Unification on Compressed Terms

Author: Adrià Gascón
Director: Guillem Godoy

Màster en Computació
Departament de Llenguatges i Sistemes informàtics (LSI)
Setembre 2009

Contents

Acknowledgements [1]

Thanks to Guillem Godoy for introducing me to the world of research and showing me how to face interesting (and sometimes frustrating) problems with patience, passion, and most of all good mood.

Thanks to my family for always being more proud of me than what I would actually deserve.

[1] Part of this work was supported by the FORMALISM project (TIN2007-66523), funded by the Spanish government.

Abstract

First-order term unification is an essential concept in areas like functional and logic programming, automated deduction, deductive databases, artificial intelligence, information retrieval, compiler design, etc. We build upon recent developments in grammar-based compression mechanisms for terms and investigate algorithms for first-order unification and matching on compressed terms.

We prove that the first-order unification of compressed terms is decidable in polynomial time, and also that a compressed representation of the most general unifier can be computed in polynomial time. Furthermore, we present a polynomial time algorithm for first-order matching on compressed terms. Both algorithms represent an improvement in time complexity over previous results [GGSS09, GGSS08].

We use several known results on the tree grammars used for compression, called singleton tree grammars (STG)s, like polynomial time computability of several subalgorithmms: certain grammar extensions, deciding equality of represented terms, and generating their preorder traversal. An innovation is a specialized depth of an STG that shows that unifiers can be represented in polynomial space.

Chapter 1

Introduction

The task of solving equations is an important component of any mathematically founded science. In general, solving an equation $s \doteq t$ consists of finding a substitution σ for variables occurring in both expressions s and t such that $\sigma(s) = \sigma(t)$. The range for the variables, the kind of expressions s and t, and their semantics, as well as the semantics of $=$ depend on the context. By specifying some of these parameters we can define the well-known *first-order term unification problem.*

The first-order unification problem

In the context of this problem the expressions s and t are terms with leaf variables standing for terms, all function symbols are non-interpreted, and $=$ is interpreted as syntactic equality. Intuitively, we can see terms as trees, the widely-used data structure. To be more concrete, we usually consider terms built of functions symbols f, g, a, b (where f and g are binary, and a,b are nullary), and variable symbols x and y. Therefore, the *unification problem* $s \doteq t$ for terms $s = f(x, b)$ and $t = f(a, y)$ is concerned to the question: *is it possible to replace the variables x,y in s and t by terms such that the two terms obtained this way are (syntactically) equal*? In this example, if we replace x by a and y by b then s and t become equal, and we obtain an unified term, i.e. the resulting term after applying the substitution, $f(a, b)$. Hence, the substitution $\{x \mapsto a, y \mapsto b\}$ is called a *unifier* for s and t.

Robinson [Rob65] showed that the first-order Unification problem is decidable and that whenever a unifier exists, there always exists a *most general unifier*, i.e. a unifier such that every other unifier can be obtained by *instantiation.* Even more, in first-order unification, whenever this most general unifier exists, it is unique up to variable renaming. Robinson's algorithm for computing most general unifier requires exponential time and space in the worst case. A great deal of effort has gone into improving the efficiency of first-order unification. Among several other results, there are the ones by Venturini-Zilli [VZ75], reducing the complexity of Robinson's algorithm to quadratic time, and by Martelli and Montanari [MM82], presenting a linear time algorithm for unification.

The first-order matching problem

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. Like term unification, this is a common problem in areas like functional and logic programming, automated deduction, deductive databases, artificial intelligence, information retrieval, compiler design, etc.

Variants

In most applications of unification and matching, one is not interested just in the *decision problem*, which simply asks for a "yes" or "no" answer to the question commented above. A unification or matching algorithm should thus not only decide solvability of a given instance of these problems, but also either compute a *most general unifier*, i.e. a unifier such that every other unifier can be obtained by *instantiation*, or compute all solutions. The first-order term unification and matching problems are efficiently solvable, but their expressivity is often insufficient to deal with the current challenges in the areas mentioned above. For this reason, several variants and generalizations of these problems have been studied. Incorporating more complex interpretation of the function symbols and equality predicate under equational theories has been widely considered (see [BS94, BS01]). In this case, instead of requiring that the terms are made syntactically equal, equational unification is concerned to make the terms equivalent with respect to a congruence induced by certain equational axiom E. For example, if $E = f(a,a) \approx g(a,a)$, then the terms $f(a,x)$ and $g(x,a)$, which are not (syntactically) unifiable, are E-unifiable.

Another extension of first-order unification is concerned with allowing other kinds of variables related to terms. This is the case of *context variables*, i.e. variables which can be substituted by contexts, which are terms with a single hole (syntactically, the hole is a special constant denoted by $\bullet$). Context variables have arity one, hence they have a subterm t. Once they are instanciated by a context, the hole denotes where t has to be inserted. For example, we say that the context unification equation $F(f(h(a), h(a))) \doteq h(f(F(a), F(a)))$ has only one solution $\{F \mapsto h(\bullet)\}$ and the unified term is $h(f(h(a), h(a)))$. The context unification instance $F(f(x, b)) \doteq f(a, F(y))$ has several solutions such as $\{F \mapsto f(a, \bullet), x \mapsto a, y \mapsto b\}$, $\{F \mapsto f(a, f(a, \bullet)), x \mapsto a, y \mapsto b\}$, $\{F \mapsto f(a, f(a, f(a, \bullet))), x \mapsto a, y \mapsto b\}$, and so on. The unified terms are $f(a, f(a, b))$, $f(a, f(a, f(a, b)))$, and $f(a, f(a, f(a, f(a, b))))$ respectively. On the other hand, $f(a, F(x))$, and $F(f(b, a))$ are not unifiable. The *context unification problem* was first introduced by Comon [Com91] and its decidability still remains open. However, some particular cases has been already solved [SSS04, SS02, SSS02, LSSV06b, GGSS08]. Analogously to the first-order case, the context matching problem is the particular case of context unification where one of the sides of the equation contains no variables.

However, the extension of unification and matching treated in this work is concerned with reconsidering complexity issues for the first-order unification problem when applied to compressed input terms. In recent years there has been an increase of interest in compression mechanisms based on grammar representation, since other mechanisms can in general be efficiently simulated. These compression techniques were initially used for words [Pla95, Loh06, Lif07], and led to important results in string processing, with applications [HSTA00, GM02, LR06] in software/hardware verification, information retrieval, and bioinformatics. In that sense, *Straight-Line Programs (SLP)*, or the equivalent formalism of *Singleton Context Free Grammars (SCFG)*, are now a widely accepted formalism for text compression. Later, grammar-based compression was extended to terms/trees [BLM05, SS05, CDG^{+}97] with applications on XML tree structure compression [BLM05] and XPATH [LM05]. STG-based compressors have already been developed [MMS08]. Essentially, an SCFG, i.e. a context free grammar where all nonterminals generate a singleton language, is used for representing single words, and similarly, every nonterminal in a *singleton tree grammar (STG)* represents one tree. An STG can succinctly represent terms/trees which are exponentially big in size and height. Efficient algorithms have been developed for checking whether two compressed inputs represent the same word/term [Pla95, Loh06, Lif07], and for finding occurrences of one of them within the other (fully compressed pattern matching)[KRS95, KPR96, MST97, Lif07]. Recently, it was shown that tree grammars using multi-hole-contexts are polynomially equivalent to STGs [LMSS09]. STGs have also been used for complexity analysis of unification algorithms in [LSSV06b, LSSV06a], and the context matching problem [GGSS08].

Overview of results obtained in this work

In [GGSS09], and in [GGSS08], there were presented polynomial time algorithms for first-order unification and matching , in both cases with terms represented with STGs. As a nobel contribution we describe, in Chapter 3 and Chapter 4 respectively, faster algorithms for this two problems. Moreover, we believe that the presented solutions represent also a gain in simplicity which makes them easily implementable.

Chapter 2

Preliminaries

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+97], [Vil04], and [BS01].

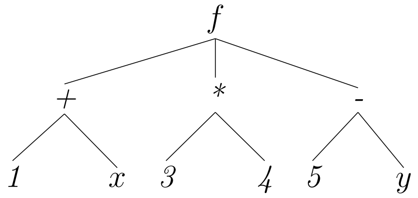
2.1 Terms

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



where f,+, $-$ *and* $*$ *are function symbols,* 1,3,4 *and* 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. We can regard a constant as a function with arity 0.

Once introduced the intuitive idea on what the terms are, we can define them, as well as the notation used in this document, in a more strict way. A *signature* $\mathcal{F}$ is a finite set of symbols. We also define *arity* as a mapping from $\mathcal{F}$ into N, where N denotes the set of natural numbers including 0. The *arity* of a symbol $f \in \mathcal{F}$ is $\mathtt{ar}(f)$. The set of symbols of arity i is denoted as $\mathcal{F}_i$. The symbols of arity $0, 1, 2, \ldots, i$ are respectively called constants/nullary, unary, ..., i-ary symbols. We assume that $\mathcal{F}$ contains at least one constant.

Let $\mathcal{X}$ be a set of symbols called *first-order variables*, denoted by x, y, x, which are nullary. We assume that the set $\mathcal{X}$ and $\mathcal{F}$ are disjoint. We employ the syntax of *second-order terms* (without abstraction) and denote terms as $s, t, u, v, \ldots$.

The set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of terms is recursively defined by:

- $\mathcal{F}_0 \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$.
- $\mathcal{X} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$.
- if $i \geq 1$, $f \in \mathcal{F}_i$ and $t_1, \ldots, t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, then $f(t_1, \ldots, t_i) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

If $\mathcal{X} = \emptyset$ then $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is also written $\mathcal{T}(\mathcal{F})$. Terms in $\mathcal{T}(\mathcal{F})$ are called *ground terms*. Hence, a term without occurrences of free variables is said to be *ground*.

2.2 Terms, trees, and positions

As commented above, it is easy to think about a term as a tree structure. At this point, an alternative definition for terms is presented to go further than just the intuitive idea.

A finite ordered *tree* t over a set of labels E is a mapping from a prefix-closed set $\mathtt{Pos}(t) \subseteq N^*$ into E, where N^* denotes the set of finite strings over N. Thus, a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ may be viewed as a finite ordered ranked tree, the leaves of which are labeled with variables or constant symbols and the internal nodes are labeled with symbols of positive arity, with out-degree equal to the arity of the label, i.e. a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ can be also defined as a partial function $t : N^* \to \mathcal{F} \cup \mathcal{X}$ with domain $\mathtt{Pos(t)}$ satisfying the following properties:

i. $\mathtt{Pos(t)}$ is finite, nonempty and prefix-closed

ii. $\forall i \in \texttt{Pos(t)}$, if $t(i) \in \mathcal{F}_n$, $n \geq 1$, then $\{j \mid i \cdot j \in \texttt{Pos(t)}\} = \{1, \ldots, n\}$

iii. $\forall i \in \texttt{Pos(t)}$, if $t(i) \in \mathcal{V} \cup \mathcal{F}_0$, then $\{j \mid i \cdot j \in \texttt{Pos(t)}\} = \emptyset$

where $\cdot$ denotes the concatenation.

Thus, each element in $\texttt{Pos(t)}$ is a *position*. Positions are denoted p, q, as sequences of positive integers. In $f(t_1, \ldots, t_n)$ the position of the i^{th} subterm, t_i, is i. The empty word is denoted λ, $p \prec q$ means the prefix relation, $p \cdot q$ the concatenation, and $t|_p$ the subterm at position p of t.

2.3 Subterms

A *subterm* $t|_p$ of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ at position p is recursively defined as

- $t|_\lambda = t$
- $f(t_1, \ldots, t_n)|_{i \cdot p} = t_i|_p$

where λ denotes the empty string.

2.4 Functions on terms

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

- $\texttt{height(t)} = 0$, $|t| = 1$ if $t \in \mathcal{V}$
- $\texttt{height(t)} = 0$, $|t| = 1$ if $t \in \mathcal{F}_0$
- $\texttt{height(t)} = 1 + \max_{1 \leq i \leq n}(\texttt{height(t}_\texttt{i}\texttt{)})$, $|t| = 1 + \sum_{i=0}^{n} |t_i|$ if t is of the form $t = f(t_1, \ldots, t_n)$

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

Example 2.4.1 *Let t be the term $f(g(a, g(c, x)), f(a, b, a), a)$. Then, the term $g(c, x)$ occurs at position* 12*. Hence, $t|_{12} = g(c, x)$.* $\texttt{height(t)} = 3$*, and* $|t| = 11$*.*

2.4.1 Preorder traversal of a term

We denote by $\mathtt{pre}(t)$ the preorder traversal (as a word) of a term t. It is recursively defined as $\mathtt{pre}(t) = t$, if t is a constant or a first-order variable, and $\mathtt{pre}(t) = f \cdot \mathtt{pre}(t_1) \cdot \ldots \cdot \mathtt{pre}(t_m)$, if $t = f(t_1, \ldots, t_m)$ with $m > 0$. Two arbitrary different trees may have the same preorder traversal, but when they represent terms over a fixed signature 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 $\{1, \ldots, |\mathtt{pre}(t)|\}$ of $\mathtt{pre}(t)$ and the positions $\mathtt{Pos}(t)$ of t, which associates every position $p \in \mathtt{Pos}(t)$ to the index $i \in \{1, \ldots, |\mathtt{pre}(t)|\}$ you find at $\mathtt{root}(t|p)$ while traversing the tree in preorder. We can recursively define the two mappings $\mathtt{pIndex}(t, p) \rightarrow \{1, \ldots, |\mathtt{pre}(t)|\}$ and $\mathtt{iPos}(t, i) \rightarrow \mathtt{Pos}(t)$ as follows. $\mathtt{pIndex}(t, \lambda) = 1$, $\mathtt{pIndex}(f(t_1, \ldots, t_m), i.p) = (1+|t_1|+\ldots+|t_{i-1}|)+\mathtt{pIndex}(t_i, p)$, $\mathtt{iPos}(t, 1) = \lambda$, and $\mathtt{iPos}(f(t_1, \ldots, t_m), 1 + |t_1| + \ldots + |t_{i-1}| + k) = i \cdot \mathtt{iPos}(t_i, k)$ for $1 \leq k \leq |t_i|$.

2.5 Substitutions

A *substitution* σ is a mapping from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The *domain* of σ is $\mathcal{X}$, althought sometimes it is assumed to be a subset $\mathcal{X}' \subset \mathcal{X}$ depending on the context. Substitutions mapping from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F})$ are called *ground*. By $\sigma(t)$ we denote the result of applying σ to the term t. We understand σ recursively extended to terms as $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$.

Example 2.5.1 *Let t be the term $f(f(x_1, a), x_2)$ and let σ be the substitution $\{x_1 \mapsto g(a, b), x_2 \mapsto b\}$. Then $\sigma(t) = f(f(g(a, b), a), b)$ and the domain of σ is $\{x_1, x_2\}$.*

2.6 Contexts

The structure of a *context* will be useful when compressing terms. The dag (directed acyclic graph) representation structure compresses terms in width by reusing multiple occurrences of subterms. Similarly, an STG-based representation compresses also in height by reusing multiple occurrences of contexts.

Intuitively, contexts are terms with a single occurrence of a hole, denoted $\bullet$, into which terms (or other contexts) may be inserted. We denote contexts by upper case letters C, D. We can provide a formal definition by considering a context to be a term in an extended signature that includes an extra constant symbol $\bullet$. Hence, the smallest context contains just the hole and has size 1. If C and D are contexts and t is a term, CD and Ct represent the term that is like C except that the occurrence of $\bullet$ is replaced by D and t, respectively. If $D_1 = D_2D_3$ for contexts D_1, D_2, D_3, then D_2 is called a *prefix* of D_1, and D_3 is called a *suffix* of D_1. The position of the hole in a context C is called *hole path*, denoted $\mathtt{hp}(C)$, and its length is denoted as $|\mathtt{hp}(C)|$.

Example 2.6.1 *Let C be the context $f(f(g(a,\bullet),a),b)$, D be the context $h(f(a,\bullet))$ and t be the term $g(c,x_1)$. Then $Ct = f(f(g(a,g(c,x_1)),a),b)$, $CD = f(f(g(a,h(f(a,\bullet))),a),b)$, $|C| = 7$, $|t| = 3$, $|Ct| = 9$. $\mathtt{hp}(C) = 112$, and $|\mathtt{hp}(C)| = 3$.*

2.7 Unification and matching

Very generally speaking, unification tries to identify two simbolic expressions by replacing certain sub-expresions(variables) by other expressions. Hence, this task consists on solving equations $s \doteq t$ by finding a substitution σ for variables ocurring in both expressions s and t such that $\sigma(s) = \sigma(t)$. In particular, 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 2.7.1 *The first-order unification problem for terms $s = f(a,x)$ and $t = f(y,f(a,b))$ has a solution $\sigma = \{x \mapsto f(a,b), y \mapsto a\}$. The terms $s' = f(a,x)$, $t' = f(x,f(a,b))$ cannot be unified.*

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 2.7.2 *Given a signature $\mathcal{F}$ 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 (the solution), such that $\sigma(s_i) = \sigma(t_i)$ for all i.*

A simple algorithm for first-order unification might be 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 $n_2 = \mathtt{height}(s)$.

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 the arity for the terminals in G is fixed, the index k corresponds to a unique position $p \in \texttt{Pos}(s) \cap \texttt{Pos}(t)$, as commented in Section 2.4.1. 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 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
            $\sigma := \sigma \cup \{x \rightarrow t\}$
            return True
        EndIf
    ElseIf $t = x$ is a variable Then
        If $x$ occurs in $s$ Then return False
        Else
            $\sigma := \sigma \cup \{x \rightarrow s\}$
            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 defined 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 $s = f(s_1,\ldots,s_m) \wedge t = f(t_1,\ldots,t_m)$, $m > 0$ Then
      return (Match($s_1,t_1$)
      $\wedge$ Match($\sigma(s_2),t_2$)
      $\vdots$
      $\wedge$ Match($\sigma(s_m),t_m$))
   EndIf
   If $s = f(s_1,\ldots,s_m) \wedge t = g(t_1,\ldots,t_n)$, $n,m \geq 0$ 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

This two generic algorithms for solving first-order unification and matching will be crucial later in the paper since our approaches consist on adapting each of them to the compressed case.

2.8 Representations for terms

2.8.1 DAGs

Definition 2.8.1 *A term dag is a directed acyclic graph whose nodes are labelled with function symbols, constants, or variables, whose outgoing edges from a node are ordered, and where the outdegree of any node labelled with a symbol f is equal to the arity of f.*

In such a graph, each node has an interpretation as a term, and we shall speak of nodes and terms if they were one and the same. If terms are large

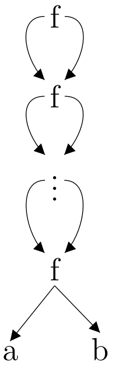


Figure 2.3: Encoding using dags of term in example 2.8.2

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 one would require exponential space using the term representation to represent t_n, whereas a *directed acyclic graph (dag)* representation requires linear space because of the reusement of the nodes. Hence, dags allow to represent terms of exponential width in linear space.

Example 2.8.2 *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 2.3.*

One of the reasons for the exponential execution time of Robinson's algorithm 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, which helps to avoid repeated calculations.

2.8.2 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)* [LSSV04, Pla94], which can only generate strings, extending the expressivity of SCFGs by terms and contexts. This is consistent with [BLM05], and also with the context free tree grammars in [CDG⁺97]. However, the latter are slightly more general in permitting contexts with several holes.

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 SCFG to represent terms we introduce *Singleton Tree Grammars (STG)*.

Definition 2.8.3 *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* $Z \rightarrow \alpha$ *where* $Z \in V$ *and* $\alpha \in (V \cup \Sigma)^*$ *. A rule* $Z \rightarrow \alpha$ *is called a* $Z - rule$*.*

Example 2.8.4 *A context-free grammar for the language consisting of all strings over* $\{a, b\}$ *for which the number of a's and b's are different is:*

$$\begin{array}{ll} S \rightarrow U & S \rightarrow V \\ U \rightarrow TaU & U \rightarrow TaT \\ V \rightarrow TbV & V \rightarrow TbT \\ T \rightarrow aTbT & T \rightarrow bTaT \\ T \rightarrow \lambda & \end{array}$$

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 non terminals T, U, V are recursive. For this reason, arbitrarily long strings may be generated. Furthermore, due to that recursivity and to the fact that there is more than one rule containing a given non terminal in its left-hand side, every non terminal can generate more than one string. SCFGs are called singleton because each non terminal generates just one string.

Definition 2.8.5 *A* singleton context free grammar (SCFG) *is a non-recursive context-free grammar such that for every nonterminal* Z *there is exactly one* $Z - rule$*. Then every non-terminal* Z *generates just one word, denoted* w_Z*, and we say that* Z *defines* w_Z*. We do not distinguish a particular start symbol. Hence, a singleton context-free grammar is defined as a 3-tuple* $G = (V, \Sigma, P)$*, analogously to context-free grammars.*

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

Now we can define *singleton tree grammars (STG)* as an extension of the already presented SCFGs in order to capture terms and contexts. Note that SCFGs can be obtained from STGs for the case of a monadic signature, i.e. all function symbols have arity one except for one constant.

Definition 2.8.6 *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 non-terminals, or non-terminals of arity* 0, $\mathcal{CN}$ *is a set of context non-terminals, or non-terminals of arity* 1, *and* Σ *is a signature of function symbols (the terminals), such that the sets* $\mathcal{TN}$, $\mathcal{CN}$, *and* Σ *are pairwise disjoint. The set of non-terminals* $\mathcal{N}$ *is defined as* $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. *The rules in* R *may be of the form:*

- $A \rightarrow f(A_1, \ldots, A_m)$, *where* $A, A_i \in \mathcal{TN}$, *and* $f \in \Sigma$ *is an* m*-ary terminal symbol.*
- $A \rightarrow C_1 A_2$ *where* $A, A_2 \in \mathcal{TN}$, *and* $C_1 \in \mathcal{CN}$.
- $C \rightarrow \bullet$ *where* $C \in \mathcal{CN}$.
- $C \rightarrow C_1 C_2$, *where* $C, C_i \in \mathcal{CN}$.
- $C \rightarrow f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)$, *where* $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m \in \mathcal{TN}$, $C, C_i \in \mathcal{CN}$, *and* $f \in \Sigma$ *is an* m*-ary terminal symbol.*
- $A \rightarrow A_1$, *(*λ*-rule) where* A *and* A_1 *are term non-terminals.*

Let $N_1 >_G N_2$ *for two non-terminals* N_1, N_2, *iff* $N_1 \rightarrow t$, *and* N_2 *occurs in* t. *The STG must be non-recursive, i.e. the transitive closure* $>_G^+$ *must be terminating. Furthermore, for every non-terminal* N *of* G *there is exactly one rule having* N *as left-hand side. Given a term* t *with occurrences of non-terminals, the derivation of* t *by* G *is an exhaustive iterated replacement of the non-terminals by the corresponding right hand sides. The result is denoted as* $w_{G,t}$. *In the case of a non-terminal* N *we also say that* N generates $w_{G,N}$. *We will write* w_N *when* G *is clear from the context.*

Note that we have used Σ instead of $\mathcal{F}$ for denoting the set of terminals of the grammar, although it is also a signature. We explain the reasons as follows. In this work, STGs are used for representing first-order terms and contexts. In particular, a terminal A of a 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.

Example 2.8.7 *The terms in the equation $s \doteq t$, where $s = f(g(a,b), h(x))$, and $t = f(h(b), g(a,x))$, are generated by term non-terminals A_s and A_t, respectively, in the following STG.*

$$\begin{array}{ll}
A_s \to f(A_1, A_2) & A_t \to f(A_3, A_4) \\
A_1 \to C_1 A_b & A_3 \to C_2 A_b \\
A_2 \to C_2 A_x & A_4 \to C_1 A_x \\
C_1 \to g(A_a, C_.) & C_2 \to h(C_.) \\
A_b \to b & A_x \to x \\
C_. \to \bullet & A_a \to a
\end{array}$$

A directed acyclic graph (dag) 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 2.8.8 *A* DAG *is an STG where the set of context non-terminals $\mathcal{CN}$ is empty, and moreover, there are only rules of the form $A \to f(A_1, \ldots, A_m)$.*

Example 2.8.9 *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 STG to represent t_n provides an efficient encoding (as shown in example 2.8.2 for the case of the dag representation).*

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

Nevertheless, STG-represented terms may have exponential height in 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 2.8.10 *The term $s = f^{2^n}(a)$ described by the following grammar would have exponencial height in a term or dag representation.*

$$\begin{array}{l}
s \to C_n A_a \\
A_a \to a \\
C_. \to \bullet \\
C_0 \to f(C_.) \\
C_1 \to C_0 C_0
\end{array}$$

$C_2 \rightarrow C_1C_1$
$C_3 \rightarrow C_2C_2$
$\vdots$
$C_n \rightarrow C_{n-1}C_{n-1}$

Definition 2.8.11 *The* size $|G|$ *of an STG* G *is the sum of the sizes of its rules, where the size of a rule* $N \rightarrow u$ *is* $1 + |u|$. *The* depth *within* G *of a non-terminal* N *is defined recursively as* $\mathtt{depth}(N) := 1 + \mathtt{max}\{\mathtt{depth}(N') \mid N'$ *is a non-terminal in* u *where* $N \rightarrow u \in G\}$ *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 non-terminals of* G*, and it is denoted as* $\mathtt{depth}(G)$.

Plandowski [Pla94, Pla95] proved decidability in polynomial time for the word problem for SCFG, i.e., given a SCFG P and two non-terminals A and B, to decide 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 [BLM05, SS05] Plandowski's result is generalized to STG. Since the result in [BLM05] is based on a linear reduction from terms to words and a direct application of Plandowski's result, it also holds for the Lifshits result. Hence, we have the following.

theorem 2.8.12 *([Lif07, BLM05]) Given a STG* G*, and two tree non-terminals* A, B *from* G*, it is decidable in time* $\mathcal{O}(|G|^3)$ *whether* $w_A = w_B$.

Several properties on STGs are efficiently decidable. The following lemmas will be used all along the paper.

Lemma 2.8.13 *Let* G *be an STG. The number* $|w_N|$*, for every non-terminal* N *of* G*, is computable in time* $\mathcal{O}(|G|)$.

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

- if $(N \rightarrow f(N_1, \ldots, N_m) \in G)$ then $|w_N| = 1 + |w_{N_1}| + \ldots + |w_{N_m}|$, where $N_1, \ldots, N_m$ are non-terminals of G and f is a function symbol with $\mathtt{ar}(f) = m$.
- if $N \rightarrow C_1N_2$ then $|w_N| = |w_{C_1}| + |w_{N_2}| - 1$, where C_1 is a context non-terminal and N_2 is a non-terminal 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 non-terminals of G, $|w_N|$ is computable in linear time over $|G|$ using a dynamic programming scheme. □

Lemma 2.8.14 *Given an STG G, a terminal α, and a non-terminal 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 squeme: note that α occurs in w_N iff either $w_N \to \alpha \in G$, or α occurs in $w_{N'}$ for some non-terminal N' occurring in the right-hand side of the rule for N. □

Chapter 3

First-order unification with STGs

In this section we prove that the first-order unification problem can be solved in polynomial time even when the input is compressed using STGs.

Definition 3.0.15 *The* first-order unification problem with STG *has an STG G representing first-order terms and contexts as input, plus two term non-terminals A_s and A_t of G representing terms $s = w_{G,A_s}$ and $t = w_{G,A_t}$. Its decisional version asks 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.

3.1 Outline of the algorithm

Given a STG G as a compressed representation of two terms s and t, we compute a minimal 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 non-unifiability. Otherwise, either $\mathtt{pre}(s)$ or $\mathtt{pre}(t)$, say $\mathtt{pre}(s)$, contains a variable x at k. Note that, since the arity for the terminals in G is fixed, the index k corresponds to a unique position $p \in \mathtt{Pos}(s) \cap \mathtt{Pos}(t)$, as commented in Section 2.4.1. 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 equal, in which case we state unifiability.

```
Input:  An STG $G$ and term non-terminals $A_s$ and $A_t$.
          (we write $s$ and $t$ for $w_{A_s}$ and $w_{A_t}$).
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 3.1: Unification Algorithm of STG-Compressed Terms

Note that, as commented in Section 2.7, 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. Hence, the difficulties are induced by the task of performing all the operations mentioned above on the compressed representation of terms. In [BLM05] it was shown how to succintly represent the preorder traversal word of a term generated by an STG using an SCFG. We reproduce this construction in Section 3.2 to compute an SCFG $\mathtt{Pre}_G$ with non-terminals $\mathcal{P}_s$ and $\mathcal{P}_t$ generating $\mathtt{pre}(s)$ and $\mathtt{pre}(t)$, respectively. We also need to compute, given $\mathtt{Pre}_G$, the minimal index k in which $\mathtt{pre}(s)$ and $\mathtt{pre}(s)$ differ. In Section 3.3 we show how to perform this task efficiently. Our approach is based on a recent result on compressed string processing [Lif07]. As commented above, k corresponds to a unique position $p \in \mathtt{Pos}(s) \cap \mathtt{Pos}(t)$. In Section 3.4, we present the procedure to, given G and k, extend G such that a new non-terminal 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 by a term u is easily representable with STGs by simply transforming x into a non-terminal x 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, we develop a notion of restricted depth, and show that its value is preserved along the execution, and that the size increase at each step can be

$$\begin{array}{rcl}
A \rightarrow f(A_1,\ldots,A_m) & \Rightarrow & \mathcal{P}_A \rightarrow f\mathcal{P}_{A_1}\ldots\mathcal{P}_{A_m} \\
A \rightarrow C_1A_2 & \Rightarrow & \mathcal{P}_A \rightarrow \mathcal{L}_{C_1}\mathcal{P}_{A_2}\mathcal{R}_{C_1} \\
A \rightarrow A_1 & \Rightarrow & \mathcal{P}_A \rightarrow \mathcal{P}_{A_1} \\
C \rightarrow C_1C_2 & \Rightarrow & \left\{ \begin{array}{l} \mathcal{L}_C \rightarrow \mathcal{L}_{C_1}\mathcal{L}_{C_2} \\ \mathcal{R}_C \rightarrow \mathcal{R}_{C_2}\mathcal{R}_{C_1} \end{array} \right. \\
C \rightarrow f(A_1,\ldots,A_{i-1},C_i,A_{i+1},\ldots,A_m) & \Rightarrow & \left\{ \begin{array}{l} \mathcal{L}_C \rightarrow f\mathcal{P}_{A_1}\ldots\mathcal{P}_{A_{i-1}}\mathcal{L}_{C_i} \\ \mathcal{R}_C \rightarrow \mathcal{R}_{C_i}\mathcal{P}_{A_{i+1}}\ldots\mathcal{P}_{A_n} \end{array} \right. \\
C \rightarrow \bullet & \Rightarrow & \left\{ \begin{array}{l} \mathcal{L}_C \rightarrow \lambda \\ \mathcal{R}_C \rightarrow \lambda \end{array} \right.
\end{array}$$

Figure 3.2: Generating the Preorder Traversal

bounded by this restricted depth, which is shown in Section 3.5.

3.2 Computing the preorder traversal of a term.

In [BLM05] it is shown how to construct, from a given STG G, an SCFG $\mathtt{Pre}_G$ 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 non-terminal A and its rule $A \rightarrow \alpha$ of G, which rule $\mathcal{P}_A \rightarrow \alpha'$ of $\mathtt{Pre}_G$ is required in order to make the non-terminal $\mathcal{P}_A$ of $\mathtt{Pre}_G$ satisfy $w_{\mathtt{Pre}_G,\mathcal{P}_A} = \mathtt{pre}(w_{G,A})$. To this end, for each context non-terminal C of G we also need non-terminals of $\mathtt{Pre}_G$ 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 non-terminal A of G, the corresponding newly generated non-terminal $\mathcal{P}_A$ of $\mathtt{Pre}_G$ generates $\mathtt{pre}(w_A)$.

Lemma 3.2.1 *Let G be a STG. A SCFG $\mathtt{Pre}_G$ of size $\mathcal{O}(|G|)$ can be constructed in time $\mathcal{O}(|G|)$ such that, for each non-terminal N of G, there exists a non-terminal $\mathcal{P}_N$ in $\mathtt{Pre}_G$ satisfying $w_{\mathtt{Pre}_G,\mathcal{P}_N} = \mathtt{pre}(w_{G,N})$.*

3.3 Computing the first different position of two words.

Given two non-terminals p_1 and p_2 of an SCFG P, we want to find the minimum 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 SCFG and equality is checkeable 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 3.3.1 *[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 non-terminals 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 pre-computation of Lemma 3.3.1 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. 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 non-terminals 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 3.3.1, each call takes time $\mathcal{O}(|P|)$, and at most $\mathtt{depth}(P)$ 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 3.3.1, that is, $\mathcal{O}(|P|^3)$.

Lemma 3.3.2 *Let P be an SCFG of size n, and let p_1, p_2 be non-terminals 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 $\mathcal{O}(|P|^3)$.*

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

Figure 3.3: Algorithm for the Index of the First Difference

3.4 Isolating variables

As commented in Section 2.4.1, the index k from the previous subsection defines a position $p = \texttt{iPos}(t,k)$ of a term t generated by an STG G. We show how to compute, in linear time, an extension of the STG G with a non-terminal generating $t|_p$. We use the SCFG $\texttt{Pre}_G$ presented in Definition 3.2.1.

Definition 3.4.1 *Let G be an STG. Let N a non-terminal of G, and let k be a natural number satisfying $k \leq |\texttt{Pre}(w_{G,N})|$. 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 non-terminal N' generating the subterm of w_{G,C_1} at position $\texttt{iPos}(w_{G,C_1},k)$. If N' is a context non-terminal then $\texttt{kExt}(G,N,k)$ additionaly contains the rule $A \to N'A_2$, where A is a new term non-terminal.*
- *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 non-terminal N' generating the subterm of w_{G,C_1} at position $\texttt{iPos}(w_{G,C_1},k)$. If N' is a context non-terminal then $\texttt{kExt}(G,N,k)$ additionaly contains the rule $C \to N'C_2$, where C is a new context non-terminal.*
- *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 $\texttt{kExt}(G,N,k) = \texttt{kExt}(G,N_2,k-k')$.*
- *If $(N \to C_1N_2) \in G$ and $|w_{\texttt{Pre}_G,\mathcal{L}_{C_1}}|+|w_{N_2}| < k$ then $\texttt{kExt}(G,N,k) = \texttt{kExt}(G,C_1,k-|w_{N_2}|+1)$.*
- *If $(N \to A_2) \in G$ then $\texttt{kExt}(G,N,k) = \texttt{kExt}(G,A_2,k)$.*
- *In any other case $\texttt{kExt}(G,N,k)$ is* undefined.

Lemma 3.4.2 *Let G be an STG. Let N a non-terminal of G, and let k be a natural number such that $k \leq |\texttt{Pre}(w_{G,N})|$. Then G can be extended to a STG G' in time $\mathcal{O}(|G|)$ with $\mathcal{O}(\texttt{depth}(G))$ new non-terminals such that one of them generates the subterm of $w_{G,N}$ at position $\texttt{iPos}(w_{G,N}, k)$.*

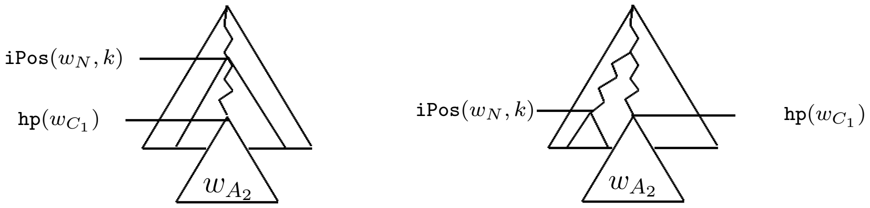
Proof.

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

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

- 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 $\texttt{iPos}(w_N, k)$, it holds that $\texttt{iPos}(w_N, k) = i \cdot \texttt{iPos}(w_{N_i}, k - k')$. Hence, $w_N|_{\texttt{iPos}(w_N,k)} = w_{N_i}|_{\texttt{iPos}(w_{N_i},k-k')}$ by definition of subterm of w_N. Moreover, since in this case $\texttt{kExt}(G, N, k) = \texttt{kExt}(G, N_i, k - k')$, the fact that $\texttt{kExt}(G, N, k)$ generates $w_N|_{\texttt{iPos}(w_N,k)}$ follows by induction hypothesis.

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

  $\texttt{iPos}(w_N, k) \prec \texttt{hp}(w_{C_1})$ $\qquad$ $\texttt{iPos}(w_N, k)$ and $\texttt{hp}(w_{C_1})$ disjoint

  In the former case, $w_{C_1}|_{\texttt{iPos}(w_{C_1},k)}$ is a prefix of w_{C_1} and $w_N|_{\texttt{iPos}(w_N,k)} = w_{C_1}|_{\texttt{iPos}(w_{C_1},k)} w_{A_2}$. In this case $\texttt{kExt}(G, N, k)$ is constructed by using $\texttt{kExt}(G, C_1, k)$, which constains a new non-terminal N' generating $w_{C_1}|_{\texttt{iPos}(w_{C_1},k)}$ by induction hypothesis, plus the rule $A \to N' A_2$, where A is a new term non-terminal. Hence, it holds that $w_A = w_{N'} w_{A_2} = w_{C_1}|_{\texttt{iPos}(w_N,k)} w_{A_2} = w_N|_{\texttt{iPos}(w_N,k)}$ and thus $\texttt{kExt}(G, N, k)$ generates $w_N|_{\texttt{iPos}(w_N,k)}$. In the latter case, $w_N|_{\texttt{iPos}(w_N,k)} = w_{C_1}|_{\texttt{iPos}(w_{C_1},k)}$. 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 $k \leq |w_{\mathtt{Pre}_G,\mathcal{L}_{C_1}}|$ is solved analogously to the previous one.

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

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

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

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

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

3.5 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 non-terminal and adding the necessary rules for making X generate the replacing value. We define this notion of application of a substitution as follows.

Definition 3.5.1 *Let G be an STG. Let X be a terminal representing a first-order variable and let A be a term non-terminal 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 non-terminal.*

When one or more substitutions of this form are applied, in general the depth of the non-terminals 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 non-terminals N belonging to a special set V satisfying the following condition.

Definition 3.5.2 *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 non-terminal A in V, the rule of G of the form $A \to \alpha$ is a λ-rule.*

Definition 3.5.3 *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ be an STG and let V be a λ-set for G. For every non-terminal 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 $\mathtt{max}(\emptyset) = 0$).*

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

The `Vdepth` *of G is the maximum of the* `Vdepth` *of its non-terminals.*

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

Lemma 3.5.4 *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 non-terminal of G. Let G' be $\{X \mapsto A\}(G)$. Then, for any non-terminal N of G it holds that $\mathtt{Vdepth}_{G'}(N) = \mathtt{Vdepth}_G(N)$.*

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

Lemma 3.5.5 *Let G be an STG, let C be a context non-terminal of G, let V be a λ-set for G, let k be a natural number such that $w_C|_{\texttt{iPos}(w_C,k)}$ is a context, and let G' be $\texttt{kext}(G, C, k)$.*

Then, for every non-terminal N of G it holds that $\texttt{Vdepth}_G(N) = \texttt{Vdepth}_{G'}(N)$, and for every new non-terminal N' in G' and not in G, it holds that $\texttt{Vdepth}_{G'}(N') \leq \texttt{Vdepth}_G(C)$. Moreover, the number of new added non-terminals is bounded by $\texttt{Vdepth}_G(C)$.

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

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

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

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

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

□

Lemma 3.5.6 *Let G be an STG, let N be a non-terminal of G, let V be a λ-set for G, let k be a natural number satisfying $k \leq |\texttt{Pre}(w_{G,N})|$, and let G' be $\texttt{kext}(G, N, k)$.*

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

Proof. The identity $\texttt{Vdepth}_G(N'') = \texttt{Vdepth}_{G'}(N'')$ for each non-terminal N'' of G is straightforward from the fact that $\texttt{kext}(G, N, k)$ does not change the rules for the non-terminals occurring in G. We will prove the fact that $\texttt{Vdepth}_{G'}(N'') \leq \texttt{Vdepth}(G)$ for each new non-terminal N'' in G' and not in G, plus the fact that at most $\texttt{Vdepth}(G)$ new non-terminals have been added by induction on $\texttt{depth}_G(N)$. The base case ($\texttt{depth}(N) = 1$) trivially holds since, in this case, the STG G is not modified. For the induction step we distinguish cases according to the definition of $\texttt{kExt}(G, N, k)$. The only interesting cases are either when $(N \to C_1A_2) \in G$ and $k \leq |w_{\texttt{Pre}_G,\mathcal{L}_{C_1}}|$ or $(N \to C_1C_2) \in G$ and $k \leq |w_{\texttt{Pre}_G,\mathcal{L}_{C_1}}|$. Note that these are the only cases in which the grammar might be extended with new non-terminals 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 $k \leq |w_{\texttt{Pre}_G,\mathcal{L}_{C_1}}|$. In this case the non-terminal N' in $\texttt{kext}(G, C_1, k)$ generating the subterm of w_{G,C_1} at position $\texttt{iPos}(w_{G,C_1}, k)$ is a either a term non-terminal or a context non-terminal. We will solve the two cases separately. First assume that N' is a term non-terminal. In this case $\texttt{kext}(G, N, k)$ is constructed as $\texttt{kext}(G, C_1, k)$. Since $\texttt{Vdepth}(C_1) < \texttt{Vdepth}(N)$, the lemma holds by induction hypothesis in this case. On the other hand, if N' is a context non-terminal, the construction of $\texttt{kext}(G, N, k)$ is done by computing $\texttt{kext}(G, C_1, k)$ and adding the rule $A \to N'A_2$, where A is an additional new term non-terminal. By Lemma 3.5.5, for all the new non-terminals N'' in $\texttt{kext}(G, C_1, k)$ and not in G, $\texttt{Vdepth}_{G'}(N'') \leq \texttt{Vdepth}_G(C_1)$. Moreover, the number of new added non-terminals is bounded by $\texttt{Vdepth}_G(C_1)$. Hence, $\texttt{Vdepth}_{G'}(N') \leq \texttt{Vdepth}_G(C_1)$

and, since $\texttt{Vdepth}(C_1) < \texttt{Vdepth}(N)$, at most $\texttt{Vdepth}_G(N) \leq \texttt{Vdepth}(G)$ new non-terminals have been added in the construction of $\texttt{kext}(G, N, k)$. Furthermore, since $\texttt{Vdepth}_G(N) = 1 + \texttt{max}(\texttt{Vdepth}_G(C_1), \texttt{Vdepth}_G(A_2))$, $\texttt{Vdepth}_{G'}(A) = 1 + \texttt{max}(\texttt{Vdepth}_{G'}(N'), \texttt{Vdepth}_{G'}(A_2))$ and $\texttt{Vdepth}_G(A_2) = \texttt{Vdepth}_{G'}(A_2)$, it also holds that $\texttt{Vdepth}_{G'}(A) \leq \texttt{Vdepth}_G(N) \leq \texttt{Vdepth}(G)$. □

3.6 A polynomial time algorithm for first-order unification with STGs

From a high level perspective the structure of our algorithm described in Section 3.1 is very simple and rather standard. Most algorithms for first-order unification are variants of this scheme. They represent the terms with directed acyclic graphs (dags), implemented somehow, in order to avoid the space explosion due to the repeated instantiation of variables by terms. In our setting, those terms are represented by STGs. In fact, the input is an STG G, and two term non-terminals A_s and A_t representing s and t, respectively. Since our algorithm is just an adaptation of the algorithm defined in Figure 2.1 to the case where the inputted terms are represented using STGs we will not argue about its correctness.

In previous sections we showed how to efficiently perform all the required operations on STGs: Decide whether s and t are equal, generate a compressed representation for $\texttt{pre}(s)$ and $\texttt{pre}(t)$, look for the minimum index k such that $\texttt{pre}(s)[k] \neq \texttt{pre}(s)[k]$, construct the term $t|_p$, where $p = \texttt{iPos}(t, k)$, and replace the variable $x = s|_p$ by $t|_p$ everywhere.

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 non-terminal). 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 3.5.4 and 3.5.6. Moreover, by Lemma 3.5.6, at most n new non-terminals 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 3.6.1 *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.*

3.6.1 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 $R = \{A_t \to g(B_1, A), A_s \to g(B_2, A), A \to C_4 A_a, C_4 \to C_3 C_3, C_3 \to C_2 C_2, C_2 \to C_1 C_1, C_1 \to C_0 C_0, C_0 \to f(C_.), C_. \to \bullet, A_a \to a, D \to C_3 C_2, B_1 \to D B_x, B_2 \to C_4 B_y, B_x \to x, B_y \to y\}$, 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

$\{\mathcal{P}_{A_t} \to g\mathcal{P}_{B_1}\mathcal{P}_A, \mathcal{P}_{A_s} \to g\mathcal{P}_{B_2}\mathcal{P}_A, \mathcal{P}_A \to \mathcal{L}_{C_4}\mathcal{P}_{A_a}\mathcal{R}_{C_4}, \mathcal{P}_{A_a} \to a, \mathcal{P}_{B_1} \to \mathcal{L}_D\mathcal{P}_{B_x}\mathcal{R}_D, \mathcal{L}_D \to \mathcal{L}_{C_3}\mathcal{L}_{C_2}, \mathcal{R}_D \to \mathcal{R}_{C_2}\mathcal{R}_{C_3}, \mathcal{P}_{B_x} \to x, \mathcal{P}_{B_2} \to \mathcal{L}_{C_4}\mathcal{P}_{B_y}\mathcal{R}_{C_4}, \mathcal{L}_{C_4} \to \mathcal{L}_{C_3}\mathcal{L}_{C_3}, \mathcal{L}_{C_3} \to \mathcal{L}_{C_2}\mathcal{L}_{C_2}, \mathcal{L}_{C_2} \to \mathcal{L}_{C_1}\mathcal{L}_{C_1}, \mathcal{L}_{C_1} \to \mathcal{L}_{C_0}\mathcal{L}_{C_0}, \mathcal{L}_{C_0} \to f\mathcal{L}_{C_.}, \mathcal{L}_{C_.} \to \lambda, \mathcal{R}_{C_.} \to \lambda, \mathcal{R}_{C_0} \to \mathcal{R}_{C_.}, \mathcal{R}_{C_1} \to \mathcal{R}_{C_0}\mathcal{R}_{C_0}, \mathcal{R}_{C_2} \to \mathcal{R}_{C_1}\mathcal{R}_{C_1}, \mathcal{R}_{C_3} \to \mathcal{R}_{C_2}\mathcal{R}_{C_2}, \mathcal{R}_{C_4} \to \mathcal{R}_{C_3}\mathcal{R}_{C_3}, \mathcal{P}_{B_y} \to y, \}$. 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 $\mathtt{Pre}_G$ 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 $\mathtt{index}(\mathcal{P}_{A_t}$,$\mathcal{P}_{A_s}$,0,$\mathtt{Pre}_G)$, the following sequence of calls is produced: $\mathtt{index}(\mathcal{P}_{A_t}, \mathcal{P}_{A_s}, 0, \mathtt{Pre}_G)$, $\mathtt{index}(\mathcal{P}_{B_1}, \mathcal{P}_{A_s}, 1, \mathtt{Pre}_G)$, $\mathtt{index}(\mathcal{P}_{B_x}, \mathcal{P}_{A_s}, 13, \mathtt{Pre}_G)$. 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 $\mathtt{iPos}(w_{G,A_s}, 14) = 1^{13}$. We compute now and extension $\mathtt{kExt}(G, A_s, 14)$ of G, as described in Definition 3.4.1, such that a new term non-terminal A'_s generates $w_{A_s}|_{1^{13}}$. We obtain the following set of rules, where rules in bold correspond to the added non-terminals due to the `kext` constructions w.r.t to the STG G given as input: $\{\mathbf{A'_s} \to \mathbf{C_2 B_y}, A_t \to g(B_1, A), A_s \to g(B_2, A), A \to C_4 A_a, C_4 \to C_3 C_3, C_3 \to C_2 C_2, C_2 \to C_1 C_1, C_1 \to C_0 C_0, C_0 \to f(C_.), C_. \to \bullet, A_a \to a, D \to C_3 C_2, B_1 \to D B_x, B_2 \to C_4 B_y, B_x \to x, B_y \to y\}$.

Note that, in the extended grammar, $w_{A'_s} = w_{A_s}|_{\mathtt{iPos}(w_{G,A_s},14)} = w_{A_s}|_{1^{13}} = f^4(y)$. 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 2.8.14. Finally, we perform the substitution $\{x \mapsto A'_s\}(G)$ by converting x into a non-terminal of the grammar generating $w_{A'_s}$ as stated in Definition 3.5.1. The set of rules of the obtained grammar G' after the `kext` construction and this assignment is $\{\mathbf{x} \to \mathbf{A'_s}, \mathbf{A'_s} \to \mathbf{C_2 B_y}, A_t \to g(B_1, A), A_s \to g(B_2, A), A \to C_4 A_a, C_4 \to C_3 C_3, C_3 \to C_2 C_2, C_2 \to C_1 C_1, C_1 \to C_0 C_0, C_0 \to f(C_.), C_. \to \bullet, A_a \to a, D \to C_3 C_2, B_1 \to D B_x, B_2 \to C_4 B_y, B_x \to x, B_y \to y\}$.

Note that $w_{G',A'_s} = w_{G,A_s}|_{\mathtt{iPos}(w_{G,A_s},14)} = w_{G,A_s}|_{1^{13}} = f^4(y)$, 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 $\sigma(x) = w_{G',x}$.

Chapter 4

First-order matching with STGs

In this section we prove that the first-order matching problem can be solved in polynomial time even when the input is compressed using STGs.

Definition 4.0.2 *The* first-order matching problem with STG *has an STG G representing first-order terms and contexts as input, plus two term non-terminals A_s and A_t of G representing terms $s = w_{G,A_s}$ and $t = w_{G,A_t}$, where t is ground. Its decisional version asks for the existence of a substitution σ such that $\sigma(s) = t$ whereas its computational version asks for a representation of σ.*

First-order matching is a particular case of first-order unification. However, taking advantatge of the fact that one of the terms is ground leads to a faster algorithm with respect to the one presented in the previous chapter. We also improve previous results for this problem [GGSS08].

4.1 Outline of the algorithm

The structure of our algorithm is sketched in Figure 4.1. Note that, as commented in Section 2.7, 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 on a STG G as a compressed representation of two terms s and t.

As in the first-order unification case, the algorithm works with 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 ocurrence 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 syntantically 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 previous section for the unification 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 non-terminals $A_s$ 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 minimum index $k$ such that pre($s$)[$k$] $= x \in \mathcal{X}$.
   If iPos($t, k$) is undefined Then Halt stating that the initial $s$ and $t$ match.
   Extend $G$ by the assignment $\{x \mapsto t|_p\}$, where $p =$ 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 4.1: Matching Algorithm for STG-Compressed Terms

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

4.2 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 solved efficiently as stated in the following Lemma.

Lemma 4.2.1 *Let P be a SCFG, and let p be a non-terminal of P representing the preorder traversal word of a first-order term. Then, the minimum index k such that $w_p[k]$ is a variable can be computed in time $\mathcal{O}(|P|)$.*

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

$$\texttt{index}(p,P) = \begin{cases} 1 & \text{, if } p \to \alpha \in P \land \alpha \in \mathcal{X} \\ \texttt{index}(p_1,P) & \text{, if } (p \to p_1 p_2) \in P \land \\ & \exists x \in \mathcal{X} : x \text{ occurs in } w_{P,p_1} \\ |w_{P,p_1}| + \texttt{index}(X_2,P) & \text{, Otherwise.} \end{cases}$$

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 $\texttt{index}(p, P)$ computes the minimum index k such that $w_p[k]$ is a variable can be shown by induction on $\texttt{depth}(p)$. With respect to the time complexity, for each non-terminal p of a SCFG P, both the number $|w_p|$ and whether w_p contains a variable can be precomputed in linear time as stated in Lemmas 2.8.13 and 2.8.14, respectively. Once this precomputations are done $\texttt{index}(p, P)$ can be computed by a single run over the rules of P and hence, it runs also in linear time. □

4.3 A polynomial time algorithm for first-order matching with STGs

The algorithm presented in the previous section 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 $V := \mathcal{X}$ 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 non-terminal). As in the unification case, the final size of the grammar is bounded by $m + |V|n$ thanks to Lemmas 3.5.4 and 3.5.6. Our algorithms iterates at most V times. By Lemmas 3.4.2, and 4.2.1 each iteration takes linear time. Finally we check equality of two words generated by a SCFG P, which takes time $\mathcal{O}(|P|^3)$ thanks to Theorem 2.8.12. Hence, we have the following:

theorem 4.3.1 *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 inputted STG, n represents its depth, and V represents the set of different first-order variables occurring in the inputted terms). This holds for the decision question, as well as for the computation of the unifier, whose components are represented by the final STG.*

Chapter 5

Conclusion & further work

We presented instantiation-based algorithms for the first-order matching problem and the first-order unification problem, that can be immediately executed on the compressed representation of large terms and run in polynomial time on the size of the representation. This results represent an improvement in time complexity with respect to previous work. Furthermore, we believe that the obtained algorithms represent also a gain in simplicity whichs makes their implementation feasible. It would be also interesting to investigate optimizations for these algorithms, as well as finding an improved upper bound. We also believe that it would be natural to consider the context matching problem using an STG encoding for terms under certain restrictions like fixing the number of context variables (this restriction was already consider using a dag representation in [GGSS08]). Finally, we think that our techniques could be useful to decide the one context unification problem in NP when the input is represented by an STG. This problem has been solved for plain terms as input in [GGSST09].

This project has been useful for me for being introduced in several research tasks such as those directly related to problem solving, those related to the composition of a paper showing the obtained results as well as the experience of going throught a revision process for a conference. Furthermore, in the context of this project I had the oportunity of presenting a paper in an international conference. Without a doubt this has been a rewarding task.

Bibliography

[BLM05] G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML documents. In *Proc. of DBPL 2005*, volume 3774 of *LNCS*, pages 199–216, 2005.

[BS94] F. Baader and J. Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, 1994.

[BS01] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science and MIT Press, 2001.

[CDG$^+$97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997. release 1.10.2002.

[Com91] H. Comon. Completion of rewrite systems with membership constraints. *Journal of Symbolic Computation*, 25(4):397 – 419, 1991.

[GGSS08] A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context matching for compressed terms. In *23rd IEEE LICS*, pages 93–102, 2008. http://www.lsi.upc.edu/ ggodoy/publications.html.

[GGSS09] A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification with singleton tree grammars. In *RTA*, pages 365–379. Springer, 2009.

[GGSST09] A. Gascón, G. Godoy, M. Schmidt-Schauß, and A. Tiwari. Context Unification with One Context Variable. *Journal of Symbolic Computation*, 2009. To appear.

[GM02] B. Genest and A. Muscholl. Pattern matching and membership for hierarchical message sequence charts. In *Proc. of LATIN 2002*, pages 326–340. Springer-Verlag, 2002.

[HSTA00] M. Hirao, A. Shinohara, M. Takeda, and S. Arikawa. Fully compressed pattern matching algorithm for balanced straight-line programs. In *SPIRE '00*, page 132, Washington, DC, USA, 2000. IEEE Computer Society.

[KPR96] M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *In Proc. 4th Scandinavian Workshop on Algorithm Theory*, pages 392–403. SpringerVerlag, 1996.

[KRS95] M. Karpinski, W. Rytter, and A. Shinohara. Pattern-matching for strings with short description. In *CPM '95*, pages 205–214. Springer-Verlag, 1995.

[Lif07] Y. Lifshits. Processing compressed texts: A tractability border. In *CPM 2007*, pages 228–240, 2007.

[LM05] M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. In *Proc. of the 10th CIAA '05*, 2005.

[LMSS09] M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction in grammar-compressed trees. In *12th FoSSaCS*, volume 5504 of *LNCS*, pages 212–226. Springer, 2009.

[Loh06] M. Lohrey. Word problems and membership problems on compressed words. *SIAM Journal on Computing*, 35(5):1210–1240, 2006.

[LR06] S. Lasota and W. Rytter. Faster algorithm for bisimulation equivalence of normed context-free processes. In *Proc. MFCS'06*, volume 4162 of *LNCS*, pages 646–657. Springer-Verlag, 2006.

[LSSV04] J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic second-order unification is NP-complete. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 55–69. Springer, 2004.

[LSSV06a] J. Levy, M. Schmidt-Schauß, and M. Villaret. Bounded second-order unification is NP-complete. In *Proc. RTA-17*, volume 4098 of *LNCS*, pages 400–414. Springer, 2006.

[LSSV06b] J. Levy, M. Schmidt-Schauß, and M. Villaret. Stratified context unification is NP-complete. In *Proc. Third IJCAR 2006*, volume 4130 of *LNCS*, pages 82–96. Springer, 2006.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. on programming languages and systems*, 4(2):258–282, 1982.

[MMS08] S. Maneth, N. Mihaylov, and S. Sakr. XML tree structure compression. *DEXA*, 0:243–247, 2008.

[MST97] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. 8th CPM*, number 1264 in LNCS, pages 1–11. Springer-Verlag, 1997.

[Pla94] W. Plandowski. Testing equivalence of morphisms in context-free languages. In Jan van Leeuwen, editor, *Proc. 2nd ESA'94*, volume 855 of *LNCS*, pages 460–470, 1994.

[Pla95] W. Plandowski. *The Complexity of the Morphism Equivalence Problem for Context-Free Languages*. PhD thesis, Department of Mathematics, Informatics and Mechanics, Warsaw University, 1995.

[Rob65] J.A. Robinson. A machine oriented logic based on the resolution principle. *J. of the ACM*, 12(1):23–41, 1965.

[SS02] M. Schmidt-Schauß. A decision algorithm for stratified context unification. *J. of Logic and Computation*, 12(6):929–953, 2002.

[SS05] M. Schmidt-Schauß. Polynomial equality testing for terms with shared substructures. Frank report 21, Institut für Informatik. FB Informatik und Mathematik. J. W. Goethe-Universität Frankfurt am Main, November 2005.

[SSS02] M. Schmidt-Schauß and K. U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symb. Comput.*, 33(1):77–122, 2002.

[SSS04] M. Schmidt-Schauß and J. Stuber. On the complexity of linear and stratified context matching problems. *Theory of Computing Systems*, 37:717–740, 2004.

[Vil04] M. Villaret. *On some variants of second-order unification.* PhD thesis, Universitat Politècnica de Catalunya, 2004.

[VZ75] M. Venturini-Zilli. Complexity of the unification algorithm for first-order expressions. *Calcolo*, 12:361–371, 1975.