminals generating $w_{G',T}|_{i.1}$ and $w_{G',T}|_{i.2}$, for $i \in \{1, \ldots, |\Delta|\}$, can be computed efficiently, and thus, by Theorem 3.2.16, checking whether $w_{G',T}|_{i.1} = w_{G',T}|_{i.2}$ can be solved in polynomial time. $\square$

Chapter 7

Context Matching

In this chapter, we prove two results for context matching, the particular case of context unification in which one of the sides of the input equations does not have variables of any kind.

7.1 k-Context Matching with DAGs

The context matching problem is NP-complete [SSS98]. In this section we reconsider this problem by introducing the additional restriction stating that the maximum number k of different context variables of a given instance is fixed for the problem. We refer to this problem as *k-context matching*, which is in fact a family of problems indexed by k. Our goal is to prove that a complete representation of all solutions is computable in polynomial time when the input terms are represented with DAGs. This variant is called *k-context matching with DAGs* (k-CMD problem).

Our algorithm is presented as nondeterministic, but where the guessing is restricted to a polynomial number of possibilities. In Section 7.1.1, we solve the problem for the simpler case of uncompressed terms. This case is easy, but serves for a better understanding of some ideas presented later, and shows how the use of nondeterminism simplifies explanations. In Section 7.1.2, we present a particular case in which the context solution for a context variable can be inferred. This idea is used several times in the algorithm. In Section 7.1.3, we give the intuition behind the algorithm in order to help getting over the technical difficulties. In Section 7.1.4 we specify the data representation used in the algorithm, based on STGs. We explain the advantages of using STGs for representing DAGs, such as clarity, but also simplicity when analyzing complexity of the required operations on DAGs. In Section 7.1.5 we present the algorithm by means of a set of rules that modify the initial set of equations and prove that they are sound and complete. Moreover, we show that they give a complete representation of all the solutions of the initial set of equations. In Section 7.1.6 we analyze the computational complexity of our algorithm, to conclude that it runs in nondeterministic polynomial time.

7.1.1 k-Context Matching for Uncompressed Terms

The k-context matching problem can be easily solved in polynomial time. Let $\{s \doteq t\}$ be an instance of the problem, where t is a ground term and s contains at most k different context variables. Any solution of $\{s \doteq t\}$ instantiates each context variable by a subcontext of t. The number of different subcontexts of t is bounded by $|t|^2$, since any context occurring in t can be defined by two positions of t: the root position and the hole position of the context. Hence, it suffices to do at most k guessings of contexts for the context variables among $|t|^2$ possibilities. After applying this partial substitution, we have to check if the resulting first-order matching problem has a solution. Since k is assumed to be fixed and the application of the partial substitutions does not lead to a exponential blowup of the size if the equations, the overall execution time is polynomial.

When the input is compressed with DAGs, the problem becomes more difficult. In particular, the number of different contexts of the right-hand side can be exponential with respect of the size of the representation. The following example illustrates this fact.

Example 7.1.1 *Consider a natural number $n > 1$ and the term t_n defined by the set of equations $\{t_0 = f(a,b), t'_0 = f(b,a), t_1 = f(t_0,t'_0), t'_1 = f(t'_0,t_0), t_2 = f(t_1,t'_1), t'_2 = f(t'_1,t_1), \ldots, t_n = f(t_{n-1},t'_{n-1})\}$. Note that the number of different subcontexts of t_n is exponential with respect to n. Hence, the approach used to solve k-CMD in polynomial time in the uncompressed case cannot be applied when the input is compressed. For instance, guessing the solution of F in the matching equation $F(f(f(b,a),f(a,b))) \doteq t_n$ would take exponential time when t_n is represented with a DAG.*

7.1.2 Inferring the Joint Context

One of the key points for obtaining a polynomial time algorithm is the fact that in some cases, the context solution for a context variable can be inferred. In this section we present this particular case regardless of which is the representation for terms: consider the simple case where we have two matching equations of the form $F(s) \doteq u$ and $F(t) \doteq v$, and suppose that u and v are different. Suppose also that we know the existence of a solution

σ for these equations, but the only known information about the solution σ is $|\mathtt{hp}(\sigma(F))|$, i.e. the length of the hole position of $\sigma(F)$. It can be proved that this information suffices to obtain $\sigma(F)$. With this aim we define below $\mathtt{JointCon}(u, v, l)$, the *joint context* of u and v, for any terms u and v, and natural number l, which intuitively corresponds to the known information $|\mathtt{hp}(\sigma(F))|$.

Definition 7.1.2 *Let $u \neq v$ be terms and let $l \in \mathbb{N}$. We define $\mathtt{JointCon}(u, v, 0)$ to be the empty context $\bullet$. We also define $\mathtt{JointCon}(f(u_1, \ldots, u_m), g(v_1, \ldots, v_m), l + 1) = f(u_1, \ldots, u_{i-1}, \mathtt{JointCon}(u_i, v_i, l), u_{i+1}, \ldots, u_m)$ in the case where $f = g$ and there exists $i \in \{1, \ldots, m\}$ such that $u_j = v_j$ for all $j \neq i$. Otherwise, $\mathtt{JointCon}(f(u_1, \ldots, u_m), f(v_1, \ldots, v_m), l + 1)$ is undefined.*

Note that in the second case of the previous definition, if $f = g$ and such an i exists, then it is unique. This is because $f(u_1, \ldots, u_m)$ and $g(v_1, \ldots, v_m)$ are different, and hence, $u_j = v_j$ for all $j \neq i$ implies that $u_i \neq v_i$.

Example 7.1.3 *Let u, v, w be $f(a, g(h(a, a), c), b)$, $f(a, g(h(b, b), c), b)$ and $g(f(a, b, c), b)$, respectively. Then, $\mathtt{JointCon}(u, v, 0) = \mathtt{JointCon}(u, w, 0) = \bullet$, $\mathtt{JointCon}(u, v, 1) = f(a, \bullet, b)$, $\mathtt{JointCon}(u, w, 1)$ is undefined, $\mathtt{JointCon}(u, v, 2) = f(a, g(\bullet, c), b)$, and $\mathtt{JointCon}(u, v, 3)$ is undefined.*

The following lemma relates the $\mathtt{JointCon}$ operation with our problem.

Lemma 7.1.4 *Let s, t, u, v be terms with $u \neq v$. Let σ be a solution of $\{F(s) \doteq u, F(t) \doteq v\}$. Then $\sigma(F) = \mathtt{JointCon}(u, v, |\mathtt{hp}(\sigma(F))|)$.*

Proof. We prove the claim by induction on $|\mathtt{hp}(\sigma(F))|$. If $|\mathtt{hp}(\sigma(F))|$ is 0, then $\sigma(F)$ is $\bullet$, which coincides with $\mathtt{JointCon}(u, v, |\mathtt{hp}(\sigma(F))|)$. Now, suppose that $|\mathtt{hp}(\sigma(F))|$ is $l + 1$ for some natural number l. This implies that $\sigma(F)$ is of the form $f(w_1, \ldots, w_{i-1}, c, w_{i+1}, \ldots, w_m)$ for some function symbol f, context c, and $i \in [1, m]$. Since σ is a solution of $\{F(s) \doteq u, F(t) \doteq v\}$, then u and v are of the form $f(u_1, \ldots, u_m)$ and $f(v_1, \ldots, v_m)$, respectively. For the same reason, $w_j = u_j = v_j$ for all $j \neq i$, and moreover, $\sigma(c[s]) = u_i$ and $\sigma(c[t]) = v_i$. Since $u \neq v$ holds, we also have $u_i \neq v_i$. Consider a new context variable F' and the extension of σ as $\sigma(F') = c$. Then, σ is also a solution of $\{F'(s) \doteq u_i, F'(t) \doteq v_i\}$. Note that $|\mathtt{hp}(\sigma(F'))|$ is l, which is smaller than $|\mathtt{hp}(\sigma(F))|$. By induction hypothesis, $\sigma(F') = \mathtt{JointCon}(u_i, v_i, |\mathtt{hp}(\sigma(F'))|)$. Hence, we conclude $\sigma(F) = f(w_1, \ldots, w_{i-1}, c, w_{i+1}, \ldots, w_m) = f(w_1, \ldots, w_{i-1}, \sigma(F'), w_{i+1}, \ldots, w_m) = f(w_1, \ldots, w_{i-1}, \mathtt{JointCon}(u_i, v_i, |\mathtt{hp}(\sigma(F'))|), w_{i+1}, \ldots, w_m) = \mathtt{JointCon}(u, v, |\mathtt{hp}(\sigma(F))|)$ □

7.1.3 The Intuition Behind the Algorithm

Our algorithm is presented as a set of nondeterministic rules that deal with a set of equations on terms represented by DAGs. When we reason about its complexity, we argue about the determinized version that computes all guessing possibilities.

As already mentioned, we cannot directly guess a context of the right-hand side for every context variable, since there may be exponentially many contexts. In spite of this fact, we show that making an adequate use of the cases where the joint context can be inferred, the number of possibilities for each guessing can be drastically reduced. This fact allows us to use this approach also for the case when terms are represented with DAGs.

After some standard applications of simplification and first-order variable elimination, we can assume that every matching equation in the set Δ is of the form $F(s) \doteq t$, for some context variable F. Now, our goal is to remove one context variable by performing a guess, where the overall number of possibilities remains polynomial.

Suppose first that Δ contains two equations of the form $F(s_1) \doteq t_1$ and $F(s_2) = t_2$ with $t_1 \neq t_2$. Our goal is to infer the solution context for F as in the last subsection. However, to apply Lemma 7.1.4, we still need the length of the hole position of $\sigma(F)$, for a possible solution σ. But this length can be guessed from $\{0, \ldots, \mathtt{min}(\mathtt{height}(t_1), \mathtt{height}(t_2))\}$ which is linear in the input size, since we are using DAGs for term representation.

Another situation is when Δ is of the form $\{F(s_1) \doteq t, \ldots, F(s_n) \doteq t\} \cup \Delta'$ for some term t and F does not occur elsewhere. In this case, a solution σ for Δ necessarily satisfies that $\sigma(F)$ is a certain context c such that t is of the form $c[t']$ for some subterm t' of t. There may be exponentially many occurrences of t' in t as a term, but there is only a linear number of different subterms of t. Hence, we only have to look for t', which can be guessed among only a linear number of possibilities of subterms of t. Then the problem can be reduced to $\{s_1 \doteq t', \ldots, s_n \doteq t'\} \cup \Delta'$. Note that the variable F does not appear any more.

Now, suppose that some context variable has an occurrence at some non-root position in some term occurring in Δ. A particular case occurs when there is an equation $F(s) \doteq t$ in Δ such that a subterm of s is of the form $F(s')$, i.e. the context variable F appears twice, at the root, and at some other position. Any possible solution σ satisfies that either $\sigma(F)$ is the empty context $\bullet$, which can be decided with a single guessing, or else $\sigma(F(s'))$ equals a proper subterm t' of t. In the latter case, the pair of equations $\{F(s) \doteq t, F(s') \doteq t'\}$ with $t \neq t'$ allows us to proceed again by inferring the context, as in the first case.

If none of the previous cases hold, then there exist equations $F_1(s_1) \doteq t_1, F_2(s_2) \doteq t_2, \ldots, F_n(s_n) \doteq t_n$ in Δ, where F_1 occurs in s_2, F_2 occurs in s_3, and so on, and F_n occurs in s_1. In this sequence there is a maximal height term, say t_1. Thus, $\texttt{height}(t_1) \geq \texttt{height}(t_2)$. Note that s_2 contains a subterm of the form $F_1(s'_2)$. Then, similarly as above, either $\sigma(F_2) = \bullet$ or we can use the equations $F_1(s_1) \doteq t_1, F_1(s'_2) \doteq t'_2$, with t'_2 chosen from the proper subDAGs of t_2, to infer $\sigma(F_2)$.

With this approach each one of the k context variables is instantiated by a guessing among a polynomial number of possibilities. Hence, at this point we can bring forward that the final cost of the algorithm will be exponential in k, which is a constant of the problem. However, we also need to choose a representation for DAGs that allows to efficiently instantiate both first-order and context variables. This is done in the next section.

7.1.4 DAG Representation of the k-CMD Algorithm

Before presenting our algorithm for the k-CMD problem in detail, it is necessary to define how we represent DAGs and how our algorithm deals with such a representation. As stated in Definition 7.1.5 of Chapter 3, DAGs can be represented as a particular case of an STG, i.e an STG which does not have context nonterminals. Let us restate such definition here.

Definition 7.1.5 *A* DAG *is an STG where the set of context nonterminals $\mathcal{CN}$ is empty, and moreover, there are only rules of the form $A \rightarrow f(A_1, \ldots, A_m)$.*

For reasons that will be made clear soon we denote DAGs using this representation.

Definition 7.1.6 *An instance of the k-context-matching problem with DAGs is a pair $\langle \Delta, G \rangle$ where the STG G is a DAG and Δ is a set of equations $\{A_{s_1} \doteq A_{t_1}, \ldots, A_{s_n} \doteq A_{t_n}\}$, where each A_{s_i} and each A_{t_i} is a term nonterminal of G, and each $w_{A_{t_i}}$ is ground. The question is to compute a substitution σ (the solution) for the variables such that $\sigma(w_{A_{s_i}}) = w_{A_{t_i}}$ for every equation i in Δ.*

During the execution of the k-CMD algorithm, the equations are processed, and G is transformed in order to represent the partial solution at each step. More concretely, first-order variables are converted into term nonterminals, and context variables are converted into context nonterminals, whose generated terms and contexts represent substitutions of a partial solution. By *variables* we mean the variables of the problem and by *function* *symbols* we mean the terminals of the grammar which are not variables although, initially, all of them are terminals of the grammar. The initial G has no context nonterminals, and it may incorporate them in order to represent that the context variables have been instantiated. Our algorithm implements the instantiation of variables as a transformation of the STG: the variables are transformed into nonterminals by adding rules for them, without changing the original rules. This ensures that right-hand sides of equations always represent subterms of an original $w_{G,A_{t_i}}$. Hence, although context variables are created during the execution, right-hand sides are always represented by a subset of the initial G, which is still a DAG according to Definition 7.1.5.

Using STGs for describing DAGs, instead of just talking about DAGs understood as directed acyclic graphs, has several advantages. First, we do not have to think about nodes and arrows. STGs are more syntactic and it is easier and clearer to add(remove) rules to(from) an STG than to talk about redirecting arrows, new inserted nodes, etc. Second, the formalism of STGs is an improvement in clarity and simplicity with respect to the usual concept of solved form for representing partial and final solutions. At the end of the execution, the obtained substitution for a first-order variable x will be w_x, i.e. this variable will be a term nonterminal, and its generated term will be the substitution computed for it. Analogously, a context variable F will be transformed into a context nonterminal, and the substitution computed for it will be w_F. Third, analyzing the size increase of the representation due to variable instantiation is much simpler: adding a rule $F \to \alpha$ for a context variable F and transforming F into a context nonterminal is easy to analyze, whereas replacing each node in the DAG labeled with F by new nodes representing its substitution is a more complicated operation. On the other hand, this representation has the disadvantage that the set of equations is not enough by itself, but needs the STG. For this reason, our algorithm needs to use the rules of G and perform some replacements of nonterminals by their corresponding definition.

There is a case where our algorithm has to guess a partial solution from an exponential number of possibilities. This happens when we have equations $F(s_1) \doteq t, \ldots, F(s_n) \doteq t$, and the context variable F does not appear elsewhere. In this case, the only important information to be kept is which subterm t' of t has to be selected in order to generate the equations $s_1 \doteq t', \ldots, s_n \doteq t'$. The solution for F might be any context c such that $c[t'] = t$, that is, the hole position of the solution of F is any path from the root of t to an occurrence of t'. This situation is illustrated in the following example.

Example 7.1.7 *Let G be an STG with set of rules* $\{A_0 \to f(A_1, A_1),\ A_1 \to$

$f(A_2, A_2)$, $\ldots$, $A_{n-1} \to f(A_n, A_n)$, $A_n \to a$, $A_{s_1} \to F(A_n)$, $A_{s_2} \to F(A_x)$, $A_x \to x\}$, where the terminals F and x stand for a context variable and a first-order variable, respectively. As seen in Chapter 3, A_0 represents the complete binary tree of height n. Consider the 1-CMD instance $\langle \{A_{s_1} \doteq A_0, A_{s_2} \doteq A_0\}, G\rangle$. Note that, any context C of w_{A_0} with $|\mathtt{hp}(C)| = n$ is a solution for F.

We want to show that all solutions can be computed in polynomial time, but the number of solutions may be exponentially large only due to the choice possibilities of c. For this reason, in the algorithm we have a third component, apart from the set of equations Δ and the STG G, representing the possible elections for the variables F of this kind. This component is a set of expressions of the form $F \in \mathtt{Contexts}(A, A')$, representing that F can be replaced by any context c such that $c[w_{A'}] = w_A$ for unifiers σ. Hence, our algorithm deals with triples $\langle \Delta, G, \Gamma\rangle$ where Γ is the set containing this kind of expressions.

As a last ingredient, we need to adapt the operation `JointCon`, presented in Section 7.1.2, to our representation.

Definition 7.1.8 *Let G be an STG and A, B be two term nonterminals of G such that $w_A \neq w_B$ and $\mathtt{restriction}(G, \{A, B\})$ is a DAG representing ground terms. Let l be a natural number.*

Then, $\mathtt{JointCG}(G, A, B, l)$ is defined as an extension of G recursively as follows. $\mathtt{JointCG}(G, A, B, 0)$ contains G plus the rule $C \to \bullet$, where C is a new context nonterminal. For the case of $\mathtt{JointCG}(G, A, B, l+1)$, if the rules of A and B are of the form $A \to f(A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n)$ and $B \to f(A_1, \ldots, A_{i-1}, B_i, A_{i+1}, \ldots, A_n)$, for some i satisfying $w_{A_i} \neq w_{B_i}$, then $\mathtt{JointCG}(G, A, B, l+1)$ contains $\mathtt{JointCG}(G, A_i, B_i, l)$, which has a context nonterminal C' generating $\mathtt{JointCon}(w_{A_i}, w_{B_i}, l)$, plus the rule $C \to f(A_1, \ldots, A_{i-1}, C', A_{i+1}, \ldots, A_n)$, where C is a new context nonterminal. In any other case, $\mathtt{JointCG}(G, A, B, l+1)$ is undefined.

Lemma 7.1.9 *Let G be an STG and A, B be two term nonterminals of G such that $w_A \neq w_B$ and $\mathtt{restriction}(G, \{A, B\})$ is a DAG representing ground terms. Let l be a natural number. Assume also that $\mathtt{restriction}(G, \{A, B\})$ is compressed optimally, i.e. equal terms are represented by the same term nonterminal.*

Then, $\mathtt{JointCG}(G, A, B, l)$ adds at most $\mathtt{depth}(G)$ new context nonterminals to G, and has one symbol generating $\mathtt{JointCon}(w_A, w_B, l)$. Moreover, all the added context nonterminals C have rules which are of the form $C \to \bullet$ or $C \to f(A_1, \ldots, A_{i-1}, C', A_{i+1}, \ldots, A_n)$, where the terminal f is necessarily a function symbol, i.e. it is not a variable. The time complexity of this construction is $\mathcal{O}(\mathtt{depth}(G))$.

Definition 7.1.10 *Let G be an STG and A, B be two term nonterminals of G such that $w_A \neq w_B$ and* $\texttt{restriction}(G, \{A, B\})$ *is a DAG representing ground terms. Let F be a context variable which is a terminal of arity 1 of G. Let l be a natural number. Assume also that* $\texttt{restriction}(G, \{A, B\})$ *is compressed optimally, i.e. equal terms are represented by the same term nonterminal.*

Then, $\texttt{JointCGF}(G, F, A, B, l)$ *is an STG obtained from* $\texttt{JointCG}(G, A, B, l)$*, which has a context nonterminal C not occurring in G and generating the context* $\texttt{JointCon}(w_A, w_B, l)$*, by transforming F into a context nonterminal, and replacing the nonterminal C by F everywhere. This corresponds to the instantiation of F by the context generated by C.*

Example 7.1.11 *We give a simplified example for the k-CMD algorithm. In particular we illustrate that two equations $F(s_1) \doteq t_1, F(s_2) \doteq t_2$ with $t_1 \neq t_2$ allow a simple guess of the solution of F. Let the equations be $F_1(F_2(F_3(x))) \doteq f(a, f(f(a, b), c))$ and $F_1(F_3(F_2(y))) \doteq f(a, f(a, f(a, c)))$, where F_i, for $i \in \{1, 2, 3\}$, is a context variable and x, y are first-order variables. Then F_1 can only be replaced by $\bullet$ or $f(a, \bullet)$, since $f(f(a, b), c) \neq f(a, f(a, c))$. Let us choose the second possibility. After decomposition we obtain:*
$F_2(F_3(x)) \doteq f(f(a, b), c), F_3(F_2(y)) \doteq f(a, f(a, c))$.
Now F_3 may be the empty context or a nonempty one. Let us guess that it is nonempty. Then we can choose among the equations $F_2(y) \doteq f(a, c), F_2(y) \doteq a, F_2(y) \doteq c$. Let us choose the first one, which leads to the intermediate equations $F_2(F_3(x)) \doteq f(f(a, b), c), F_2(y) \doteq f(a, c)$, with $f(f(a, b), c) \neq f(a, c)$. The only possibility for F_2 is $f(\bullet, c)$, which leads to an instantiation. The equations are, after decomposition:
$F_3(x) \doteq f(a, b), F_3(f(y, c)) \doteq f(a, f(a, c))$.
Using the same scheme, we obtain the solution $F_3 = f(a, \bullet), x = b, y = a$.

Example 7.1.12 *We give another simplified example for the k-CMD algorithm illustrating the case $F(s_1) \doteq t, F(s_2) \doteq t$. Let the equations be $F(f(x, b)) \doteq f(f(a, b), f(f(a, b), f(a, b)))$ and $F(y) = f(f(a, b), f(f(a, b), f(a, b)))$. In principle, F can be instantiated by any subcontext of the right term. There are as many different subcontexts as positions in the right term. But only guessing among the proper subterms, i.e. $\{a, b, f(a, b), f(f(a, b), f(a, b))\}$, we obtain, for a concrete selection, the equations $f(x, b) \doteq f(a, b), y \doteq f(a, b)$. Later, we represent the solutions for the context F as* $\texttt{Contexts}(f(x, b), f(f(a, b), f(f(a, b), f(a, b))))$*. If the right term is represented as a DAG, then the number of positions may be far larger than the number of possible subterms.*

7.1.5 Rules of the k-CMD Algorithm

The k-CMD algorithm is presented in Figures 7.1, 7.2 and 7.3 as a set of transformation rules which deal with triples $\langle \Delta, G, \Gamma\rangle$, where Δ is a set of equations defined over an STG G, where the right hand sides of equations are nonterminals in a DAG representing ground terms, and Γ is a set of expressions each one representing all solutions for a context variable, as described in the previous section. We assume that, initially, equal subterms in the right-hand sides of equations are represented by the same term nonterminal, i.e. optimal DAG compression is used. This will hold during the execution. Given an instance of the problem $\langle \{A_{s_1} \doteq A_{t_1}, \ldots, A_{s_n} \doteq A_{t_n}\}, G\rangle$, the starting triple is $\langle \{A_{s_1} \doteq A_{t_1}, \ldots, A_{s_n} \doteq A_{t_n}\}, G, \emptyset\rangle$, and the constant L occurring in the rules is $\max_{1\leq i\leq n}(\texttt{height}(w_{G,A_{t_i}}))$.

There are two kinds of choices the algorithm can do. On the one side there are the "don't care" selections, which include the strategy stating which rule is applied and the selection of the equations involved in the rule application. On the other side we have the guessings, which make the algorithm nondeterministic. Those correspond to the decisions marked as "guessed" in the conditions of the rules, but also to the selection performed when the resulting part of a rule has a disjunction.

$$\textsc{Unfold1:}\quad \frac{\langle \Delta \cup \{A \doteq B\}, G = (\mathcal{TN} \cup \{A, B\}, \mathcal{CN}, \Sigma, R \cup \{A \to u\}), \Gamma\rangle}{\langle \Delta \cup \{u \doteq B\}, G, \Gamma\rangle}$$

$$\textsc{Unfold2:}\quad \frac{\begin{array}{l}\langle \Delta \cup \{CA \doteq B\}, G = (\mathcal{TN} \cup \{A, B\}, \mathcal{CN} \cup \{C\}, \Sigma,\\ \quad R \cup \{C \to f(A_1, \ldots, C_i, \ldots, A_m)\}), \Gamma\rangle\end{array}}{\langle \Delta \cup \{f(A_1, \ldots, C_i A, \ldots, A_m) \doteq B\}, G, \Gamma\rangle}$$

$$\textsc{Unfold3:}\quad \frac{\langle \Delta \cup \{CA \doteq B\}, G = (\mathcal{TN} \cup \{A, B\}, \mathcal{CN} \cup \{C\}, \Sigma, R \cup \{C \to \bullet\}), \Gamma\rangle}{\langle \Delta \cup A \doteq B\}, G, \Gamma\rangle}$$

Figure 7.1: Unfold-Rules of the k-CMD Algorithm

We differentiate our set of inference rules in two disjoint subsets. We call the first rules *unfolding* rules (see Fig. 7.1), since their purpose is to replace the nonterminals of G occurring in the equations by their definition in G. Hence, these rules are related to our grammar-based representation for DAGs. We refer to the rest of the rules as *solving* rules, since they represent the actual algorithm as described in Section 7.1.3; these are splitted into the first-order rules (see Fig. 7.2) and the context-variable rules (see Fig. 7.3). The application of *solving* rules transforms the set of equations into a new set. Depending on the case, more than one rule can be applied to a given set of equations. Hence, the inference system represents, in fact, a family

Decompose:

$$\frac{\begin{array}{c}\langle \Delta \cup \{f(u_1,\dots,u_m) \doteq B\}, G = (\mathcal{TN} \cup \{B_1,\dots,B_m\},\\ \mathcal{CN}, \Sigma \cup \{f\}, R \cup \{B \to f(B_1,\dots,B_m)\}), \Gamma\rangle\end{array}}{\langle \Delta \cup \{u_1 \doteq B_1, \dots, u_m \doteq B_m\}, G, \Gamma\rangle}$$

where f is a function symbol ($m = \texttt{arity}(f)$)

Fail:

$$\frac{\begin{array}{c}\langle \Delta \cup \{f(u_1,\dots,u_m) \doteq B)\}, G = (\mathcal{TN} \cup \{B_1,\dots,B_{m'}\},\\ \mathcal{CN}, \Sigma \cup \{f,g\}, R \cup \{B \to g(B_1,\dots,B_{m'})), \Gamma\rangle\end{array}}{\perp}$$

where $f \neq g$ are function symbols ($m = \texttt{arity}(f)$, $m' = \texttt{arity}(g)$)).

Elimx:

$$\frac{\langle \Delta \cup \{x \doteq B\}, G = (\mathcal{TN} \cup \{B\}, \mathcal{CN}, \Sigma \cup \{x\}, R), \Gamma\rangle}{\langle \Delta \cup \{x \doteq B\}, G' = (\mathcal{TN} \cup \{B, x\}, \mathcal{CN}, \Sigma, R \cup \{x \to B\}), \Gamma\rangle}$$

where x is a first-order variable and a terminal.

Figure 7.2: First-Order-Rules of the k-CMD Algorithm

of algorithms, depending on the strategy for deciding which rule to apply and to which subset of equations. As commented before, our initial set of equations is of the form $\{A_{s_1} \doteq A_{t_1}, \dots, A_{s_n} \doteq A_{t_n}\}$. But after applying the transformation rules, the form of these equations may change. Nevertheless, at any step of the algorithm the current equations are *simple*, according to the following definition.

Definition 7.1.13 *Let $G = (\mathcal{TN}, \mathcal{TC}, \Sigma, R)$ be an STG, and let $u \doteq v$ be an equation, where $u, v \in \mathcal{T}(\mathcal{TN} \cup \mathcal{TC} \cup \Sigma)$. The equation $u \doteq v$ is called* simple *over G if it is of one of the following forms.*

- $A \doteq B$
- $CA \doteq B$
- $\alpha(A_1, \dots, A_m) \doteq B$
- $f(A_1, \dots, A_{i-1}, C_iA, A_{i+1}, \dots, A_m) \doteq B,$

where A is a term nonterminal of G, B is a term nonterminal of G representing a ground term, C is a context nonterminal of G, and the terms $\alpha(A_1, \dots, A_m)$ and $f(A_1, \dots, A_{i-1}, C_i, A_{i+1}, \dots, A_m)$ are right-hand sides of rules of G, for a terminal α, a terminal f, which is also a function symbol, term nonterminals $A_1, \dots, A_m$, and a context nonterminal C_i. Variables can only occur as some α.

The following lemma shows that no rule of the form $C \to C_1C_2$ occurs in the k-CMD algorithm.

ElimF1:
$$\frac{\begin{array}{c}\langle \Delta \cup \{F(A_1) \doteq B_1, F(A_2) \doteq B_2\},\\ G = (\mathcal{TN} \cup \{A_1, A_2, B_1, B_2\}, \mathcal{CN}, \Sigma \cup \{F\}, R), \Gamma\rangle\end{array}}{\langle \Delta \cup \{F(A_1) \doteq B_1, F(A_2) \doteq B_2\}, G' = \texttt{JointCGF}(G, F, B_1, B_2, l), \Gamma\rangle}$$
where F is a context variable and a terminal and $w_{B_1} \neq w_{B_2}$. l is guessed over $[0, L]$ such that $\texttt{JointCG}(G, B_1, B_2, l)$ is defined.

ElimF2:
$$\frac{\begin{array}{c}\langle \Delta \cup \{F(A_1) \doteq B, F(A_2) \doteq B, \ldots, F(A_n) \doteq B\},\\ G = (\mathcal{TN} \cup \{A_1, \ldots, A_n, B, B'\}, \mathcal{CN}, \Sigma \cup \{F\}, R), \Gamma\rangle\end{array}}{\langle \Delta \cup \{A_1 \doteq B', A_2 \doteq B', \ldots, A_n \doteq B'\}, G, \Gamma \cup \{F \in \texttt{Contexts}(B, B')\}\rangle}$$
where F is a context variable and a terminal not occurring in the w_{A_i}'s, nor w_u, for all equations $u \doteq v \in \Delta$. B' is guessed over the term nonterminals of $\texttt{restriction}(G, \{B\})$.

ElimF3:
$$\frac{\langle \Delta \cup \{F(A) \doteq B\}, G = (\mathcal{TN} \cup \{A, B, B'\}, \mathcal{CN}, \Sigma \cup \{F\}, R), \Gamma\rangle}{\begin{array}{l}\langle \Delta \cup \{F(A) \doteq B\}, G = (\mathcal{TN} \cup \{A, B, B'\}, \mathcal{CN} \cup \{F\},\\ \quad \Sigma, R \cup \{F \to \bullet\}), \Gamma\rangle\\ \mid \langle \Delta \cup \{F(A) \doteq B\}, G' = \texttt{JointCGF}(G, F, B, B', l), \Gamma\rangle\end{array}}$$
where F is a context variable and a terminal occurring in w_A. The term nonterminal B' is guessed over the term nonterminals of $\texttt{restriction}(G, \{B\})$ excluding B, and l is guessed over $[1, L]$ such that $\texttt{JointCG}(G, B, B', l)$ is defined.

ElimF4:
$$\frac{\begin{array}{c}\langle \Delta \cup \{F_1(A_1) \doteq B_1, F_2(A_2) \doteq B_2\},\\ G = (\mathcal{TN} \cup \{A_1, A_2, B_1, B_2, B_2'\}, \mathcal{CN}, \Sigma \cup \{F_1, F_2\}, R), \Gamma\rangle\end{array}}{\begin{array}{l}\langle \Delta \cup \{F_1(A_1) \doteq B_1, F_2(A_2) \doteq B_2\}, G = (\mathcal{TN} \cup \{A_1, A_2, B_1, B_2, B_2'\},\\ \quad \mathcal{CN} \cup \{F_2\}, \Sigma \cup \{F_1\}, R \cup \{F_2 \to \bullet\}), \Gamma\rangle\\ \mid \langle \Delta \cup \{F_1(A_1) \doteq B_1, F_2(A_2) \doteq B_2\}, G' = \texttt{JointCGF}(G, F_1, B_1, B_2', l), \Gamma\rangle\end{array}}$$
where $F_1 \neq F_2$ are context variables that are terminals, F_1 occurs in w_{A_2}, and $\texttt{height}(w_{B_1}) \geq \texttt{height}(w_{B_2})$. B_2' is guessed over the term nonterminals of $\texttt{restriction}(G, \{B_2\})$ excluding B_2, and l is guessed over $[0, L]$ such that $\texttt{JointCG}(G, B_1, B_2', l)$ is defined.

Figure 7.3: Elim-F-Rules of the k-CMD Algorithm

Lemma 7.1.14 *Let $\langle \Delta, G, \Gamma\rangle$ be a triple obtained by our algorithm at any point of the execution. Then, the rules of G are of the following forms.*

- $A \to A_1$
- $A \to \alpha(A_1, \ldots, A_m)$,
- $A \to CA_1$
- $C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)$
- $C \to \bullet$

where $A, A_1, A_2, \ldots, A_m$ are term nonterminals of G, C, C_i are context nonterminals of G, α is a terminal of G, and f is a terminal of G, which is also a function symbol.

Proof. We prove the lemma by induction on the number of applied inference rules. For the base case, note that the lemma holds for the STG G_0 given as input since, by Definitions 7.1.5 and 7.1.6, all the rules in G_0 are of the form $A \to \alpha(A_1, \ldots, A_m)$ for some term nonterminals $A_1, \ldots, A_m$ and a terminal α of G_0.

For the inductive case, let $\langle \Delta', G', \Gamma' \rangle$ be the triple from which $\langle \Delta, G, \Gamma \rangle$ was obtained by an inference rule application. By induction hypothesis $\langle \Delta', G', \Gamma' \rangle$ satisfies the conditions of the lemma. We distinguish cases according to the inference rule applied to $\langle \Delta', G', \Gamma' \rangle$ in order to show that the rules in G follow the conditions of the lemma. Note that for the inference rules that do not modify the STG (*unfolding* rules, `Decompose`, `Fail`, and ElimF2), this is straightforward. Otherwise, if Elimx was the applied rule, x became a term nonterminal and a rule of the form $x \to A$ was added to G' for some terminal x representing a first-order variable and term nonterminal A. Note that the added rule satisfies the conditions of the lemma. Finally, if the applied rule was either ElimF1, ElimF3, or ElimF4 then either G was extended by the `JointCon` construction or a rule $F \to \bullet$ was added to G', for some context variable F. By Lemma 7.1.9, in both cases all the added rules satisfy the condition of the lemma. □

Lemma 7.1.15 *Let $\langle \Delta, G, \Gamma \rangle$ be the triple obtained by our algorithm at a point of the execution. Then, the set Δ consists of* simple *equations over G.*

Proof. Since for the triple given as input $\langle \Delta_0, G_0, \Gamma_0 = \emptyset \rangle$ all the equations in Δ are of the form $A_s \doteq A_t$ for some term nonterminals A_s, A_t in G_0, the statement of the Lemma holds in this case. Hence, for proving this lemma it suffices to check that after an inference step where $\langle \Delta, G, \Gamma \rangle$ was obtained from a triple $\langle \Delta', G', \Gamma' \rangle$, each new equation in Δ is *simple* over G. Checking this is an easy task for rules `Fail`, Elimx, ElimF1, ElimF2, ElimF3, ElimF4, Unfold2, and Unfold3, since the new produced equations are explicitly defined. For `Decompose` the result follows by induction hypothesis. Finally, the produced equations due to the application of Unfold1 are of the form $u \doteq B$, where B is a term nonterminal and u corresponds to a right-hand side of a rule in G' and hence, they satisfy the condition to be simple over G due to Lemma 7.1.14. □

Before proving soundness, completeness and termination of our inference system we should define a notion of solution of the triples the k-CMD algorithm deals with.

Definition 7.1.16 *A solution of $\langle \Delta, G, \Gamma \rangle$ is a substitution σ such that $\sigma(w_{G,u}) = w_{G,v}$ for each equation $u \doteq v$ in Δ, $\sigma(w_{G,x}) = \sigma(x)$ for each* *first-order variable x, and $\sigma(w_{G,F}) = \sigma(F)$ for each context variable F. Furthermore, for each context variable F such that $(F \in \mathtt{Contexts}(A_1, A_2)) \in \Gamma$, where A_1 and A_2 are term nonterminals of G, it holds that $\sigma(F)\sigma(w_{G,A_2}) = w_{G,A_1}$.*

Let $\langle \Delta, G, \Gamma \rangle$ be a triple generated by our algorithm at any point of the execution. Note that some of the variables may have been isolated and, hence, the STG G was extended in order to represent the corresponding instantiations. As stated in the previous definition, a solution of $\langle \Delta, G, \Gamma \rangle$ has to be consistent with this extensions. The following lemma, together with the definition of a solution σ of $\langle \Delta, G, \Gamma \rangle$, states that our representation for partial solutions by extending the grammar is correct in the sense that the same term is obtained by applying a solution to the term generated by G before and after such an extension. It will be helpful when proving soundness and completeness.

Lemma 7.1.17 *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma \cup \{V\}, R)$ be an STG obtained at any point of the execution of the k-CMD algorithm. Let V be a terminal of G representing either a first-order or a context variable. Let G' be the STG obtained from G by converting V into a nonterminal of the grammar and adding some new rules and nonterminals such that V generates a certain term or context $w_{G',V}$. Let σ be a substitution such that $\sigma(V) = \sigma(w_{G',V})$. Let t be a term in $\mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma \cup \{V\})$ or a context in $\mathcal{C}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma \cup \{V\})$. Then, $\sigma(w_{G,t}) = \sigma(w_{G',t})$.*

Proof. The proof is an easy induction on the size of t and the number of rule applications to derive $w_{G,t}$. □

Lemma 7.1.18 *The set of rules is sound.*

Proof. Let $\langle \Delta', G', \Gamma' \rangle$ be the triple obtained by our algorithm by applying an inference step on $\langle \Delta, G, \Gamma \rangle$. By inspecting the rules, we can check that every solution σ of $\langle \Delta', G', \Gamma' \rangle$ is also a solution of $\langle \Delta, G, \Gamma \rangle$: We distinguish cases depending on which rule was applied for obtaining $\langle \Delta', G', \Gamma' \rangle$ from $\langle \Delta, G, \Gamma \rangle$.

Note that the rules Elimx, ElimF1, ElimF3 and ElimF4 instantiate either a first-order or a context variable V. Therefore, if one of those rules was the rule applied to $\langle \Delta, G, \Gamma \rangle$ then G' was obtained from G by transforming V into a nonterminal of the STG and adding some nonterminals and their corresponding rules such that V generates $w_{G',V}$. By Definition 7.1.16, for being a solution of $\langle \Delta', G', \Gamma' \rangle$, σ satisfies $\sigma(V) = \sigma(w_{G',V})$. Hence, G and G' satisfy the conditions of Lemma 7.1.17 and we can conclude $\sigma(w_{G,t}) =$

$\sigma(w_{G',t})$ for every term t in $\mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$, where $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$. It follows that $\sigma(x) = \sigma(w_{G,x})$ for every first-order variable x, and $\sigma(F) = \sigma(w_{G,F})$ for every context variable F. Moreover, since none of these rules changed neither the set Δ nor Γ, σ is also a solution for $\langle \Delta, G, \Gamma \rangle$.

Suppose the rule applied is ElimF2. In this case, $G' = G$ but both sets Δ and Γ are changed. Concretely, a set of equations of the form $\{F(A_1) \doteq B, F(A_2) \doteq B, \ldots, F(A_n) \doteq B\}$ of Δ is replaced by a set of equations of the form $\{A_1 \doteq B', A_2 \doteq B', \ldots, A_n \doteq B'\}$ to obtain Δ' and the restriction $F \in \texttt{Contexts}(B, B')$ was added to Γ to obtain Γ'. By Definition 7.1.16, since σ is a solution of $\langle \Delta', G', \Gamma' \rangle$, it holds $\sigma(w_{G',A_i}) = w_{G',B'}$ for each $i \in [1, n]$, and $\sigma(F)w_{G',B'} = w_{G',B}$. Since $G = G'$ and $\Delta - \{F(A_i) \doteq B \mid i \in [1, n]\} = \Delta' - \{A_i \doteq B' \mid i \in [1, n]\}$, it suffices to prove that $\sigma(w_{G,F(A_i)}) = w_{G,B}$ for $i \in [1, n]$ to show that σ is also a solution of $\langle \Delta, G, \Gamma \rangle$. Since $G = G'$ and $\sigma(w_{G',A_i}) = w_{G',B'}$ then $\sigma(w_{G,A_i}) = w_{G,B'}$ holds. Furthermore, it holds that $\sigma(w_{G,F(A_i)}) = \sigma(F(w_{G,A_i})) = \sigma(F)\sigma(w_{G,A_i}) = \sigma(F)\sigma(w_{G,B'}) = \sigma(F)\sigma(w_{G',B'}) = w_{G',B} = w_{G,B}$. Hence, we proved that $\sigma(w_{G,F(A_i)}) = w_{G,B}$ and thus σ is also a solution of $\langle \Delta, G, \Gamma \rangle$.

For rule `Fail`, it is obvious that the assumption of a solution σ for the resulting triple $\langle \Delta', G', \Gamma' \rangle$ cannot be satisfied.

Suppose the rule applied is `Decompose`. Then, $G' = G$, $\Gamma = \Gamma'$ and an equation $f(u_1, \ldots, u_m) \doteq B$ in Δ where $B \to f(B_1, \ldots, B_m)$ is the rule in G is replaced by the equations $u_1 \doteq B_1, \ldots, u_m \doteq B_m$ to obtain Δ'. Hence, it suffices to prove that $\sigma(w_{G,f(u_1,\ldots,u_m)}) = w_{G,f(B_1,\ldots,B_m)}$ in order to show that σ is also a solution for $\langle \Delta, G, \Gamma \rangle$. Since σ is a solution of $\langle \Delta', G', \Gamma' \rangle$ it holds $\sigma(w_{G',u_1}) = w_{G',B_1}, \ldots, \sigma(w_{G',u_m}) = w_{G',B_m}$. Thus, $\sigma(w_{G,f(u_1,\ldots,u_m)}) = \sigma(w_{G',f(u_1,\ldots,u_m)}) = f(\sigma(w_{G',u_1}), \ldots, \sigma(w_{G',u_m})) = f(w_{G',B_1}, \ldots, w_{G',B_m}) = f(w_{G,B_1}, \ldots, w_{G,B_m}) = w_{G,f(B_1,\ldots,B_m)}$.

In the case where the rule applied is an *unfolding rule*, note that these rules just replace nonterminals of G by their definition in G. Hence, since $w_N = w_\alpha$ for each nonterminal N with a rule $N \to \alpha \in G$, every solution of $\langle \Delta', G', \Gamma' \rangle$ is also a solution of $\langle \Delta, G, \Gamma \rangle$. □

The following lemma is an adaptation of Lemma 7.1.4 to our STG-based representation for DAGs, which will be helpful when proving completeness.

Lemma 7.1.19 *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ be an STG. Let u_1, u_2 be terms in $\mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$. Let B_1, B_2 be term nonterminals of G such that $w_{G,B_1} \neq w_{G,B_2}$ and both w_{G,B_1} and w_{G,B_2} are ground. Let* $\texttt{restriction}(G, \{B_1, B_2\})$ *be compressed optimally as a DAG. Let σ be a solution of $\langle \{F(u_1) \doteq B_1, F(u_2) \doteq B_2\}, G, \Gamma \rangle$ where the context variable F is a terminal of G. Let $G' = \texttt{JointCGF}(G, F, B_1, B_2, |\texttt{hp}(\sigma(F))|)$. Then, $\sigma(F) = w_{G',F}$.*

Proof. This lemma directly follows from Lemma 7.1.4 and Definition 7.1.10. □

Rules can be applied in a "don't care"-fashion, whereas the selections within the rules are "don't know"-nondeterministic:

Lemma 7.1.20 *For every solution σ of $\langle \Delta, G, \Gamma \rangle$, and for every rule application, there is a result $\langle \Delta', G', \Gamma' \rangle$ such that σ is also a solution of $\langle \Delta', G', \Gamma' \rangle$. Moreover, any maximal sequence of rule applications computes a representation of all solutions, by gathering all guesses and alternatives in the rules.*

Proof. Let σ be a solution for some triple $\langle \Delta, G, \Gamma \rangle$ obtained by our algorithm. It suffices to show that after applying any applicable rule to $\langle \Delta, G, \Gamma \rangle$, one of the resulting triples $\langle \Delta', G', \Gamma' \rangle$ among the possible guesses also has σ as solution. We distinguish cases depending on which inference step was applied for obtaining $\langle \Delta', G', \Gamma' \rangle$ from $\langle \Delta, G, \Gamma \rangle$. We state explicitly here $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ because it will be necessary, in some cases, to refer to the set of terms $\mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$.

Assume the applied rule is `Decompose`. Then, $G' = G$, $\Gamma = \Gamma'$ and an equation $f(u_1, \ldots, u_m) \doteq B$ in Δ with rule $B \to f(B_1, \ldots, B_m)$ is replaced by the equations $u_1 \doteq B_1, \ldots, u_m \doteq B_m$ to obtain Δ', where each $u_i \in \mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$. Hence, it suffices to prove $\sigma(w_{G',u_1}) = w_{G',B_1}, \ldots, \sigma(w_{G',u_m}) = w_{G',B_m}$ in order to show that σ is also a solution for $\langle \Delta', G', \Gamma' \rangle$. Since σ is a solution of $\langle \Delta, G, \Gamma \rangle$, it holds that $\sigma(w_{G,f(u_1,\ldots,u_m)}) = w_{G,f(B_1,\ldots,B_m)}$ which implies $\sigma(f(w_{G,u_1}, \ldots, w_{G,u_m})) = f(w_{G,B_1}, \ldots, w_{G,B_m})$, and hence $\sigma(w_{G,u_1}) = w_{G,B_1}, \ldots, \sigma(w_{G,u_m}) = w_{G,B_m}$. Finally, since $G = G'$, σ is also a solution of $\langle \Delta', G', \Gamma' \rangle$.

Assume the applied rule is ElimX. Then $\Gamma = \Gamma'$ and $\Delta = \Delta'$. For a concrete equation $x \doteq B \in \Delta$, G was extended to G' by converting x into a term nonterminal and adding the rule $x \to B$. Since σ is a solution of $\langle \Delta, G, \Gamma \rangle$ and x is a terminal of G, $w_{G,x} = x$ and $\sigma(x) = w_{G,B}$ holds. Furthermore, $w_{G',x} = w_{G',B} = w_{G,B}$ since B is the definition of x in G' and none of the rules of G were changed to obtain G'. Hence, $\sigma(x) = w_{G,B} = w_{G',B} = w_{G',x} = \sigma(w_{G',x})$, where the last equality holds because $w_{G',x}$ is ground. Thus, we can apply Lemma 7.1.17 and claim that, for every term t in $\mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$, $\sigma(w_{G,t}) = \sigma(w_{G',t})$. Hence, since $\Gamma = \Gamma'$ and $\Delta = \Delta'$, σ is also a solution for $\langle \Delta', G', \Gamma' \rangle$.

For the `Fail` rule it is clear that the assumption on the existence of a solution cannot be satisfied.

Suppose that the applied rule is ElimF1. In this case, $\Delta = \Delta'$, $\Gamma = \Gamma'$ and G was extended to G' by converting the terminal F, which is a context variable, into a context nonterminal. Some rules and nonterminals were added such that F generates a ground context $w_{G',F}$. We first show that $\sigma(F) = \sigma(w_{G',F})$ holds for one of the possible guesses when applying this rule.

Since $|\mathtt{hp}(\sigma(F))|$ is smaller than or equal to L ($|\mathtt{hp}(\sigma(F))| \in [0, L]$) we can assume that l is guessed as $|\mathtt{hp}(\sigma(F))|$ in the rule application. Then, by the conditions for this rule application, there are equations of the form $F(A_1) \doteq B_1, F(A_2) \doteq B_2$ in Δ such that $w_{B_1} \neq w_{B_2}$. Furthermore, both w_{B_1} and w_{B_2} are ground and G' is constructed as $\mathtt{JointCGF}(G, F, B_1, B_2, |\mathtt{hp}(\sigma(F))|)$. Hence, by Lemma 7.1.19, $\sigma(F) = w_{G',F}$. Moreover, we can apply Lemma 7.1.17 and conclude that $\sigma(w_{G,t}) = \sigma(w_{G',t})$ for every term $t \in \mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$. Thus, σ is a solution of $\langle \Delta', G', \Gamma' \rangle$.

Suppose now that the applied rule is ELIMF2. In this case, $G = G'$, and some equations of the form $F(A_1) \doteq B, F(A_2) \doteq B, \ldots, F(A_n) \doteq B$ of Δ such that F does not occur in w_u for any other equation $u \doteq v$ in Δ were replaced by the equations $A_1 \doteq B', \ldots, A_n \doteq B'$ to obtain Δ'. Moreover, the restriction $F \in \mathtt{Contexts}(\mathtt{B}, \mathtt{B}')$ was added to Γ to obtain Γ'. Since σ is a solution of $\langle \Delta, G, \Gamma \rangle$, it is also a solution of $\{F(A_1) \doteq B, F(A_2) \doteq B, \ldots, F(A_n) \doteq B\}$. Hence, there exists some subterm $w_{G,B'}$ of $w_{G,B}$ satisfying $\sigma(w_{G,A_1}) = w_{G,B'}, \ldots, \sigma(w_{G,A_n}) = w_{G,B'}$ which corresponds to $w_{G,B}|_{\mathtt{hp}(\sigma(F))}$. In our representation choosing a subterm of $w_{G,B}$ is equivalent to choosing one of the term nonterminals of $\mathtt{restriction}(G, \{B\})$. Thus, we can consider the case where B' is the term nonterminal guessed in the rule application. In this case $\sigma(w_{G',A_1}) = w_{G',B'}, \ldots, \sigma(w_{G',A_n}) = w_{G',B'}$ holds since $G = G'$. Therefore, σ is also a solution for $\langle \Delta', G', \Gamma' \rangle$. With respect to $\sigma(F)$, it satisfies $\sigma(F) w_{G,B'} = w_{G,B}$, which is exactly the condition added to Γ by the rule application in order to keep a representation of all possible instantiations for the context variable F.

Suppose that the applied rule is ELIMF3. In this case, $\Delta = \Delta'$, $\Gamma = \Gamma'$ and G was extended to G' by converting a terminal F representing a context variable into a context nonterminal. Some rules and nonterminals were added such that F generates the ground term $w_{G',F}$. We first show that $\sigma(F) = \sigma(w_{G',F})$ holds for one of the possible guesses when applying this rule.

By the condition of this rule application, $F(A) \doteq B$ is an equation in Δ where F occurs in w_A. The case $\sigma(F) = \bullet$ is covered by the first alternative of the rule. Now assume that $\sigma(F) \neq \bullet$. Since F occurs in $w_{G,A}$, there exists a proper subterm of $w_{G,F(A)}$ (a subterm of $w_{G,A}$) of the form $F(u)$ for some term $u \in \mathcal{T}(\Sigma)$. Since $\sigma(F(w_{G,A})) = w_{G,B}$ holds and $\sigma(F) \neq \bullet$, there exists a proper subterm $w_{G,B'}$ of $w_{G,B}$ such that $\sigma(F(u)) = w_{G,B'}$ and, for the same reason as in the previous case, B' is a term nonterminal in $\mathtt{restriction}(\mathtt{G}, \{\mathtt{B}\})$ excluding B. We consider the case where the term nonterminal B' is guessed by the rule application and l is guessed as $|\mathtt{hp}(\sigma(F))|$. When these two guesses are done, G' is constructed as $\mathtt{JointCGF}(G, F, B, B', |\mathtt{hp}(\sigma(F))|)$. Furthermore, we know that σ satisfies $\sigma(F(u)) = w_{G,B'}$ and $\sigma(w_{G,F(A)}) = w_{G,B}$. Moreover, $w_{G,B'}$ and $w_{G,B}$ are ground, and $w_{B'} \neq w_B$, since $w_{B'}$ is a proper subDAG of w_B. Hence, we can apply Lemma 7.1.19 and conclude $\sigma(F) = w_{G',F}$. As before, we can apply Lemma 7.1.17 and conclude that σ is a solution of $\langle \Delta', G', \Gamma' \rangle$.

Suppose that the applied rule is ElimF4. In this case, $\Delta = \Delta'$, $\Gamma = \Gamma'$ and G was extended to G' by either converting a terminal F_2 or a terminal $F_1 \neq F_2$, each of them representing a context variable, into a context nonterminal. Each of these cases corresponds to one of the two alternatives of the rule. In the first case the rule $F_2 \to \bullet$ was added, such that F_2 generates $w_{G',F_2} = \bullet$, the empty context. In the second case some rules and nonterminals were added, such that F_1 generates the ground context w_{G',F_1}. We first show that either $\sigma(F_2) = \sigma(w_{G',F_2})$, in the former case, or $\sigma(F_1) = \sigma(w_{G',F_1})$ in the latter case.

By the condition of the application of ElimF4, there is a pair of equations in Δ of the form $F_1(A_1) \doteq B_1$ and $F_2(A_2) \doteq B_2$. Furthermore, F_1 occurs in w_{G,A_2}, and $\mathtt{height}(w_{G,B_1}) \geq \mathtt{height}(w_{G,B_2})$. The case $\sigma(F_2) = \bullet$ is covered by the first alternative of the rule, and it is obvious that $\sigma(F_2) = \sigma(w_{G',F_2}) = \bullet$ holds in this case. Now assume that $\sigma(F_2) \neq \bullet$. Since F_1 occurs in w_{G,A_2}, there exists a proper subterm of $w_{G,F_2(A_2)}$ (a subterm of w_{G,A_2}) of the form $F_1(u)$, for some $u \in \mathcal{T}(\mathcal{TN} \cup \mathcal{CN} \cup \Sigma)$. Moreover, since $\sigma(w_{G,F_2(A_2)}) = w_{G,B_2}$ holds, and $\sigma(F_2) \neq \bullet$, there exists a proper subterm w_{G,B'_2} of w_{G,B_2} such that $\sigma(F_1(u)) = w_{G,B'_2}$ and, for the same reason as in the previous case, B'_2 is represented by a term nonterminal in $\mathtt{restriction(G,\{B_2\})}$ excluding B_2, since the subterm is proper. We consider the case where the term nonterminal B'_2 is guessed by the rule application and l is guessed as $|\mathtt{hp}(\sigma(F_1))|$. Hence, G' is constructed as $\mathtt{JointCGF}(G, F_1, B_1, B'_2, |\mathtt{hp}(\sigma(F_1))|)$. We know that σ has to satisfy $\sigma(w_{F_1(A_1)}) = w_{G,B_1}$ and $\sigma(F_1(u)) = w_{G,B'_2}$. Moreover, w_{G,B_1} and w_{G,B'_2} are ground, and $w_{G,B_1} \neq w_{G,B'_2}$ holds, since $\mathtt{height}(w_{G,B_1}) \geq \mathtt{height}(w_{G,B_2})$ and w_{G,B'_2} is a proper subterm of w_{G,B_2}. Hence, by Lemma 7.1.19, $\sigma(F_1) = w_{G',F_1}$. As before, we can apply Lemma 7.1.17 and conclude that σ is a solution of $\langle \Delta', G', \Gamma' \rangle$.

Finally, assume that the applied rule is an *unfolding* rule. Note that the application of an *unfolding* rule does not modify the set of restrictions Γ nor the grammar G. Δ' is obtained by replacing the left-hand side of an equation $u \doteq v$ in Δ by a new equation $u' \doteq v$. Since $G = G'$, it holds that $w_{G,u} = w_{G',u}$. Moreover, since σ is a solution of $\langle \Delta, G, \Gamma \rangle$, it satisfies

$\sigma(w_{G,u}) = w_{G,v}$. Hence, it suffices to check that $w_{G,u} = w_{G,u'}$ when $u \doteq v$ is replaced by $u' \doteq v$ by the rule application. But this is direct from the fact that this replacements are due to a rule application of G, and we are done. □

Definition 7.1.21 *A triple $\langle \Delta, G, \Gamma \rangle$ is* solved *if there are no occurrences of terminals of G representing first-order or context variables in Δ.*

Proposition 7.1.22 *For every initial triple $\langle \Delta_0, G_0, \Gamma_0 = \emptyset \rangle$, the determinized algorithm will compute a complete set of solved triples $\langle \Delta_1, G_1, \Gamma_1 \rangle, \ldots, \langle \Delta_n, G_n, \Gamma_n \rangle$, such that σ is a solution of $\langle \Delta_0, G_0, \Gamma_0 = \emptyset \rangle$ iff it is a solution of some $\langle \Delta_i, G_i, \Gamma_i \rangle$, for $i \in [1, n]$.*

Proof.

Termination holds, see the argumentation on the complexity in the next section. Since we have proved soundness and completeness, it remains to show that if some intermediate $\langle \Delta, G, \Gamma \rangle$ is not solved, then an inference rule can be applied. The k-CMD algorithm represents instantiations of variables by transforming them into nonterminals of the STG. Hence, the fact that a triple $\langle \Delta, G, \Gamma \rangle$ is not solved means that there are occurrences of terminals of G representing first-order variables or context variables in Δ (Definition 7.1.21).

Assume that no inference rule can be applied. We will deduce the form of the equations $u \doteq B \in \Delta$ under this assumption until we reach a contradiction. Let $A, A_1, \ldots, A_m, B, B_1, \ldots, B_m$ be term nonterminals of G, let C_i, C be context nonterminals of G, and let f, g be terminals of G representing function symbols of arity m and m', respectively.

Note that u cannot be of the form $f(A_1, \ldots, A_m)$ nor $f(A_1, \ldots, A_{i-1}, C_i A, A_{i+1}, \ldots, A_m)$ since, as v is a nonterminal B with rule $B \to g(B_1, \ldots, B_{m'})$, either `Decompose` (if $f = g$), or `Fail` (if $f \neq g$) would be applicable. Hence, by 7.1.15, at this point u can be of the forms x, $F(A)$ or CA, where x is a terminal of the grammar representing a first-order variable and F is a terminal of the grammar representing a context variable. This implies that, if $u = x$ then ElimX is applicable, and if $u = CA$ then Unfold2 is applicable. Thus, u can only be of the form $F(A)$. Hence, since we argued about an arbitrarily chosen equation $u \doteq v \in \Delta$, every equation i in Δ is of the form $F_i(A_i) \doteq B_i$. Moreover, since neither ElimF1,ElimF2 nor ElimF3 can be applied, for every terminal F_i representing a context variable occurring in Δ there exists an equation $F_j(A_j) \doteq B_j$ in Δ, such that F_i is different from the terminal F_j, and F_i occurs in w_{A_j}. Since the set Δ is finite, there exist equations $F_1(A_1) \doteq B_1, F_2(A_2) \doteq B_2, \ldots, F_n(A_n) \doteq B_n$

with $n \geq 2$ satisfying that F_1 occurs in w_{A_2}, F_2 occurs in w_{A_3}, $\ldots$, F_{n-1} occurs in w_{A_n}, and F_n occurs in w_{A_1}, and where the F_i's are pairwise different. Let i be such that w_{B_i} has maximal height among the $w_{B_1}, \ldots, w_{B_n}$, say $i = 1$. Hence, we may take the equation $F_1(A_1) \doteq B_1$ and the equation $F_2(A_2) \doteq B_2$ and apply rule ElimF4, which is a contradiction. □

The following example shows that the `Decompose` rule may have an exponential number of executions if multiple insertions of the same equation in one inference sequence are allowed. Hence, our algorithm must keep track of already treated equations in order to avoid this execution sequence.

Example 7.1.23 *Let G be an STG defined by the following set of rules: $\{B_1 \to f(B_2, B_2), B_2 \to f(B_3, B_3), \ldots, B_i \to f(B_{i+1}B_{i+1}), \ldots, B_{n-1} \to f(B_n, B_n), B_n \to a, A_1 \to f(A_2, A_2'), A_2 \to f(A_3, A_3'), \ldots, A_i \to f(A_{i+1}A_{i+1}'), \ldots, A_{n-1} \to f(A_n, A_n'), A_n \to a, A_1' \to f(A_2, A_2'), A_2' \to f(A_3, A_3'), \ldots, A_i' \to f(A_{i+1}A_{i+1}'), \ldots, A_{n-1}' \to f(A_n, A_n'), A_n' \to a, A \to f(A_1, A_1'), B \to f(B_1, B_1)\}$ We now consider a decomposition sequence for the equation $A \doteq B$, it decomposes depth-first. Note that G satisfies the assumption on an optimally compressed representation of* `restriction`$(G, \{B\})$*, whereas the representation of A is not optimally compressed.*

$$\begin{aligned}
&\{A \doteq B\} \Longrightarrow \{f(A_1, A_1') \doteq B\}\\
&\Longrightarrow \{A_1 \doteq B_1, A_1' \doteq B_1\}\\
&\Longrightarrow \{f(A_2, A_2') \doteq B_1, A_1' \doteq B_1\}\\
&\Longrightarrow \{A_2 \doteq B_2, A_2' \doteq B_2, A_1' \doteq B_1\}\\
&\Longrightarrow \{f(A_3, A_3') \doteq B_2, A_2' \doteq B_2, A_1' \doteq B_1\}\\
&\Longrightarrow \{A_3 \doteq B_3, A_3' \doteq B_3, A_2' \doteq B_2, A_1' \doteq B_1\}\\
&\vdots\\
&\Longrightarrow \{A_i \doteq B_i, A_i' \doteq B_i, A_{i-1}' \doteq B_{i-1}, \ldots, A_1' \doteq B_1\}\\
&\vdots\\
&\Longrightarrow \{A_n \doteq B_n, A_n' \doteq B_n, A_{n-1}' \doteq B_{n-1}, \ldots, A_1' \doteq B_1\}\\
&\Longrightarrow \{a \doteq a, a \doteq a, A_{n-1}' \doteq B_{n-1}, \ldots, A_1' \doteq B_1\}
\end{aligned}$$

Hence, the depth-first strategy may lead to an exponentially long sequence of decompositions.

7.1.6 Complexity of the k-CMD Algorithm

Let $\langle \Delta = \{A_{s_1} \doteq A_{t_1}, \ldots, A_{s_n} \doteq A_{t_n}\}, G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R), \Gamma = \emptyset\rangle$ be the *initial configuration* of the execution, and let $\langle \Delta', G', \Gamma' \rangle$ be the last one.

Recall that $L = \max_{1\leq i\leq n}(\texttt{height}(w_{G,A_{t_i}}))$ and k denotes the number of different context variables in the problem. Let V denote the set of first-order variables.

Our inference rules may add new nonterminals and their corresponding rules to the grammar. Concretely, at most $|V|$ rules of the form $x \to A$ and at most kL rules of the forms $C \to f(A_1, \dots, A_{i-1}, C_i, A_{i+1}, \dots, A_m)$ and $C \to \bullet$ are added to G during an execution. Therefore, at any point of the execution, any right-hand side of a rule of the current STG G'' of the form $f(A_1, \dots, A_m)$ is in fact a right-hand side of a rule of the initial G.

We count the number of different equations $u \doteq v$ that may appear during the execution. Our equations are simple with respect to the final G' by Lemma 7.1.15. Thus, u is either of the form A ($|\mathcal{TN}| + |V|$ possibilities), or $f(A_1, \dots, A_m)$ (an original right-hand side of a rule, thus $|\mathcal{TN}|$ possibilities), or CA ($kL(|\mathcal{TN}| + |V|)$ possibilities), or $f(A_1, \dots, A_{i-1}, C_iA, A_{i+1}, \dots, A_m)$ ($kL(|\mathcal{TN}| + |V|)$ possibilities).

On the other hand, v can only be a term nonterminal A, an original term nonterminal, thus there are $|\mathcal{TN}|$ possibilities.

Therefore, the total number of different equations in a branch of nondeterministic execution is $\mathcal{O}(\texttt{depth}(G)|G|^2)$. Assuming we avoid repetition of equations, this will also be the maximum number of execution steps. Each of those steps chooses an equation and applies an inference rule to it. The corresponding operations can be performed in logarithmic time with the adequate data structures. Thus, the nondeterministic execution time is $\mathcal{O}(\texttt{depth}(G)|G|^2\texttt{log}(|G|))$.

k guessings over L possibilities are done during the execution. Therefore, the execution time of the deterministic version of this algorithm is $\mathcal{O}((\texttt{depth}(G))^{k+1}|G|^2\texttt{log}(|G|))$.

Theorem 7.1.24 *Computing all solutions (and hence deciding solvability) of an instance of the k-context matching with DAGs problem can be done in polynomial time. The worst case running time is* $\mathcal{O}((\texttt{depth}(G))^{k+1}|G|^2\texttt{log}(|G|))$, *where k is the number of context variables and* $|G|$ *is the size of the input DAG.*

7.2 Compressed Context Matching

As a complement to Theorem 7.1.24, we are now ready to show that context matching with STG-compressed terms is NP-complete. NP-hardness with STGs follows from NP-hardness of the same problem without any compression (see [SSS98]). Hence, we just have to prove that this problem is in NP.

Our goal is to be able to guess a solution of polynomial size for a given input context matching problem, and to check it efficiently. To this end, we will use once again the constructions defined in Chapter 3 to define applications of substitutions, subterms, and subcontexts in the STG setting.

Let us remark on how we represent the input and the solutions for this problem. An input consists of an STG G and two nonterminals A_s an A_t of G. We want to decide whether there exists a substitution σ for the first-order and context variables occurring in w_{A_s} such that $\sigma(w_{A_s}) = w_{A_t}$. In the input of the algorithm, the first-order and the context variables are 0-ary and 1-ary terminals of G, respectively. A solution σ can be represented by another STG G', where the first-order and the context variables are term and context nonterminals of G', respectively. That is, $\sigma(x) = w_{G',x}$ and $\sigma(F) = w_{G',F}$, for each first-order variable x and context variable F. For proving NP inclusion, we just show that, if such σ exists then there exists an extension G' of G, which is polynomially bounded in the size of G, satisfying $w_{G',A_s} = w_{G',A_t} = w_{G,A_t}$. The fact that this equality can be checked in polynomial time follows again from Theorem 3.2.16. In the proof of the following Lemma we use the construction kExt and pCon defined in Section 3.2.1.

Lemma 7.2.1 *Let G be an STG, and let A_s and A_t be term nonterminals of G. Let $\langle A_s, A_t, G\rangle$, be an STG-context-matching problem instance, and let σ be a substitution such that $\sigma(w_{G,A_s}) = w_{G,A_t}$ (a solution). Then, there exists an extension G' of G such that $w_{G',A_s} = w_{G',A_t} = w_{G,A_t}$. Furthermore, $|G'|$ is polynomially bounded by $|G|$.*

Proof. Let $\{x_1, \ldots, x_n\}$ and $\{F_1, \ldots, F_m\}$ be the set of first-order variables and context variables, respectively, occurring in w_{G,A_s}. For each first-order variable x_i, $\sigma(x_i)$ is a subterm of w_{A_t} at some position p_i. Thus, for each first-order variable x_i, we construct the STG $G'_{x_i} = (\mathcal{TN}'_{x_i}, \mathcal{CN}'_{x_i}, \Sigma'_{x_i}, R'_{x_i})$ as $\texttt{kExt}(A_t, G, \texttt{pIndex}(w_{A_t}, p_i))$, which contains a term nonterminal A_{x_i} generating $w_{A_{x_i}} = \sigma(x_i)$. Then we convert x_i into a nonterminal generating $\sigma(x_i)$ by defining $G_{x_i} = (\mathcal{TN}_{x_i}, \mathcal{CN}_{x_i}, \Sigma_{x_i}, R_{x_i})$ from G'_{x_i} as $G_{x_i} = (\mathcal{TN}'_{x_i} \cup \{x_i\}, \mathcal{CN}'_{x_i}, \Sigma'_{x_i} - \{x_i\}, R'_{x_i} \cup \{x_i \to A_{x_i}\})$. Similarly, for each context variable F_j, $\sigma(F_j) = C$ is a prefix context of some subterm of $t = w_{A_t}$. Therefore, there exist positions q_j, q'_j satisfying that C is the prefix context of $t|_{q_j}$ with the hole at position q'_j. Thus, for each context variable F_j, we construct $G'_{F_j} = (\mathcal{TN}'_{F_j}, \mathcal{CN}'_{F_j}, \Sigma'_{F_j}, R'_{F_j})$ as $\texttt{pCon}(\texttt{kExt}(A_t, G, \texttt{pIndex}(t, q_j)), A_{F_j}, q'_j)$, where $\texttt{kExt}(A_t, G, \texttt{pIndex}(t, q_j))$ contains a term nonterminal A_{F_j} generating $t|_{q_j}$, and G'_{F_j} contains a context nonterminal C_{F_j} generating $\sigma(F_j)$. Then we convert F_j into a context nonterminal by defining $G_{F_j} = (\mathcal{TN}_{F_j}, \mathcal{CN}_{F_j}, \Sigma_{F_j}, R_{F_j})$ from G'_{F_j} as $G_{F_j} = (\mathcal{TN}'_{F_j}, \mathcal{CN}'_{F_j} \cup \{F_j\}, \Sigma'_{F_j} - \{F_j\}, R'_{F_j} \cup \{F_j \to C_{F_j}\})$.

Note that each extension of G that instantiates certain variables is independent from the others, since all of them ask for subterms/subcontexts of w_{A_t}, which is ground, and does not change after substituting a variable. Hence, each $w_{A_{x_i}}$ and each $w_{C_{F_j}}$ can be defined independently from the rest using the STG G given as input. In fact, without loss of generality, we can assume that the new added nonterminals for each G_{x_i} and each G_{F_j} are disjoint. Thus, we construct G' as $G' = (\bigcup_{i=1}^{n} \mathcal{TN}_{x_i} \cup \bigcup_{j=1}^{m} \mathcal{TN}_{F_j}, \bigcup_{i=1}^{n} \mathcal{CN}_{x_i} \cup \bigcup_{j=1}^{m} \mathcal{CN}_{F_j}, \bigcap_{i=1}^{n} \Sigma_{x_i} \cap \bigcap_{j=1}^{m} \Sigma_{F_j}, \bigcup_{i=1}^{n} R_{x_i} \cup \bigcup_{j=1}^{m} R_{F_j})$.

By Lemma 3.2.23, each $\texttt{kExt}(A_t, G, \texttt{pIndex}(t, p_i))$ and each $\texttt{kExt}(A_t, G, \texttt{pIndex}(t, q_j))$ has at most $\texttt{depth}(G)$ new nonterminals. By the same Lemma, each $\texttt{depth}(\texttt{kExt}(A_t, G, \texttt{pIndex}(t, p_i)))$ and each $\texttt{depth}(\texttt{kExt}(A_t, G, \texttt{pIndex}(t, q_j)))$ is bounded by $\texttt{depth}(G)$. Thus, each G_{x_i} has at most $\texttt{depth}(G) + 1$ new nonterminals, each $\texttt{depth}(G_{x_i})$ is bounded by $\texttt{depth}(G) + 1$. By Lemma 3.2.31, each $\texttt{pCon}(\texttt{kExt}(A_t, G, \texttt{pIndex}(t, q_j)), A_{F_j}, q'_j)$ has at most $\texttt{depth}(G) * (2\texttt{depth}(G) + 3)$ new nonterminals. Thus, each G_{F_i} has at most $\texttt{depth}(G) + \texttt{depth}(G) * (2\texttt{depth}(G) + 3) + 1$ new nonterminals, that is $2\,\texttt{depth}(G)^2 + 4\,\texttt{depth}(G) + 1$.

Therefore, $|G'|$ is bounded by $|G| + n(\texttt{depth}(G) + 1) + m(2\,\texttt{depth}(G)^2 + 4\,\texttt{depth}(G) + 1)$. □

Theorem 7.2.2 *Context matching with STGs is in NP and hence it is NP-complete.*

Proof. Let G be an STG, and let A_s and A_t be term nonterminals of G. In order to verify that a given extension G' of G represents a solution for the match equation $\{A_s \doteq A_t, G\}$ it suffices to decide whether $w_{G',A_s} = w_{G',A_t}$ which can be done in polynomial time w.r.t. $|G'|$ by Theorem 3.2.16. By Lemma 7.2.1 if $\{A_s \doteq A_t, G\}$ has a solution σ then there exists an extension of polynomial size w.r.t $|G|$ representing σ. Thus there is a polynomial time verifier for the STG-context-matching problem, and hence it belongs to NP. Since context matching is already known to be NP-hard [SSS98], we obtain NP-completeness. □

For the special case of matching of strings compressed with SCFGs we obtain also NP-completeness: An instance of the matching problem for strings is a list of equations $s_1 \doteq t_1, \ldots, s_n \doteq t_n$, where s_i, t_i are strings, only s_i may contain string variables, and a solution σ may replace string variables by strings, and must solve all equations, i.e. $\sigma(s_i) = t_i$ for all i.

Corollary 7.2.3 *String-matching where left and right hand sides are compressed using an SCFG, is NP-complete.*

Proof. It is well-known that string matching is NP-hard [BKN87], and using a monadic signature, Theorem 7.2.2 shows the claim. □

Chapter 8

Towards a PTIME algorithm for One Context Unification

As seen in the previous chapter, one context unification can be solved in nondeterministic polynomial time. Moreover, it can be solved in polynomial time if the input set of equations contains an equation of the form $F(s) \doteq c(F(t))$, where F is the context variable and c is a non empty context. Since we can assume that the left hand-side of every equation in the input has an occurrence of the context variable at position lambda, we should focus in the particular case of one context unification where right-hand sides of equations do not contain the context variable. Let us argue about an even more restricted case, and consider the instance $\{F(s_1) \doteq t_1, \ldots, F(s_k) \doteq t_k\}$ of the one context unification problem where the s_is are ground terms and the t_is do not contain the context variable F. Whether there is a solution σ satisfying $\sigma(t_1) = \cdots = \sigma(t_k)$ can be checked efficiently. If it is not the case then, for any solution θ, there must be a position p such that the $\theta(t_i)$s are all equal in every position parallel to p. Note that p can be chosen to be λ, but $\mathtt{hp}(\theta(F))$ also satisfies such property. In other words, if such solution θ exists then $t_1, \ldots, t_k$ unify *up to a position.* Let us further develop this idea by first taking a step back and looking again at first-order unification.

In contrast with other variants of term unification, first-order unification is very strict: it requires the unifier to make the two given terms exactly equal, and if that is not possible, then the unification process fails. For example, while $f(x, f(a, y))$ unifies with $f(f(a, b), x)$, changing an a into a b in one of these two terms makes them nonunifiable. The terms $f(x, f(a, y))$ and $f(f(b, b), x)$ are, however, almost unifiable: if we are willing to accept disagreement at one position, the rest of the terms unify. In the terms $f(x, f(a, y))$ and $f(f(b, b), x)$, but for the position 21, the two terms unify. Hence, we call position 21 a *distinguishing position* of these two terms, since we can find a unifier, namely $\sigma = \{x \mapsto f(b,b), y \mapsto b\}$, that causes agreement on all other positions. When we apply σ to the two terms, then at position 21, we get two distinct terms: namely, a and b. For the distinguishing position 21, the pair (a,b) is the *distinguishing witness.*

Consider again an instance $\{F(s_1) \doteq t_1, \ldots, F(s_k) \doteq t_k\}$ of the one context unification problem where the s_is are ground terms and the t_is do not contain F. If a solution σ satisfies that $\sigma(t_1) = \cdots = \sigma(t_k)$ does not hold, then $\mathtt{hp}(\sigma(F))$ is a distinguishing position of $t_1, \ldots, t_k$. Thus, roughly speaking, if we can find "all" distinguishing positions of the multiequation $t_1 \doteq \cdots \doteq t_k$, then we can test each one in polynomial time to see if it is the position of the hole in $\sigma(F)$. In the affirmative case, σ can be easily obtained from $\mathtt{hp}(\sigma(F))$ in polynomial time. In this chapter, we focus on finding all distinguishing positions for an equation $t_1 \doteq t_2$ and leave the extension to multiequations and the application to one context unification for future work.

8.1 Term representation

In this chapter we deal with the explicit representation for terms. However, we will often refer to $\mathtt{subterms}(t)$ to denote the set of subterms of a term t and, for a set of terms S, we define $\mathtt{subterms}(S)$ to be the set that contains all subterms of terms in S. Moreover, given a set of term equations Δ, by $\mathtt{terms}(\Delta)$ we denote all terms occurring as right-hand side or left hand-side of some equation in Δ. We also define $\mathtt{subterms}(\Delta)$ to be $\mathtt{subterms}(\mathtt{terms}(\Delta))$. Intuitively, $\mathtt{subterms}(\Delta)$ corresponds to the set of nonterminals in an optimally compressed DAG representing the terms in Δ. In fact, all the results of this chapter can be presented using DAGs for term representation as done in Chapter 7. However, doing so would not improve the presentation.

In this section, we define the most general unifier of terms s and t, denoted $\mathtt{mgu}(s = t)$, as *any* substitution σ such that, for every unifier θ of s and t, $\sigma \leq \theta$ holds. If such substitution does not exist we say that $\mathtt{mgu}(s = t)$ is *not defined.* In our considerations in this section, we need to argue about two different terms u, v becoming equal after the application of a substitution σ, regardless of the specific form of σ. The intuition behind this consideration is that once two (sub)terms u, v are made equal due the applicaton of a substitution σ, u and v become different names for the same term. In other words, u and v are equivalent in the equivalence relation induced by σ. Hence, an application of a substitution can be defined by means of such equivalence relation on terms.

If s and t are unificable then there exists a unification relation, i.e. a homogeneous, acyclic term relation satisfying the unification axiom, that makes s and t equivalent (see Definition 2.11 and Lemma 2.12 in [BS01]). We call the least of such unification relations the closure of s and t, and denote it by $\equiv_{s,t}$.

It is easy to see that $\equiv_{s,t}$ is uniquely determined by any most general unifier of s,t (and hence $\mathtt{mgu}(s,t)$). This fact is proven in Lemma 2.12 in [BS01]. We define, for a set of terms S and terms $u, s, t \in S$, $[u]_{\mathtt{mgu}(s,t)}$ as $\{v \in S \mid u \equiv_{s,t} v\}$, i.e. the equivalence class of u in the equivalence relation induced by $\mathtt{mgu}(s,t)$ in S. For terms $s, t \in S$ and most general unifiers σ, θ of terms in S, we define the relation $\cong$ as $(s, \sigma) \cong (t, \theta)$ if (a) $[s]_\sigma \setminus \mathcal{X} = [t]_\theta \setminus \mathcal{X} \neq \emptyset$ holds, i.e $[s]_\sigma$ and $[t]_\theta$ contain the same non empty set of non variable terms, or (b) $[s]_\sigma$ and $[t]_\theta$ are sets of variables and their intersection is non empty. The definition of $\cong$ is extended to sets of terms S, T as $(S, \sigma) \cong (T, \theta)$ if there exists a bijection $b : S \to T$ such that $\forall s \in S : (s, \sigma) \cong (b(s), \theta)$.

Although the definition of $\cong$ might seem quite involved, we will only apply it on sets S, T satisfying $|S| = |T| = 2$. Moreover, in the non trivial cases, $\sigma(S)$ and $\theta(T)$ will always contain at least one variable.

The following simple lemma, which follows by induction on the size of $\mathtt{dom}(\sigma)$, is key in some of our arguments and captures the reason why the DAG representation leads to space efficient solutions in some term unification problems.

Lemma 8.1.1 *Let $S \subset \mathcal{T}(\mathcal{F}, \mathcal{X})$ be a finite set of terms and $\sigma : \mathcal{X} \mapsto S$ be a substitution satisfying that σ is acyclic as a system of equations. Then, $|\mathtt{subterms}(\sigma(S))| \leq |\mathtt{subterms}(S)|$.*

Note that, if we have a set of term equations Δ and a subset $\Gamma \subseteq \Delta$ such that $\mathtt{mgu}(\Gamma)$ is defined, then $\sigma = \mathtt{mgu}(\Gamma)$ satisfies the conditions of the previous lemma and hence $|\mathtt{subterms}(\sigma(\Delta))| \leq |\mathtt{subterms}(\Delta)|$ holds.

8.2 Problem Definition: Basis for All Distinguishing Positions

We use the notation $s|_{\overline{p}}$ to describe the context obtained from the term s by "forgetting" the subterm at position p in s.

Definition 8.2.1 ($s|_{\overline{p}}$) *Given a term s and a position p, define $s|_{\overline{p}} = s[\bullet]_p$.*

Recall that $\bullet$ is considered to be a constant and hence term unification and syntactic equality naturally extend to contexts. We can now define formally the concepts of distinguishing position and distinguishing witness mentioned in the introduction.

Definition 8.2.2 (Distinguishing position) *A position p is a distinguishing position of two terms s and t if there exists a substitution σ satisfying $\sigma(s|_{\overline{p}}) = \sigma(t|_{\overline{p}})$ and $\sigma(s) \neq \sigma(t)$. The set of all distinguishing positions of terms s, t is denoted by $\mathtt{DP}(s, t)$.*

We are interested in the set of all distinguishing positions of two given terms s, t, denoted $\mathtt{DP}(s, t)$.

Example 8.2.3 *For any term s, $\mathtt{DP}(s, s) = \emptyset$. For two terms s, t such that $s \neq t$, $\lambda \in \mathtt{DP}(s, t)$ always holds. For two distinct variables x, y, the set $\mathtt{DP}(x, y)$ is infinite because it contains every position. We also have that $\mathtt{DP}(a, b) = \mathtt{DP}(a, f(a)) = \mathtt{DP}(a, x) = \{\lambda\}$, and $\mathtt{DP}(f(a), x) = \{\lambda, 1\}$.*

As another example, let us consider the set $\mathtt{DP}(x, s)$. This set is different depending on whether the term s contains x. If s has two occurrences of x, then this set will be finite; for example, $\mathtt{DP}(x, f(x, x)) = \{\lambda\}$. If s has exactly one occurrence of x, then this set can be infinite; for example, $\mathtt{DP}(x, f(x, a)) = \{\lambda, 1, 11, \ldots\}$. Finally, if x does not occur in s, $\mathtt{DP}(x, s)$ includes all positions $\mathtt{pos}(s)$ in s; for example, $\mathtt{DP}(x, h(a)) = \mathtt{pos}(h(a))$, but it may still be infinite; for example, $\mathtt{DP}(x, h(y))$ is infinite. Finally, note that the set $\mathtt{DP}(s, t)$ is prefix-closed*; that is, if position p is in $\mathtt{DP}(s, t)$, then any prefix of p is also in $\mathtt{DP}(s, t)$.*

The following example, which was already presented in Section 2.3.2 of Chapter 2, is carefully crafted to illustrate the difficulties in computing the set of all distinguishing positions in polynomial time. This example will also help in understanding several technical details of the proof later.

Example 8.2.4 *Let s be $f(x_0, x_0)$ and let t^0 be $f(a, b)$. Then, $DP(s, t^0)$ contains $\{1, 2\}$. Let t^1 be $f(f(x_1, x_1), f(a, b))$. Then, $DP(s, t^1)$ contains $\{11, 12, 21, 22\}$. In general, define t^n recursively as $t^n = f(f(x_n, x_n), t^{n-1})$. Then, $DP(s, t^n)$ is finite and it has size exponential in n.*

Example 8.2.4 shows that if we try to output all the distinguishing positions explicitly, then we cannot hope to do better than an exponential time bound. Hence, there is a need to define some notion of equivalence on distinguishing positions. Before we do that, we first define a *distinguishing witness* for a given distinguishing position p. Intuitively speaking, for a distinguishing position p, if σ is the most-general unifier of $s|_{\overline{p}} \doteq t|_{\overline{p}}$, then the distinguishing witness for p is the ordered pair $\langle \sigma(s)|_p, \sigma(t)|_p \rangle$; and we call σ the *p-tolerant most-general unifier*, or p-mgu in short.

However, the definition is complicated by the fact that the position p may not be a valid position in the original term s and t and hence $s|_{\overline{p}} \doteq t|_{\overline{p}}$

may not be well-defined *a priori*. Hence we have to define these concepts recursively. In the following definition, we use a substitution of the form $\{x \mapsto c[x]\}$. This is a way to avoid introducing new variables, which eases the presentation.

Definition 8.2.5 (Distinguishing Witness) *Let p be a distinguishing position of $s \doteq t$. The* distinguishing witness *of p in $s \doteq t$, denoted* $\mathtt{dw}(s \doteq t, p)$, *and the p-*$\mathtt{mgu}$ *of $s \doteq t$, denoted* $\mathtt{mgu}_p(s \doteq t)$, *can be recursively defined as follows.*

- *If $p = \lambda$ then $\mathtt{dw}(s \doteq t, p) = \langle s, t\rangle$ and $\mathtt{mgu}_p(s \doteq t)$ is the identity mapping.*
- *If $p = i.p'$ and $i \in \mathbb{N}$ is a valid position in s and t then $\mathtt{dw}(s \doteq t, p) = \mathtt{dw}(\sigma(s)|_i \doteq \sigma(t)|_i, p')$ and $\mathtt{mgu}_p(s \doteq t) = (\sigma \circ \mathtt{mgu}_{p'}(\sigma(s)|_i \doteq \sigma(t)|_i))$, where $\sigma = \mathtt{mgu}(s|_{\bar{i}} \doteq t|_{\bar{i}})$.*
- *If $p = i.p'$ and $i \in \mathbb{N}$ is a valid position in t and not in s, then s is a variable x and $\mathtt{dw}(s \doteq t, p) = \mathtt{dw}(x \doteq \sigma(t)|_i, p')$, and $\mathtt{mgu}_p(s \doteq t) = (\sigma \circ \mathtt{mgu}_{p'}(x \doteq \sigma(t)|_i))$, where $\sigma = \{x \mapsto t[x]_i\}$.*
- *If $p = i.p'$ and $i \in \mathbb{N}$ is a valid position in s and not in t, then t is a variable x and $\mathtt{dw}(s \doteq t, p) = \mathtt{dw}(\sigma(s)|_i \doteq x, p')$, and $\mathtt{mgu}_p(s \doteq t) = (\sigma \circ \mathtt{mgu}_{p'}(\sigma(s)|_i \doteq x, p'))$, where $\sigma = \{x \mapsto s[x]_i\}$.*
- *$\mathtt{dw}(s \doteq t, p) = \langle s, t\rangle$ and $\mathtt{mgu}_p(s \doteq t)$ is the identity mapping in any other case.*

Note that in the last case of the previous definition s and t must be variables, due to the fact that p is a distinguishing position of $s \doteq t$.

Later in the paper, we will use the fact that, if we have parallel distinguishing positions p, q of $s \doteq t$ and a substitution θ such that $\theta \leq \mathtt{mgu}_p(s \doteq t)$ and $\theta \leq \mathtt{mgu}_q(s \doteq t)$ hold, then $\mathtt{dw}(s \doteq t, p) = \mathtt{dw}(\theta(s) \doteq \theta(t), p)$ and $\mathtt{dw}(s \doteq t, q) = \mathtt{dw}(\theta(s) \doteq \theta(t), q)$ also hold.

Example 8.2.6 *Let $s = f(x_0, x_0)$ and $t^1 = f(f(x_1, x_1), f(a, b))$ be as defined in previous example. Consider the distinguishing position $p = 21$. Then, $\mathtt{mgu}_p(s \doteq t^1)$ is the substitution $\{x_0 \mapsto f(x_1, x_1), x_1 \mapsto b\}$. Moreover, $\mathtt{mgu}_1(x \doteq h(x)) = \{x \mapsto h(x)\}$.*

Consider $s = f(x_0, x_0)$ and $t^1 = f(f(x_1, x_1), f(a, b))$ and the distinguishing position $p = 21$ as before. Then, $\mathtt{dw}(s \doteq t, p)$ is the pair $\langle b, a\rangle$. Note that the pair is ordered, and hence it is different from the pair $\langle a, b\rangle$, which *happens to be* $\mathtt{dw}(s \doteq t, 22)$*. Moreover,* $\mathtt{dw}(x \doteq h(x), 1) = \langle x, h(x)\rangle$ *and, for every position* p*,* $\mathtt{dw}(x \doteq y, p) = \langle x, y\rangle$*, where* x, y *are different variables.*

Note that the following follows from Definition 8.2.5, for any variable x*, term* t *not containing* x*, and distinguishing position* p *of* x *and* t*:* $\mathtt{dw}(x \doteq t, p) = \langle x, t|_p\rangle$ *if* $p \in \mathtt{pos}(t)$ *and* $\mathtt{dw}(x \doteq t, p) = \langle x, t|_q\rangle$ *otherwise, where* q *is the longest prefix of* p *such that* $q \in \mathtt{pos}(t)$*. On the other hand, if* t *is of the form* $C[x]_q$*, for some context* C*, then* $\mathtt{dw}(x \doteq t, p)$ *is defined only if* p *is of the form* $q^n.p'$*, where* $p' < q$*, in which case* $\mathtt{dw}(x \doteq t, p)$ *is of the form* $\langle x, C'[x]\rangle$*, where* C' *is a rotation of* C*.*

Now, we can define a notion of equivalence between distinguishing positions. We will consider two distinguishing positions equivalent if they have the same distinguishing witness (viewed as an ordered pair).

Example 8.2.7 *Consider* $s = f(x_0, x_0)$ *and* $t^1 = f(f(x_1, x_1), f(a, b))$ *as before. Note that* $\mathtt{dw}(s \doteq t^1, 12) = \mathtt{dw}(s \doteq t^1, 21) = \langle b, a\rangle$*. Similarly,* $\mathtt{dw}(s \doteq t^1, 11) = \mathtt{dw}(s \doteq t^1, 22) = \langle a, b\rangle$*. In fact, if* t^n *is defined in Example 8.2.4, then* $p_1 = 11$ *and* $p_2 = 22$ *are distinguishing positions of* s, t^n *for all* n*; and moreover,* $\mathtt{dw}(s \doteq t^n, p_1)$ *and* $\mathtt{dw}(s \doteq t^n, p_2)$ *are equal for all* n*.*

As another example, for any two different positions p, q*, it holds that* $\mathtt{dw}(x \doteq y, p) = \mathtt{dw}(x \doteq y, q)$*, where* x, y *are different variables.*

Now that we have associated a distinguishing witness with every distinguishing position, we can define a *basis* for the set of all distinguishing positions. To this end, we first define the set of all distinguishing witnesses.

Definition 8.2.8 *The* set of distinguishing witnesses *of* $s \doteq t$*, denoted* $\mathtt{SDW}(s \doteq t)$*, is defined as* $\{\mathtt{dw}(s \doteq t, p) \mid p$ *is a distinguishing position of* $s \doteq t\}$*.*

Definition 8.2.9 (Basis) *A set* P *of distinguishing positions of* $s \doteq t$ *is a* basis *if* $\{\mathtt{dw}(s \doteq t, p_1) \mid \exists p \in P : p = p1.p2\} = \mathtt{SDW}(s \doteq t)$*.*

Example 8.2.10 *Consider* $s = f(x_0, x_0)$ *and* $t^1 = f(f(x_1, x_1), f(a, b))$ *as before. Note that* $\mathtt{DP}(s \doteq t) = \{\lambda, 1, 2, 11, 12, 21, 22\}$*. However, the smaller set* $\{\lambda, 1, 2, 11, 12\}$ *is a basis. In fact, the even smaller set* $\{11, 12\}$ *is also a basis. It can be observed that, for every* n*, even though* $\mathtt{DP}(s \doteq t^n)$ *has exponentially many positions, there is a polynomial size basis for* $\mathtt{DP}(s, t^n)$*.*

As seen in the previous example, there might be basis of many cardinalities. We are interested in the ones with minimal cardinality. Such minimal basis will only contain parallel positions.

```
Global: $S = \emptyset$
Global: $D = \emptyset$
BASIS ($s, t$: terms, $p$: position)
  If ($\forall w \in D \ : \ \langle s, t\rangle \neq w$) Then
    Add $\langle s, t\rangle$ to $D$ and add $p$ to $S$
    $P = \{q \mid q \in \mathtt{DP}(s, t) \wedge |q| = 1\}$
    For $q \in P$ Do:
      $\mathtt{BASIS}(\mathtt{dw}(s \doteq t, q), p.q)$
  return $S$
```

Figure 8.1: Algorithm to compute the basis for all distinguishing positions of input equation $s \doteq t$.

8.3 Algorithm for Computing a Basis

In this section we present an algorithm to compute a (not necessarily minimal) basis for $s \doteq t$ in polynomial time. The algorithm, described in Figure 8.1, is a naive enumeration of all positions. Specifically, the procedure maintains two global variables: the variable D keeps all (non equivalent) distinguishing witnesses found so far, and the variable S keeps the corresponding distinguishing positions. The algorithm just enumerates all distinguishing positions in a depth-first manner. If it finds a position whose distinguishing witness is equivalent to a pair that has been seen before, then we do not go down further (in the depth-first search). The correctness of our algorithm to compute the basis is straightforward. The difficult part is to show that the procedure terminates in a polynomial number of steps. We can prove such polynomial time bound if we prove that the computed basis has polynomial size. This is stated in the following theorem.

Theorem 8.3.1 *Let s, t be two terms. The set* $\mathtt{SDW}(s \doteq t)$ *and a basis of* $\mathtt{SDW}(s \doteq t)$ *can be computed in polynomial time.*

The rest of this chapter is devoted to proving the above theorem. As seen in the previous section, there might be basis of exponential size. First, we argue that the *length* of any position in a minimal basis is polynomially bounded. This fact is proven in the following lemma, which states, roughly speaking, that a minimal basis cannot contain distinguishing positions of length greater-than $O(n^2)$, where n is the size of $s \doteq t$. Moreover, this fact also implies the existence of a finite basis for every equation $s \doteq t$, and hence, our notion of a minimal basis is well defined.

Lemma 8.3.2 *Let s, t be terms over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and let p be a distinguishing positions of $s \doteq t$ such that $|p| > |\mathtt{subterms}(s \doteq t)|$. Then, there exists a position $q < p$ such that $\mathtt{dw}(s \doteq t, p) = \mathtt{dw}(s \doteq t, q)$.*

Proof. The lemma can be proven by induction on the number ns of *different* subterms of s and t. In the base case (ns = 2), s and t are variables and the lemma holds. Otherwise, let $p = i.p'$, with $i \in \mathbb{N}$. If s or t is a variable, the lemma follows from Definition 8.2.5. Otherwise, note that $|p'| > |\mathtt{subterms}(\mathtt{dw}(s \doteq t, i))| < ns$ holds by Definition 8.2.5, and the lemma follows from the induction hypothesis. □

Now, we only need to prove a polynomial upper bound on the size of a minimal basis, where *size* of a basis is its cardinality. First of all, to simplify our arguments, we introduce a notion of equivalence between distinguishing witnesses that is weaker than equality. In this notion, the order of the terms in a distinguishing witness is irrelevant.

Definition 8.3.3 *Let $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$ be pairs of different terms. We say they are* equivalent*, denoted $\langle s_1, t_1\rangle \equiv \langle s_2, t_2\rangle$, if $\{s_1, t_1\} = \{s_2, t_2\}$.*

We define $\mathtt{SDW}_w$ and basis with respect to $\equiv$, which we call set of weakly distinguishing witnesses and weak basis, respectively, analogously to $\mathtt{SDW}$ and basis in Definitions 8.2.8 and 8.2.9. Note that, if we have a weak basis B for $s \doteq t$, then there must be a basis B' of size at most $2|B|$. Due to this observation, henceforth we will only use the weak notion of a basis. Hence, we consider distinguishing witnesses to be *sets* of cardinality 2, and thus the notion of equivalence of Definition 8.3.3 just reduces to equality. Therefore, we write $\mathtt{SDW}$ for $\mathtt{SDW}_w$ and basis for weak basis in the rest of this chapter.

Before we prove that the cardinality of the minimal basis is polynomially bounded, let us generalize from $s \doteq t$ to a set $\{s_1 \doteq t_1, \ldots, s_k \doteq t_k\}$. The reason for this generalization is that we get such a set of equations as soon as we apply the decomposition rule (from syntactic unification) to a single initial equation $f(s_1, \ldots, s_k) \doteq f(t_1, \ldots, t_k)$. In this new setting, a basis is a function mapping equations to sets of distinguishing positions defined as follows.

Definition 8.3.4 *Let $\Delta = \{e_1, \ldots, e_n\}$ be a set of term equations such that $\sigma_i = \mathtt{mgu}(\Delta \setminus \{e_i\})$, for every $i \in \{1, \ldots, n\}$, is defined. A* distinguishing function *of Δ is a mapping $B : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ such that $B(e_i)$ is a set of distinguishing positions of $\sigma_i(e_i)$, for every $i \in \{1, \ldots, n\}$.*

According to the previous definition, a function mapping every equation of Δ to the empty set is a distinguishing function of Δ. However, it is obviously not a basis, since, roughly speaking, it does not span $\mathtt{SDW}$. Hence, we define basis as follows.

Definition 8.3.5 *Let $\Delta = \{e_1, \ldots, e_n\}$ be a set of term equations such that $\sigma_i = \mathtt{mgu}(\Delta \setminus \{e_i\})$, for every $i \in \{1, \ldots, n\}$, is defined.*

The set of distinguishing witnesses *of Δ, denoted $\mathtt{SDW}(\Delta)$, is defined as $\bigcup_{i=1}^{n}\{\mathtt{dw}(\sigma_i(e_i), p) \mid p$ is a distinguishing position of $\sigma_i(e_i)\}$.*

Let $B : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ be a distinguishing function of Δ. B is also a basis *of Δ if $\bigcup_{i=1}^{n}\{\mathtt{dw}(\sigma_i(e_i), p_1) \mid p_1.p_2 \in B(e_i)\} = \mathtt{SDW}(\Delta)$.*

The following definition of size of a basis is motivated by the fact stated in Lemma 8.3.2: distinguishing positions in a basis must have polynomial length. Hence, we define the size of a basis just as the total number of positions in its image.

Definition 8.3.6 (size of a basis) *Let $\Delta = \{e_1, \ldots, e_n\}$ be a set of term equations and let $B : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ be a basis of Δ. Then, the size of B, denoted $\mathtt{size}(B)$, is defined as $\sum_{i=1}^{n} |B(e_i)|$.*

As mentioned above, we are interested in basis of minimal size. The following definition is motivated by the fact that if two different distinguishing positions p_1, p_2 have equivalent distinguishing witnesses in Δ, then any extensions $p_1.q, p_2.q$ of p_1 and p_2 will also be equivalent (equal) and hence both of them will not be in a minimal basis. Similarly, if B is a minimal basis of Δ, then the positions in $B(e)$ must be pairwise parallel, for every equation e in Δ.

Definition 8.3.7 *Let $\Delta = \{e_1, \ldots, e_n\}$ be a set of term equations such that $\sigma_i = \mathtt{mgu}(\Delta \setminus \{e_i\})$, for every $i \in \{1, \ldots, n\}$, is defined. Let $B : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ be a distinguishing function of Δ. Then, we say that B is* redundant *for Δ if $\exists i, j, p_1.p_2 \in B(e_i), q_1.q_2 \in B(e_j) : (i \neq j \vee p_1 \neq p_2) \wedge \mathtt{dw}(\sigma_i(e_i), p_1) = \mathtt{dw}(\sigma_j(e_j), q_1)$ holds.*

8.4 Polynomial Bound for Size of Basis

Given a set of equations Δ and a minimal basis B of Δ, our goal is to polynomially bound $\mathtt{size}(B)$. We count the number of distinct distinguishing positions (witnesses) by reasoning about three cases separately. In Section 8.4.1 we count witnesses of the form $\{x, s\}$ (where x is a variable), in Section 8.4.2 we count witnesses of the form $\{s, C[s]\}$, and in Section 8.4.3 we count all remaining witnesses.

8.4.1 Counting distinguishing witnesses of form $\{x, s\}$

Let us distinguish two kinds of distinguishing positions depending of the form of their distinguishing witnesses.

Definition 8.4.1 *Let s, t be terms and let p be a distinguishing position of s and t. Let $\{u, v\} = \mathtt{dw}(s \doteq t, p)$. We say that p has a* λ-defined witness *in $s \doteq t$ if neither u nor v are variables.*

The following two lemmas follow from Definition 8.2.5.

Lemma 8.4.2 *Let s, t be terms and let p be a distinguishing position of s, t such that p has a λ-defined witness in $s \doteq t$. Then, $\forall q \leq p$: q has a λ-defined witness in $s \doteq t$.*

Lemma 8.4.3 *Let s, t be terms and let p be a distinguishing position of s, t such that p does not have a λ-defined witness in $s \doteq t$. Then, $\forall q : p \leq q, q$ does not have a λ-defined witness in $s \doteq t$.*

Let us modify a basis B of Δ to obtain a nonredundant distinguishing function $B' : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ by removing positions that do not have λ-defined witnesses, and replacing them by their prefix position that have such a witness. We will polynomially bound the size of B with respect to the size of the obtained B'. After that, it remains to prove a polynomial bound on $\mathtt{size}(B')$, using the fact that all the positions in its image have a λ-defined witness and that B' is non redundant. The following two definitions state the transformation mentioned above formally.

Definition 8.4.4 *Let s, t be terms and let p be a distinguishing position of s, t. Let p_1 be the longest prefix of p such that p has a λ-defined witness. We define $\mathtt{trim}(p, s \doteq t)$ as p_1 if p_1 exists and $\perp$, otherwise.*

Definition 8.4.5 *Let $\Delta = \{e_1, \ldots, e_n\}$ be a set of equations, let B be a distinguishing function of Δ, let σ_i be $\mathtt{mgu}(\Delta \setminus \{e_i\})$, and let S_i be $\{\mathtt{trim}(p, \sigma_i(e_i)) \mid p \in B(e_i)\}$, for each $i \in \{1, \ldots, n\}$.*

We define $\mathtt{trim}(B, \Delta) : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ as $\mathtt{trim}(B, \Delta)(e_i) = \{p \in S_i \mid \forall q \in S_i : p \not< q\}$ if $S_i \neq \{\perp\}$ and $\mathtt{trim}(B, \Delta)(e_i) = \emptyset$, otherwise, for each $i \in \{1, \ldots, n\}$.

Let us remark that, for each of the sets S_i in the previous definition, $\perp \in S_i \Leftrightarrow S_i = \{\perp\}$ follows from Lemma 8.4.3. Moreover, note that if B is a minimal basis of Δ, $\mathtt{trim}(B, \Delta)$ might no longer be a basis of Δ. However, in that case, $\mathtt{trim}(B, \Delta)$ will be non redundant as stated in the following lemma, which directly follows from Lemmas 8.4.2 and 8.4.3, and the definition of $\texttt{trim}$, and specifically the fact that we guarantee that the positions in $\texttt{trim}(B,\Delta)(e)$, for each equation $e \in \Delta$, are pairwise parallel.

Lemma 8.4.6 *Let $\Delta = \{e_1, \dots, e_n\}$ be a set of term equations such that $\sigma_i = \texttt{mgu}(\Delta \setminus \{e_i\})$, for every $i \in \{1, \dots, n\}$, is defined. Let B be a basis of Δ minimal in $\texttt{size}$. Then, $\texttt{trim}(B,\Delta)$ is a distinguishing function of Δ and it is not redundant. Moreover, $\forall i \in \{1, \dots, n\}, p \in \texttt{trim}(B,\Delta)(\sigma_i(e_i)), t \in \texttt{dw}(\sigma_i(e_i), p) : t$ is not a variable.*

As mentioned above, we prove a polynomial bound on $\texttt{size}(B)$ with respect to $\texttt{size}(\texttt{trim}(B,\Delta))$. We will use three technical auxiliary lemmas. The first of them directly follows from the definition of $\texttt{dw}$ (Definition 8.2.5).

Lemma 8.4.7 *Let x be a variable and let t be a term. Then, $|\texttt{SDW}(x \doteq t)| \leq |\texttt{subterms}(t)|$.*

The next lemma states that computing $\texttt{dw}$ for distinguishing positions that have a λ-defined witness does not increase the number of different subterms.

Lemma 8.4.8 *Let s, t be terms. Then, $|\texttt{subterms}(u) \cup \texttt{subterms}(v)| \leq |\texttt{subterms}(s) \cup \texttt{subterms}(t)|$ holds for every position $p \in \texttt{DP}(s,t)$ such that p has a λ-defined witness in $s \doteq t$, where $\{u, v\} = \texttt{dw}(s \doteq t, p)$.*

Proof. The lemma can be proven by induction on $|p|$ using Lemma 8.1.1 and distinguishing cases according to the definition of $\texttt{dw}$ (Definition 8.2.5). Note that cases $3, 4$ in Definition 8.2.5 will not be applied by the conditions of the lemma and Lemma 8.4.2. □

Lemma 8.4.9 *Let Δ be a set of term equations over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and let $\{x_1 \doteq t_1, \dots, x_m \doteq t_m\}$ be a subset of Δ, where $x_1, \dots, x_m \in \mathcal{X}$, satisfying that*

1. *$\forall i \in \{1, \dots, m\} : \texttt{mgu}(\Delta \setminus \{x_i \doteq t_i\})$ is defined,*

2. *$\forall i \in \{1, \dots, m\} : (\texttt{mgu}(\Delta \setminus \{x_i \doteq t_i\}))(\texttt{terms}((x_i \doteq t_i))$ contains a variable, and*

3. *$\forall i, j \in \{1, \dots, m\} : i \neq j \Rightarrow (\{x_i, t_i\}, \texttt{mgu}(\Delta \setminus \{x_i \doteq t_i\})) \not\cong (\{x_j, t_j\}, \texttt{mgu}(\Delta \setminus \{x_j \doteq t_j\}))$.*

Then, $x_1, \dots, x_m$ are pairwise different.

Proof. Let Γ be $\mathtt{mgu}(\Delta \setminus \{x_1 \doteq t_1, \ldots, x_m \doteq t_m\})(\{x_1 \doteq t_1, \ldots, x_m \doteq t_m\})$ and note that $\forall i \in \{1, \ldots, m\} : \mathtt{mgu}(\Gamma \setminus \{x_i \doteq t_i\}) = \mathtt{mgu}(\Delta \setminus \{x_i \doteq t_i\})$ holds. Moreover, Γ must be of the form $\{y_1 \doteq s_1, \ldots, y_m \doteq s_m\}$, for some variables $y_1, \ldots, y_m$. To argue by contradiction, assume also that $y_1 = y_2$ and let θ be $\mathtt{mgu}(\Gamma \setminus (\{y_1 \doteq s_1, y_2 \doteq s_2\}))$. Note that $y_1\theta$ must be a variable by the assumptions of the lemma, and that $\theta \leq \mathtt{mgu}(\Gamma \setminus (y_1 \doteq s_1))$ and $\theta \leq \mathtt{mgu}(\Gamma \setminus (y_2 \doteq s_2))$ hold. Hence, $\{y_1, s_1\}(\mathtt{mgu}(\Gamma \setminus (y_2 \doteq s_2))) = \{y_2, s_2\}(\mathtt{mgu}(\Gamma \setminus (y_1 \doteq s_1)))$, which contradicts the third assumption of the lemma. □

We are now ready to prove the promised bound.

Lemma 8.4.10 *Let Δ be a set of equations over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and let B be a minimal basis of Δ. Then,* $\mathtt{size}(B)| \leq (1 + \mathtt{size}(\mathtt{trim}(B, \Delta)))|\mathcal{X}||\mathtt{subterms}(\Delta)|$.

Proof. Let us partition Δ as $\Delta = \Delta_1 \cup \Delta_2 \cup \Delta_3$, with the Δ_is defined as follows. Let Δ_1 be the biggest subset of Δ such that $B(e) \neq \emptyset$ and $\mathtt{trim}(B, \Delta)(e) = \emptyset$, for every $e \in \Delta_1$. Let Δ_2 be the biggest subset of Δ such that $\mathtt{trim}(B, \Delta)(e) \neq \emptyset$, for every $e \in \Delta_2$, and let Δ_3 be $\Delta \setminus (\Delta_1 \cup \Delta_2)$, i.e. equations in Δ whose image in B is the empty set.

We first prove that $|\mathtt{size}(B|_{\Delta_1})| \leq |\mathcal{X}||\mathtt{subterms}(\Delta)|$. Let Δ_1 be $\{e_1, \ldots, e_n\}$, more explicitly written. Let σ_i be $\mathtt{mgu}(\Delta \setminus \{e_i\})$, for every $i \in \{1, \ldots, n\}$. Note that $|\mathtt{subterms}(\sigma_i(e_i))| \leq |\mathtt{subterms}(\Delta)|$ holds by Lemma 8.1.1 and, by Lemma 8.4.3 and the definition of $\mathtt{trim}$, $\mathtt{terms}(\sigma_i(e_i)) = \mathtt{terms}(\mathtt{dw}(\sigma_i(e_i), \lambda))$ contains a variable, for every $i \in \{1, \ldots, n\}$. Hence, by Lemma 8.4.9 and the minimality of B, it follows that $|\Delta_1| \leq |\mathcal{X}|$ and $|\mathtt{size}(B|_{\Delta_1})| \leq |\mathcal{X}||\mathtt{subterms}(\Delta)|$.

Now, consider the equations in Δ_2 and let us write Δ_2 more explicitly as $\{e_1, \ldots, e_m\}$. Let θ_i be $\mathtt{mgu}(\Delta \setminus \{e_i\})$, for every $i \in \{1, \ldots, m\}$. Let e_j be any equation in $\{e_1, \ldots, e_m\}$ and let p be a position in $\mathtt{trim}(B, \Delta)(e_j)$. We show that there exist at most $|\mathcal{X}||\mathtt{subterms}(\Delta)|$ positions in $B(e_j)$ that are extensions of some prefix of p. Let $p_1.k_1, \ldots, p_l.k_l$ be all positions such that, for every $i \in \{1, \ldots, l\}$, $p_i \leq p$, $k_i \in \mathbb{N}$, and $p_i.k_i$ is the prefix of some position in $B(e_j)$. By Lemma 8.1.1, $|\mathtt{subterms}(\theta_j(e_j))| \leq |\mathtt{subterms}(\Delta)|$ holds. Note that, it follows from Lemmas 8.4.7 and 8.4.8 that, to prove that there exist at most $|\mathcal{X}||\mathtt{subterms}(\Delta)|$ positions in $B(e_j)$ that are extensions of some prefix of p, it suffices to show that $l \leq |\mathcal{X}|$. Hence, to conclude the proof, we prove that $l \leq nv$, where nv is the number of variables in $\mathtt{subterms}(u \doteq v)$, with $(u \doteq v) = \theta_j(e_j)$. We use induction on nv. Note that, if $nv = 0$, then p must be λ and $l = nv = 0$. Hence, for the inductive step, assume that $nv > 0$. Let q be the shortest position among $\{p_1, \ldots, p_l\}$ and let $\{q.n_1, \ldots, q.n_c\} = \{p_i.k_i \in \{p_1.k_1, \ldots, p_l.k_l\} \mid p_i = q\}$. Also, let $\langle u_q, v_q\rangle = \mathtt{dw}(u \doteq v, q)$ and note that, by definition of q, u_q and v_q are not variables, and hence u_q and v_q are of the form $f(u_{q,1},\ldots,u_{q,\mathtt{arity}(f)})$ and $f(v_{q,1},\ldots,v_{q,\mathtt{arity}(f)})$. Moreover, we can assume without loss of generality, that u_{q,n_i} is a variable, for every $i \in \{1,\ldots,c\}$. We also define the equation $(u' \doteq v')$ as $u' = \alpha(u_{q,b})$ and $v' = \alpha(v_{q,b})$, with $b \in \mathbb{N}$ being such that $q.b$ is a prefix of p and $\alpha = \mathtt{mgu}(\bigcup_{i \neq b}(u_{q,i} \doteq v_{q,i}))$. By Lemma 8.4.9, the variables u_{q,n_i}, such that $i \in \{1,\ldots,c\}$, are pairwise disjoint and thus $|\mathtt{dom}(\alpha)| \geq c$. It follows that the number of variables in $\mathtt{subterms}(u' \doteq v')$ is smaller than $nv - c$. By induction hypothesis, $l \leq nv - c + c \leq nv$, and we are done.

Altogether implies that $\mathtt{size}(B) \leq \mathtt{size}(B|_{\Delta_1}) + \mathtt{size}(B|_{\Delta_2}) + \mathtt{size}(B|_{\Delta_3}) = |\mathcal{X}||\mathtt{subterms}(\Delta)| + \mathtt{size}(\mathtt{trim}(B,\Delta))|\mathcal{X}||\mathtt{subterms}(\Delta)| + 0$, which concludes the proof. □

Thanks to the previous lemma and Lemma 8.4.6, to prove Theorem 8.3.1, it suffices to show a bound on the $\mathtt{size}$ of a nonredundant distinguishing function B of a set of equations Δ. In the following section we state simple assumptions on the form of B and Δ with the goal of easing the presentation.

Simplifying assumptions

Consider a set of equations $\Delta = \{e_1,\ldots,e_n\}$ and a nonredundant distinguishing function B of Δ. Let $\sigma_i = \mathtt{mgu}(\Delta \setminus \{e_i\})$, for each $i \in \{1,\ldots,m\}$. Note that, by Definition 8.3.5, it must be the case that σ_i is defined for all $i \in \{1,\ldots,m\}$. First, we assume that

$$\forall i \in \{1,\ldots,n\} : B(e_i) \neq \emptyset$$

Note that, if there is an equation $e_i \in \Delta$ such that $B(e_i) = \emptyset$, then we can force our assumption by defining a new set Δ' as $\mathtt{mgu}(e_i)(\Delta \setminus \{e_i\})$, and a new distinguishing function $B' = (B \setminus \{(e_i, B(e_i))\})$. To see that this transformation is correct, note that B' is a distinguishing function of Δ' and that $\mathtt{SDW}_w(\Delta') = \mathtt{SDW}_w(\Delta)$. Let us remark, although it is not necessary for our argument, that the transformation can be performed in polynomial time and it can be applied at most $|\Delta|$ times. Moreover, B' is not redundant for Δ', since B is not redundant for Δ, and $|\mathtt{subterms}(\Delta')| \leq |\mathtt{subterms}(\Delta)|$.

Let $e_i = (s_i \doteq t_i)$, for $i \in \{1,\ldots,n\}$. Due to the previous assumption and the nonredundancy of B, the following holds:

$$\forall i,j \in \{1,\ldots,m\} : i \neq j \Rightarrow \sigma_i(\{s_i,t_i\}) \neq \sigma_j(\{s_j,t_j\})$$

The previous observation will be a key component of our argument. Finally, note that the previous assumptions trivially hold in the case when Δ contains a single equation $s \doteq t$.

During our argument, we will transform the initial set of equations Δ and a nonredundant distinguishing function B of Δ by unifying some of the equations in Δ and restricting the domain of B to obtain a new Δ' and B' such that B' is a nonredundant distinguishing function of Δ'. The following Lemma states the correctness of such operation.

Lemma 8.4.11 *Let $\Delta = \{e_1, \dots, e_n\}$ be a set of equations and let B be a nonredundant distinguishing function of Δ. Let Δ' be $\mathtt{mgu}(e_1)(\Delta \setminus \{e_1\})$ and let B' be defined as $B'(\mathtt{mgu}(e_1)(e)) = B(e)$, for every $e \in \Delta \setminus \{e_1\}$. Then, B' is a nonredundant distinguishing function of Δ', $\mathtt{size}(B) \leq |B(e_1)| + \mathtt{size}(B')$, and $|\mathtt{subterms}(\Delta')| \leq |\mathtt{subterms}(\Delta)|$.*

Proof. If $|\Delta| \leq 1$ the lemma holds trivially. Otherwise, let σ_i be $\mathtt{mgu}(\Delta \setminus \{e_i\})$, which is defined by the definition of distinguishing function, for every $i \in \{2, \dots, n\}$. It suffices to note that $\forall i \in \{2, \dots, n\}, p \in B(e_i) : \mathtt{mgu}(e_1) \leq p\text{-}\mathtt{mgu}(\sigma_i(e_i))$ and hence $\forall i \in \{2, \dots, n\}, p \in B(e_i) : \mathtt{dw}(\sigma_i(e_i), p) = \mathtt{dw}((\mathtt{mgu}(e_1) \circ \mathtt{mgu}(\Delta' \setminus \{e_i\}))(e_i), p)$ holds. Thus, B' is a nonredundant distinguishing function of Δ', the fact that $\mathtt{size}(B) \leq |B(e_1)| + \mathtt{size}(B')$ follows from the definition of $\mathtt{size}$, and $|\mathtt{subterms}(\Delta')| \leq |\mathtt{subterms}(\Delta)|$ follows from Lemma 8.1.1. □

8.4.2 Counting distinguishing witnesses of form $\{s, C[s]\}$

Recall that syntactic unification fails if we try to unify s with $C[s]$, for nontrivial context C (usually called “occur-check”). We separately identify the set of equations that can lead to such “occur-check” violations, because they can have very few distinguishing positions (formalized in Lemma 8.4.15). A *stable* set of equations is defined so that it does not (immediately) lead to “occur-check” failures.

Definition 8.4.12 *Let $\Delta = \{s_1 \doteq t_1, \dots, s_m \doteq t_m\}$ be a non empty set of term equations. We call Δ* stable *if, for every $s \doteq t \in \Delta$, the following holds:*

1. *$\Delta \setminus \{s \doteq t\}$ unifies,*

2. *$\mathtt{mgu}(\Delta \setminus \{s \doteq t\})(s) \not\leq \mathtt{mgu}(\Delta \setminus \{s \doteq t\})(t)$, and*

3. *$\mathtt{mgu}(\Delta \setminus \{s \doteq t\})(t) \not\leq \mathtt{mgu}(\Delta \setminus \{s \doteq t\})(s)$.*

Example 8.4.13 *$\Delta_1 = \{f(h(a)) \doteq g(h(h(a)))\}$ and $\Delta_2 = \{x \doteq f(y, y), x \doteq f(s, x_1), x \doteq f(x_2, t)\}$ are stable sets of equations, where x, y, x_1, x_2 are variables and s, t are terms.*

Lemma 8.4.14 *Let Δ be a stable set of equations. Then, there exists a non empty subset $\Delta' \subseteq \Delta$ such that*

1. *$\forall t_1, t_2 \in \texttt{terms}(\Delta') : t_1 \not< t_2$, and*

2. *$\forall t_1 \in \texttt{terms}(\Delta'), t_2 \in \texttt{terms}(\Delta \setminus \Delta') : t_1 \not\leq t_2$.*

Proof. We argue by contradiction. Hence, assume that Δ does not satisfy the condition of the lemma. Consider the equations in Δ to be unordered and let $\Gamma_1 \subset \Delta$ be any subset of the form $\{s_1 \doteq t_1, \ldots, s_1 \doteq t_n\}$ satisfying that $s_1 \notin \texttt{terms}(\Delta \setminus \Gamma_1)$. Note that Γ_1 must exist and can be chosen to be non empty since Δ is non empty by the definition of stable set of equations. Moreover, $\forall t \in \{t_1, \ldots, t_n\} : s_1 \not\leq t$ holds by the stability of Δ and thus Γ_1 does not violate the first condition of the lemma. Hence, Γ_1 must violate the second condition of the lemma which implies that there must be a term $s_2 \in \texttt{terms}(\Delta \setminus \Gamma_1)$ such that $\exists t \in \{s_1, t_1, \ldots, t_n\} : t \leq s_2$ holds. Then, we define the subset Γ_2 of Δ of the form $\{s_2 \doteq t'_1, \ldots, s_2 \doteq t'_{n'}\}$ analogously to Γ_1. Iterating this argument we can obtain sets $\Gamma_1, \Gamma_2, \Gamma_3, \ldots$ such that every Γ_i in the sequence satisfies that it is a non empty subset of Δ of the form $\{s_i \doteq t^i_1, \ldots, s_i \doteq t^i_{n_i}\}$, $s_i \notin \texttt{terms}(\Delta \setminus \Gamma_i)$, $\forall t \in \{t^i_1, \ldots, t^i_{n_i}\} : s_i \not\leq t$, that $s_{i+1} \in \texttt{terms}(\Delta \setminus \Gamma_i)$, and that there exists a term $u_i \in \{s_i, t^i_1, \ldots, t^i_{n_i}\}$ such that $u_i \leq s_{i+1}$ holds. Note that, there must be indexes j, k, with $j < k$, such that $s_k \in \texttt{subterms}(\Gamma_j)$. It follows that Δ contradicts condition (2) or (3) of the definition of stable, a contradiction. □

Lemma 8.4.15 *Let $\mathcal{V}$ be a set of variables. Let Δ be a non empty set of equations over $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Let $B : \Delta \mapsto \mathcal{P}(\texttt{pos}(\mathcal{F}))$ be a nonredundant distinguishing function of Δ. If Δ is not stable then there exists $e \in \Delta$ such that $|B(e)| \leq 1$.*

Proof. Note that condition (1) in Definition 8.4.12 must hold, since otherwise Δ does not satisfy the conditions in Definition 8.3.5 and thus B would not be a distinguishing function of Δ. Hence, first assume that there is an equation $(s \doteq t) \in \Delta$, such that $\texttt{mgu}(\Delta \setminus \{s \doteq t\})(s) = \texttt{mgu}(\Delta \setminus \{s \doteq t\})(t)$ and note that this case is also not possible since, in this case $B(s \doteq t)$ would be empty, contradicting our assumptions from the previous section. Hence, without loss of generality, it holds that $\texttt{mgu}(\Delta \setminus \{s \doteq t\})(s) \prec \texttt{mgu}(\Delta \setminus \{s \doteq t\})(t)$. Let $u \doteq v$ be $\texttt{mgu}(\Delta \setminus \{s \doteq t\})(s \doteq t)$ and note that there exists some position $p \neq \lambda$ such that $v|_p = u$. We now prove that $|B(s \doteq t)| \leq 1$. Since B is nonredundant, it suffices to show that $u \doteq v$ does not have two parallel distinguishing positions. To argue by contradiction, assume that $u \doteq v$ has two parallel distinguishing positions q_1, q_2. One of them, say q_1, must be parallel with p. Let σ be $\mathtt{mgu}_{q_1}(u \doteq v)$. Since σ is defined, $\sigma(u|_p) = \sigma(v|_p)$ must hold but, by the conditions of the lemma $\sigma(u|_p)$ is a proper subterm of $\sigma(v|_p)$, a contradiction. □

8.4.3 Counting the remaining distinguishing witnesses

We are now ready to prove the desired bound. As a first ingredient in the last part of the proof, let us define an operation to decompose a given set of equations Δ and its corresponding distinguishing function B. This operation is similar to decompose rule in syntactic unification. It will be a key component of our final argument, since, using Lemma 8.4.14, we will make sure that the number of different subterms in Δ decreases after applying this decomposition while the size of the distinguishing basis after the decomposition is preserved.

Definition 8.4.16 (Decompose) *Let Δ be a set of equations and let $\Gamma \subseteq \Delta$ be a subset satisfying $\forall t \in \mathtt{terms}(\Gamma) : \mathtt{h}(t) > 0$, and $\forall (s \doteq t) \in \Gamma : \mathtt{root}(s) = \mathtt{root}(t)$. Let $B : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ be nonredundant distinguishing function of Δ such that $\forall e \in \Gamma : B(e)$ is non empty and does not contain λ.*

For each equation $e = (f(s_1, \ldots, s_m) \doteq f(t_1, \ldots, t_m))$ in Γ, we define $\mathtt{decomposeEq}(e)$ as the set $\{s_i \doteq t_i \mid i \in \{1, \ldots, m\}\}$, and $\mathtt{decomposeFun}(e)$ as as the set of pairs $\{\langle s_i \doteq t_i, \{p \mid i.p \in B(e)\}\rangle \mid i \in \{1, \ldots, m\}\}$.

We define $\mathtt{decompose}(\Delta, \Gamma, B)$ to be the pair $\left(\bigcup_{e \in \Gamma}(\mathtt{decomposeEq}(e)) \cup (\Delta \setminus \Gamma), \bigcup_{e \in \Gamma}(\mathtt{decomposeFun}(e)) \cup \{\langle e', B(e')\rangle \mid e' \in (\Delta \setminus \Gamma)\}\right)$.

The following lemma states that $\mathtt{decompose}$ preserves $\mathtt{size}$ and non redundancy of distinguishing functions.

Lemma 8.4.17 *Let Δ be a set of equations and let $B : \Delta \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ be nonredundant distinguishing function of Δ. Let Γ be any subset of Δ satisfying the conditions of Definition 8.4.16. and let (Δ', B') be the $\mathtt{decompose}(\Delta, \Gamma, B)$.*

Then, B' is nonredundant distinguishing function of Δ' and $\mathtt{size}(B') = \mathtt{size}(B)$.

The following lemma states some conditions under which the size of a set of equations Δ is bounded by the number of non variable terms occurring in the equations of Δ. This will be useful in our final argument, since, to always be able to apply the $\mathtt{decompose}$ operation, we may transform the initial set of equations Δ by applying substitutions that add fresh variables.

Lemma 8.4.18 *Let $\mathcal{V}$ be a set of first-order variables and let $\Delta = \{s_1 \doteq t_1, \ldots, s_m \doteq t_m\}$ be a set of term equations over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ satisfying that*

1. $\forall i \in \{1,\ldots,m\} : \texttt{mgu}(\Delta \setminus \{s_i \doteq t_i\})(s_i) \neq \texttt{mgu}(\Delta \setminus \{s_i \doteq t_i\})(t_i)$*,*

2. $\forall i \in \{1,\ldots,m\}, u \in \{s_i,t_i\} : \texttt{mgu}(\Delta \setminus \{s_i \doteq t_i\})(u) \notin \mathcal{V}$*, and*

3. $\forall i,j \in \{1,\ldots,m\} : i \neq j \Rightarrow (\{s_i,t_i\}, \texttt{mgu}(\Delta \setminus \{s_i \doteq t_i\})) \not\cong (\{s_j,t_j\}, \texttt{mgu}(\Delta \setminus \{s_j \doteq t_j\}))$*.*

Then, $m \leq 2|\texttt{terms}(\Delta) \setminus \mathcal{V}|$*.*

Proof. If $\Delta = \emptyset$ then the lemma holds trivially. Hence, assume that $\Delta \neq \emptyset$ and consider the graph $G(V = \{s_1,t_1,\ldots,s_m,t_m\}, E = \{(s,t) \mid s \doteq t \in \Delta\})$. G must be acyclic since otherwise there exists $(s \doteq t) \in \Delta$ such that $\texttt{mgu}(\Delta \setminus \{s \doteq t\})(s) = \texttt{mgu}(\Delta \setminus \{s \doteq t\})(t)$, contradicting the first condition of the lemma. Note that, by the second condition of the lemma, every leaf of G must be a non variable term. Moreover, by the third condition of the lemma, every node in $V \cap \mathcal{V}$ must have degree at least 3. Assume, without loss of generality that, that G has only one connected component. Pick any node of G to be its root and let $T(n)$ be the number of nodes of a subtree of G with n nodes that are not variables. Note that $T(1) = 0$, $T(2) = 1$, and $T(n) \leq T(n_1) + \ldots + T(n_k) + 1$, where $k \geq 2$, $\sum_{j=1}^{k}(n_j) \leq n$, and $\forall i \in \{1,\ldots,k\} : n_i > 0$. The fact that $T(n) \leq 2n-1$ follows by induction on n. Hence, $|V| \leq 2n-1$ and $m = |E| \leq 2n-2 \leq 2n$, which concludes the proof. □

Lemma 8.4.19 *Let* $\mathcal{X}, \mathcal{Y}$ *be disjoint sets of variables. Let* $\Delta = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ *be a set of equations over* $\mathcal{T}(\mathcal{F},\mathcal{X}) \cup \mathcal{Y}$*. Let* $B : \Delta \mapsto \mathcal{P}(\texttt{pos}(\mathcal{F}))$ *be a nonredundant distinguishing function of* Δ *satisfying that* $\forall i \in \{1,\ldots,n\}, u \in \{s_i,t_i\} : \texttt{mgu}(\Delta \setminus \{s_i \doteq t_i\})(u)$ *is not a variable. Then,* $\texttt{size}(B) \leq |\texttt{subterms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})|^2$*.*

Proof. First of all, note that Δ satisfies the conditions of Lemma 8.4.18: condition (1) follows from the fact that B is a distinguishing function of Δ and the fact that $\forall e \in \Delta : B(e) \neq \emptyset$, which is one the assumptions stated in Section 8.4.1, condition (2) follows from the conditions of the lemma, and condition (3) follows from the non redundancy of B and, again, the fact that $B(e)$ is non empty for every equation in Δ. Hence, we can apply Lemma 8.4.18 and conclude that $|\Delta| \leq 2|\texttt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})|$.

We use induction on $m = |\texttt{subterms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})|$. By the conditions of the lemma, m must be at least 2. In that case, it follows from the conditions of the lemma that $\texttt{size}(B) = 1$, and the lemma holds in this case. For the inductive step, assume that $m > 2$. First, note that if Δ is not stable then, by Lemma 8.4.15, there exists an equation $e \in \Delta$

such that $B(e)$ only contains one position. In that case, we define Δ_1 as $\mathtt{mgu}(e)(\Delta \setminus \{e\})$ and $B_1 : \Delta_1 \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ as $B(\mathtt{mgu}(e)(e')) = B(e')$, for every $e' \in (\Delta \setminus \{e\})$. Note that $\mathtt{size}(B) = 1 + \mathtt{size}(B_1)$ and B_1 is a nonredundant distinguishing function of Δ_1, by Lemma 8.4.11. Note that we can repeat this construction to obtain sets of equations and nonredundant distinguishing functions $\Delta_1, B_1, \ldots, \Delta_l, B_l$ such that $\mathtt{size}(B) \leq l + \mathtt{size}(B_l)$ and B_l is a nonredundant distinguishing function of Δ_l. Clearly, if $\Delta_l = \emptyset$, then $\mathtt{size}(B_l) = 0$, $\mathtt{size}(B) \leq 2|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})| \leq m^2$, and the lemma holds in this case. Otherwise, we have that $\mathtt{size}(B) \leq l + \mathtt{size}(B_l)$, $l < 2|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})|$, and Δ_l is stable. Moreover, $|\mathtt{subterms}(\Delta_l)| \leq |\mathtt{subterms}(\Delta)|$ by Lemma 8.1.1 and $|\Delta_l| \leq 2|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})| - l$.

As a second step of the proof, let $\Delta_\lambda \subseteq \Delta_l$ be the set $\{e \in \Delta_l \mid B(e) = \{\lambda\}\}$, let $\theta = \mathtt{mgu}(\Delta_\lambda)$, let $\Gamma = \theta(\Delta_l \setminus \Delta_\lambda)$, and let $B_\Gamma : \Gamma \mapsto \mathcal{P}(\mathtt{pos}(\mathcal{F}))$ be defined as $B(\theta(e)) = B(e)$, for every $e \in (\Delta_l \setminus \Delta_\lambda)$. Then, $\mathtt{size}(B) \leq |\Delta_l| + \mathtt{size}(B_\Gamma) \leq |\Delta_l| \leq 2|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})| + \mathtt{size}(B_\Gamma)$. and B_Γ is a nonredundant distinguishing function of Γ, by Lemma 8.4.11. Also note that $|\mathtt{subterms}(\Gamma)| \leq |\mathtt{subterms}(\Delta_l)| \leq |\mathtt{subterms}(\Delta)|$. Our goal now is to prove a suitable bound on $\mathtt{size}(B_\Gamma)$.

Clearly, if $\Gamma = \emptyset$, then $\mathtt{size}(B_\Gamma) = 0$, $\mathtt{size}(B) \leq 2|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})|$, and we are done. Hence assume that $\Gamma \neq \emptyset$. In this case we have $\mathtt{size}(B) < 2|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})| + \mathtt{size}(B_\Gamma)$ as mentioned above and, since $|\mathtt{terms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})| \leq |\mathtt{subterms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})| = m$, $\mathtt{size}(B) \leq 2m - 1 + \mathtt{size}(B_\Gamma)$ holds. Note that, since Δ is stable, then Γ is also stable. By Lemma 8.4.14, there exists a non empty subset $\Gamma_{max} \subseteq \Gamma$ such that, for every term t occurring in $\mathtt{terms}(\Gamma_{max})$, t is not a proper subterm of any other term occurring in $\mathtt{terms}(\Gamma_{max})$, and t is not a subterm of any other term occurring in $\mathtt{terms}(\Gamma \setminus \Gamma_{max})$. Moreover, there must be an equation $(s \doteq t) \in \Gamma_{max}$ such that either s or t, say s, is a term of height greater than 0 and thus $s \in (\mathtt{subterms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y}))$. To see this, assume that all the terms in the equations of Γ_{max} are variables and note that, in that case, $\mathtt{mgu}(\Gamma \setminus \Gamma_{max})$ does not instantiate any term in Δ_{max}. This either contradicts the assumption of the lemma that $\forall i \in \{1, \ldots, n\}, u \in \{s_i, t_i\} : \mathtt{mgu}(\Delta \setminus \{s_i \doteq t_i\})(u)$ is not a variable, or the fact that B_Γ is a nonredundant distinguishing function since, by construction, it holds that $\forall e \in \Gamma : B(e) \neq \{\lambda\} \wedge B(e) \neq \emptyset$ and we assumed that $\Gamma \neq \emptyset$. For clarity, let us refer to such s as s_{max}. Now, consider the set $\{x_1, \ldots, x_k\}$ of all variables satisfying that there exists an equation of the form $(x_i \doteq v) \in \Gamma_{max}$, for some $i \in \{1, \ldots, k\}$ and term v, and let α_i be $\{x_i \mapsto f(y_{i,1}, \ldots, y_{i,\mathtt{arity}(f)})\}$, where the $y_{i,j}$'s are freshly introduced variables from $\mathcal{Y}$ and thus not in $\mathcal{X}$, and f is $\mathtt{root}(\mathtt{mgu}(\Gamma \setminus \{x_i \doteq v\})(x_i))$, which must be a non variable function symbol by the conditions of the lemma and thus the α_is are correctly defined. We define the substitution $\alpha = \bigcup_{i=1}^{k}(\alpha_i)$, Δ'

as $\alpha(\Gamma)$, and $B' : \Delta' \mapsto \mathcal{P}(\texttt{pos}(\mathcal{F}))$ as $B(\alpha(e)) = B(e)$, for every equation in $e \in \Gamma$. Note that B' is a nonredundant distinguishing function of Δ and that $\texttt{size}(B') = \texttt{size}(B_\Gamma)$. Hence, we have $\texttt{size}(B) \leq 2m - 1 + \texttt{size}(B_\Gamma) = 2m - 1 + \texttt{size}(B')$

Note the application of α does not instantiate variable occurring at depth greater than 0 in the terms in Γ, by construction of Γ_{max}. Recall that $B_\Gamma(e)$ is non empty and contains positions of length greater than 0, for every equation in $e \in \Gamma$. It follows that, by definition of α, $\forall t \in (\alpha(\Gamma_{max})) : \texttt{h}(s) > 0 \wedge \texttt{h}(t) > 0$ holds. Hence, let $\Delta'_{max} \subseteq \Delta'$ be $\alpha(\Gamma_{max})$ and note that $(\Delta'', B'') = \texttt{decompose}(\Delta', \Delta'_{max}, B')$ is well defined by Definition 8.4.16. Note that s_{max} does not occur in $\texttt{subterms}(\Delta'')$ and all the newly introduced variables from $\mathcal{Y}$ occur at positions λ in the terms of Δ'. Thus $|\texttt{subterms}(\Delta'') \setminus (\mathcal{X} \cup \mathcal{Y})| < |\texttt{subterms}(\Gamma) \setminus (\mathcal{X} \cup \mathcal{Y})| \leq m$. Also, note that, by Lemma 8.4.17, $\texttt{size}(B') = \texttt{size}(B'')$, B'' is a nonredundant distinguishing function of Δ''. Moreover, B'' and Δ'' satisfy the conditions of the lemma. By induction hypothesis, $\texttt{size}(B'') \leq |\texttt{subterms}(\Delta'') \setminus (\mathcal{X} \cup \mathcal{Y})|^2 \leq (m-1)^2$ and we have $\texttt{size}(B) \leq 2m-1+\texttt{size}(B_\Gamma) \leq 2m-1+\texttt{size}(B') \leq 2m-1+\texttt{size}(B'') \leq 2m-1+(m-1)^2 \leq m^2 = |\texttt{subterms}(\Delta) \setminus (\mathcal{X} \cup \mathcal{Y})|^2$, which concludes the proof. □

Lemma 8.4.20 *Let Δ be a set of equations over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and let $B : \Delta \mapsto \mathcal{P}(\texttt{pos}(\mathcal{F}))$ be a basis of Δ. Then,* $\texttt{size}(B) \leq 2|\texttt{subterms}(\Delta) \setminus \mathcal{X})|^3|X|$.

Proof. The lemma follows from Lemmas 8.4.6, 8.4.10, 8.4.19, and the observation that the size of a basis is at most twice the size of a weak basis. □

We are now ready to prove Theorem 8.3.1.

Theorem 8.4.11 *Let s, t be two terms. The set* $\texttt{SDW}(s \doteq t)$ *and a basis of* $\texttt{SDW}(s \doteq t)$ *can be computed in polynomial time.*

Proof. As commented above, the correctness of the algorithm in Figure 8.1 is straightforward. Moreover, by the previous lemma, it terminates in polynomial time. □

Note that the previous result holds even when the terms in the input equation are represented as a Directed Acyclic Graph (DAG), since the bound on the size of a basis given by Lemma 8.4.20 only depends on the number of different subterms and number of variables, which are linear with the size of a DAG representation of a term. Moreover, the result is independent of the arity of the function symbols in $\mathcal{F}$.

We proved that a useful representation of the potentially exponential set of distinguishing positions of two terms s, t can be computed in polynomial time an space. We believe that our result can be extended to multiequations, which leads to a polynomial time solution of a particular case of one context unification that is crucial to prove that the problem can be solved in polynomial time. An interesting question is whether our result can be extended to the case where s and t are compressed using a grammar-based representation, in the line of the work done in [GGSST10].

Chapter 9

Directions for further research

In previous chapter, we covered a variety of results related to variants of unification, term compression, and the combination of both. Naturally, the proposed extensions of the work presented in this thesis fall into the same categories.

Algorithms on compressed terms

In Chapter 3 we showed how some basic operations on terms such as checking equality, applying a substitution, computing a subcontext, a subterm, and a compressed representation of the preorder traversal of a term can be efficiently performed directly on an STG-compressed representation. Also, as mentioned in the introduction, the compressed linear submatching problem, i.e. finding an instance of a linear term s within a ground term t, is known to be solvable in polynomial time even when both input terms s and t are compressed using STGs [SS13]. The motivation for that work is in compressed term rewriting, since linear submatching corresponds to finding a redex of a left-linear term rewriting rule. A question that was left open in that work is whether a polynomial time algorithm also exists if we drop the linearity restriction, since NP-hardness is not known for this problem.

Another interesting question related to operations on terms is whether, given compressed terms s and t, and an order $<$ on terms, $s < t$ holds. This problem can be parameterized by the class of the ordering $<$, e.g. lexicographic path orderings or knuth-bendix orderings. At least for these two classes of orders on terms, this problem is not trivial.

More general grammar-based tree compression mechanism than STGs have been also studied. The definition of STGs given in Definition 3.2.6 does not allow repeated occurrences of parameters and hence, roughly speaking, STGs do not allow *copying*. If we drop that restriction, we obtain nonlinear STGs, a formalism equivalent to Lamping's sharing graphs [Lam90] that allows for a doubly exponential compression ratio [BLM08]. However, the existence of an efficient equivalence checking procedure for nonlinear STGs is open, since the best known complexity upper bound for this problem in PSPACE [BLM08]. I suspect that this bound can be improved to coNP, but the existence of a polynomial time algorithm does not seem plausible since, in the worst-case, $O(2^n)$ bits are needed to simply store the size of a term represented with a non-linear STG of size n. In any case, no hardness result is known.

As mentioned in the introduction, the STG encoding has been proven successful for XML representation and processing. Improvements in that direction include the study of self-indexing of STG-based representations, similarly to what has been successfully done in the case of strings. More concretely, in [CN12], Claude and Navarro presented a representation of a SCFG-compressed string that can be queried in logarithmic time with respect to the size of the compressed representation. The goal in this context is to, given an STG representation of a set of terms, use a reasonable space overhead to enrich the representation with additional precomputed data that allows to answer some common queries efficiently.

One context unification and unification on compressed terms

From the perspective of unification on compressed terms, we presented, in Chapter 4, efficient algorithms for first-order matching and unification. We also presented an experimental analysis of several possible implementations of such algorithms. Although those results have been already useful from a theoretical perspective to prove that compressed one context unification is in NP in Chapter 6, more practical applications should be explored. In particular, we would like to study the feasibility of using an STG-based representation for term indexing in the context of automatic theorem provers. However, efficient updates need to be supported over such representations. Updates on STGs have been considered in [FM07], but have not been implemented in any large system yet.

Regarding context matching, and as shown in the table in the contributions section of the introduction, there is a missing result regarding context matching with STG-compressed terms. In particular, it is not known whether the result presented in Chapter 7, stating that k-context matching can be solved in polynomial time with DAG-compressed terms, can be extended to the STG setting. The main idea behind the algorithm of Chapter 7 is based in the fact that the length of the position of the hole in the image of a context variable in a solution is polynomially bounded by the size of the input. This might no longer be true when the input is compressed using STGs and hence the algorithm of Chapter 7 cannot be directly adapted to the STG-compressed case.

An interesting variant of unification used in the context of XML processing is simulation unification [BS02]. Roughly speaking, simulation unification corresponds to term unification with context variables, sequence variables, and first-order variables, where at most one occurrence of each context and each sequence variable is allowed in the input terms, i.e. context and sequence variables are anonymous. Moreover, in some cases the direct subterms of a term are considered to be unordered. The case of simulation unification of terms s,t where t is ground and represented with an STG and s is represented explicitly is particularly interesting, since it corresponds to the situation where t comes from the data and is large but compressed and s comes from a program or query. Note that this case is similar to the subcontext matching problem mentioned above.

Another interesting problem to be studied in the compressed setting is antiunification. In contrast to term unification, antiunification corresponds to the task of constructing a common generalization of two given symbolic expressions. Analogously to the case of unification, different variants have been studied depending on what kinds of variables are allowed in the expressions and what notion of equality is used. Antiunification has been applied in program analysis, inductive learning, and clone detection [LMHH00, FM92, KLV11, BKLV13].

Finally, one of the natural extensions of the work presented in the last chapters of this thesis is to find a polynomial time algorithm for one context unification, if it exists. We already presented in Chapter 5 a polynomial time solution for a particular case and, in fact, extending the results of Chapter 8 to multiequations directly yields a polynomial time algorithm for another particular case.

Bibliography

[BBSW03] Sacha Berger, François Bry, Sebastian Schaffert, and Christoph Wieser. Xcerpt and visXcerpt: From pattern-based to visual querying of XML and semistructured data. In *VLDB*, pages 1053–1056, 2003.

[BKLV13] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. A variant of higher-order anti-unification. In *RTA*, pages 113–127, 2013.

[BKN87] Dan Benanav, Deepak Kapur, and Paliath Narendran. Complexity of matching problems. *J. Symb. Comput.*, 3(1/2):203–216, 1987.

[BLM08] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4-5):456–474, 2008.

[BLNW13] Alexander Bau, Markus Lohrey, Eric Nöth, and Johannes Waldmann. Compression of rewriting systems for termination analysis. In Femke van Raamsdonk, editor, *RTA*, volume 21 of *LIPIcs*, pages 97–112. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[BS01] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[BS02] François Bry and Sebastian Schaffert. Towards a declarative query and transformation language for xml and semistructured data: Simulation unification. In *ICLP*, pages 255–270, 2002.

[CDG$^+$07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc

Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

[CF07] Jorge Coelho and Mário Florido. Xcentric: logic programming for XML processing. In Irini Fundulaki and Neoklis Polyzotis, editors, *WIDM*, pages 1–8. ACM, 2007.

[CGG12] Carles Creus, Adrià Gascón, and Guillem Godoy. One-context unification with stg-compressed terms is in np. In *RTA*, pages 149–164, 2012.

[CLL$^+$02] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and Abhi Shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In *STOC*, pages 792–801, 2002.

[CLL$^+$05] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[CN12] Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *SPIRE*, pages 180–192, 2012.

[DST80] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.

[EN07] Katrin Erk and Joachim Niehren. Dominance constraints in stratified context unification. *Inf. Process. Lett.*, 101(4):141–147, 2007.

[FM92] Cao Feng and Stephen Muggleton. Towards inductive generalization in higher order logic. In *ML*, pages 154–162, 1992.

[FM07] Damien K. Fisher and Sebastian Maneth. Structural selectivity estimation for xml documents. In *ICDE*, pages 626–635, 2007.

[GGSS08] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Context matching for compressed terms. In *LICS*, pages 93–102, 2008.

[GGSS09] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification with singleton tree grammars. In *RTA*, pages 365–379, 2009.

[GGSS11] Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification and matching on compressed terms. *ACM Trans. Comput. Log.*, 12(4):26, 2011.

[GGSST10] Adrià Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context unification with one context variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

[GKS06] Georg Gottlob, Christoph Koch, and Klaus U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.

[GMR11] Adrià Gascón, Sebastian Maneth, and Lander Ramos. First-order unification on compressed terms. In *RTA*, pages 51–60, 2011.

[Gol81] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

[GT07] Sumit Gulwani and Ashish Tiwari. Computing procedure summaries for interprocedural analysis. In *ESOP*, pages 253–267, 2007.

[Jez12] Artur Jez. Faster fully compressed pattern matching by recompression. In *ICALP (1)*, pages 533–544, 2012.

[KLV11] Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. In *RTA*, pages 219–234, 2011.

[Kut02] Temur Kutsia. Pattern unification with sequence variables, flexible arity symbols. *Electr. Notes Theor. Comput. Sci.*, 66(5):52–69, 2002.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, pages 16–30, 1990.

[Lif07] Yury Lifshits. Processing compressed texts: A tractability border. In *CPM*, pages 228–240, 2007.

[LM06] Markus Lohrey and Sebastian Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theor. Comput. Sci.*, 363(2):196–210, 2006.

[LM11] Markus Lohrey and Christian Mathissen. Compressed membership in automata with compressed labels. In *CSR*, pages 275–288, 2011.

[LMHH00] Jianguo Lu, John Mylopoulos, Masateru Harao, and Masami Hagiya. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

[LMM13] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using repair. *Inf. Syst.*, 38(8):1150–1167, 2013.

[LMSS12] Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *J. Comput. Syst. Sci.*, 78(5):1651–1669, 2012.

[LNV05] Jordi Levy, Joachim Niehren, and Mateu Villaret. Well-nested context unification. In *CADE*, pages 149–163, 2005.

[Loh10] Markus Lohrey. Compressed membership problems for regular expressions and hierarchical automata. *Int. J. Found. Comput. Sci.*, 21(5):817–841, 2010.

[Loh12] Markus Lohrey. Algorithmics on slp-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[LSSV08] Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. The complexity of monadic second-order unification. *SIAM J. Comput.*, 38(3):1113–1140, 2008.

[LSSV11] 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.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[NPR97] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In *ACL*, pages 410–417, 1997.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA*, pages 460–470, 1994.

[Pla95] Wojciech Plandowski. *The Complexity of the Morphism Equivalence Problem for Context-Free Languages.* PhD thesis, Warsaw University, 1995.

[PW78] Mike Paterson and Mark N. Wegman. Linear unification. *J. Comput. Syst. Sci.*, 16(2):158–167, 1978.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[Ryt04] Wojciech Rytter. Grammar compression, lz-encodings, and string algorithms with implicit input. In *ICALP*, pages 15–27, 2004.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[SN10] Kunihiko Sadakane and Gonzalo Navarro. Fully-functional succinct trees. In *SODA*, pages 134–149, 2010.

[SS13] Manfred Schmidt-Schauß. Linear Compressed Pattern Matching for Polynomial rewriting (extended abstract). In *TERMGRAPH*, pages 29–40, 2013.

[SSS98] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, pages 61–75, 1998.

[SSS02] 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.

[SSS04] Manfred Schmidt-Schauß and Jürgen Stuber. The complexity of linear and stratified context matching problems. *Theory Comput. Syst.*, 37(6):717–740, 2004.

[SSS12] Manfred Schmidt-Schauß and Georg Schnitger. Fast equality test for straight-line compressed strings. *Inf. Process. Lett.*, 112(8-9):341–345, 2012.

[SSSA11] Manfred Schmidt-Schauß, David Sabel, and Altug Anis. Congruence closure of compressed terms in polynomial time. In *FroCoS*, pages 227–242, 2011.

[Vil04] Mateu Villaret. *On some Variants of Second Order Unification.* PhD thesis, Universitat Politècnica de Catalunya, 2004.