Unification with Singleton Tree Grammars

Adrià Gascón and Guillem Godoy and Manfred Schmidt-Schauß

1 LSI Department, Universitat Politècnica de Catalunya, Jordi Girona, 1-3 08034
   Barcelona, Spain, adriagascon@gmail.com ggodoy@lsi.upc.edu

2 Dept. Informatik und Mathematik, Instf. Informatik, J.W. Goethe-University,
   D-60054 Frankfurt, Germany, {schauss}@i-informatik.uni-frankfurt.de

Abstract. First-order term unification is an essential concept in areas like
functional and logic programming, automated deduction, deductive
specialization, artificial intelligence, information retrieval, compiler design,
etc. We build upon recent developments in general grammar-based
compression mechanisms for terms, which are more general than dags and
investigate algorithms for first-order unification of 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
general unifier can be computed in polynomial time.

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

1 Introduction

Solving equations is an important task in any mathematically founded science
and deserves thorough investigations. In general, solving an equation \( s = t \) con-
ists of finding a substitution \( \sigma \) for variables occurring in both expressions \( s \) and
\( t \) such that \( \sigma(s) = \sigma(t) \). The range of the variables, the kind of expressions \( s \)
and \( t \), and their semantics, as well as the semantics of \( = \) depend on the context.

By restricting some parameters we obtain the well-known first-order term unifi-
cation problem, where the expressions \( s \) and \( t \) are terms with variables standing
for terms, function symbols are uninterpreted, and \( = \) is interpreted as syntactic
equality. Therefore, the term unification problem asks for a substitution \( \sigma \) that
maps the variables to first-order terms such that \( \sigma(s) \) and \( \sigma(t) \) are syntactically
equal. For example, the first-order unification instance \( f(f(x_2,x_2),f(x_3,x_3)) = f(x_1,x_2) \) has a solution \( \{ x_1 \mapsto f(f(x_3,x_3),f(x_3,x_3)), x_2 \mapsto f(x_3,x_3) \} \). First-
order unification is efficiently solvable [MM82,BS01], and an essential algorithm

*** The first two authors were supported by Spanish Min. of Educ. and Science by
the FORMALISM project (TIN2007-66023) and by the LOGICTOOLS-2 project
(TIN2007-68003-C02-01)
in areas like functional and logic programming, automated deduction, deductive databases, artificial intelligence, information retrieval, compilers, etc.

However, many of the applications in the areas mentioned above deal with large data-objects. For this reason, some kind of internal succinct representation of terms is required in order to guarantee computability in an environment with a limited amount of resources. Therefore, it is important to reconsider complexity issues for the original problem and its variants 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,Lib07], 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,Lib07], and for finding occurrences of one of them within the other (fully compressed pattern matching)[KRS95,KPR96,MST97,Lib07]. Recently, it was shown that tree grammars using multi-hole-contexts are polynomially equivalent to STGs [LMS09]. STGs have also been used for complexity analysis of unification algorithms in [LSSV06b,LSSV06a], and the matching problem [GGSS08].

In this paper, we prove that first-order unification is decidable in polynomial time even when the input is compressed using STGs. Our algorithm generates the most general unifier in polynomial time and represents it again with an STG.

1.1 Outline of the algorithm

The global structure of the algorithm is rather standard (see [Rob65]). Given two terms s and t, we look for a minimal position p where s and t differ. If s and t contain different function symbols at p, then we terminate stating that they are not unifiable. Otherwise, one of s or t, say s, contains a variable z at p. If z properly occurs in the subterm of t at p, then we terminate, again stating non-unifiability. Otherwise, we replace z by the subterm of t at p everywhere, and re-start the process again, until both s and t become equal, in which case we state unifiability.

The difficulties are induced by the task of performing all the operations mentioned above on the compressed representation of terms and positions. Positions are usually represented as sequences of integers, each one indicating the selected
child at each level from the root. Since STGs may represent terms with an exponential height, we also need to deal with compressed representations of positions in terms. In [LSSV06b,LSSV04,GGSS08], SCFGs are already used for this purpose.

We prove that all the needed operations can be performed in polynomial time with respect to the compressed representation of the terms with STG. Many of these operations can be done by an adequate use of previous results, since operations such as computing subterms, asking for the occurrence of a variable symbol in a certain subterm, or computing prefixes and suffixes of positions and contexts are known to be efficiently computable [GGSS08]. These basic operations on STGs and SCFGs are presented in Section 3.1. In [BLM05] it was shown how to succinctly represent the preorder traversal word of a word represented by an STG using an SCFG. Since we need to find a position where \( s \) and \( t \) differ, an option is to consider the first different symbol in \( s \) and \( t \) found while traversing them in preorder. To compute it, we represent the preorder traversals of \( s \) and \( t \) with words \( w_s \) and \( w_t \) compressed with an SCFG, find the first index \( i \) where \( w_s \) and \( w_t \) differ, and obtain \( p \) from \( i \). We show how to perform all of these operations efficiently in Section 3.2.

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 of the grammar and adding rules such that it 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 space increase of the grammar, since the depth of the grammar may be doubled after each of these operations. To this end, we develop a notion of restricted depth, showing that it is preserved along the execution, and that the size increase at each step can be bounded by this restricted depth, which is shown in Section 3.3. This improves upon the proof techniques used for showing polynomiality of the first-order matching algorithm on compressed terms.

2 Preliminaries

A signature is a set \( \mathcal{F} \) along with a function \( \text{ar} : \mathcal{F} \to \mathbb{N} \). Members of \( \mathcal{F} \) are called function symbols, and \( \text{ar}(f) \) is called the arity of the function symbol \( f \). Function symbols of arity 0 are called constants. Let \( \mathcal{X} \) be a set disjoint from \( \mathcal{F} \) whose elements are called variables. The set \( \mathcal{T}(\mathcal{F}, \mathcal{X}) \) of terms over \( \mathcal{F} \) and \( \mathcal{X} \) is defined to be the smallest set containing \( \mathcal{X} \) and having the property that \( f(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \) whenever \( f \in \mathcal{F}, m = \text{ar}(f) \) and \( t_1, \ldots, t_m \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \).

The size \( |t| \) of a term \( t \) is the number of occurrences of variables and function symbols in \( t \). The height of a term \( t \) is 0 if \( t \) is a constant or a variable, and \( 1 + \max\{\text{height}(t_1), \ldots, \text{height}(t_m)\} \) if \( t = f(t_1, \ldots, t_m) \). The preorder traversal of a term \( t \), denoted \( \text{pre}(t) \), is the word defined recursively by \( \text{pre}(f(t_1, \ldots, t_m)) = f \cdot \text{pre}(t_1) \cdot \ldots \cdot \text{pre}(t_m) \). Positions of a term \( t \), denoted \( p,q \), are sequences of natural numbers that are used to identify the position of subterms of \( t \). The length of a position \( p \) is denoted by \( |p| \). The set \( \text{Pos}(t) \) of positions of \( t \) is defined by
$\text{Pos}(t) = \{ \lambda \}$ if $t$ is a constant or a variable, and $\text{Pos}(t) = \{ \lambda \} \cup \{ 1 \cdot p \mid p \in \text{Pos}(t_1) \} \cup \ldots \cup \{ m \cdot p \mid p \in \text{Pos}(t_m) \}$ if $t = f(t_1, \ldots, t_m)$, where $\lambda$ denotes the empty sequence and $p \cdot q$, or simply $pq$, denotes the concatenation of $p$ and $q$.

If $t$ is a term and $p$ a position, then $t|_p$ is the subterm of $t$ at position $p$. More formally defined, $t|_\lambda = t$ and $f(t_1, \ldots, t_m)|_p = t_d|_p$. We denote by $t[s|_p$ the term that is like $t$ except that the subterm $t|_p$ is replaced by $s$. More formally defined, $t[s]|_\lambda = s$ and $f(t_1, \ldots, t_m)[s]|_p = f(t_1, \ldots, t_{i-1}, t_i[s], t_{i+1}, \ldots, t_m)$. We can define a partial order $\le$ on $\text{Pos}(t)$ by $p \le q$ if and only if $p$ is a prefix of $q$, i.e. there is a sequence $p'$ such that $q = p \cdot p'$. We say that positions $p$ and $q$ are disjoint if they are incomparable with respect to $\le$. A substitution is a mapping $\sigma : X \to T(F, X)$. Substitutions can also be applied to arbitrary terms by homomorphically extending them by $\sigma(f(t_1, \ldots, t_m)) = f(\sigma(t_1), \ldots, \sigma(t_m))$.

Intuitively, contexts are terms with a single occurrence of a hole $[\ ]$ 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 a single extra constant symbol $\setminus$. Then, if $C$ and $D$ are contexts and $s$ is a term, $CD$ and $Cs$ represent the term that is like $C$ except that the occurrence of $\setminus$ is replaced by $D$ and $s$, respectively. If $D_1 = D_2D_3$ for contexts $D_1$, 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, and denoted $\text{hp}(C)$, and its length is denoted as $|\text{hp}(C)|$.

**Definition 2.1.** A singleton context-free grammar (SCFG) $G$ is a 3-tuple $(N, \Sigma, R)$, where $N$ is a set of non-terminals, $\Sigma$ is a set of symbols, and $R$ is a set of rules of the form $N \to \alpha$ where $N \in N$ and $\alpha \in (N \cup \Sigma)^*$. The sets $N$ and $\Sigma$ must be disjoint, $|\{ N \to \alpha \in R \mid \alpha \in (N \cup \Sigma)^* \}| = 1$ for all $N \in N$, and the SCFG must be non-recursive, i.e. the transitive closure $\ge^+_G$ generated by all $N \ge_G M$ if $N \to \alpha_1 \alpha_2 M \in R$ must be terminating. The word generated by a non-terminal $N$ of $G$, denoted by $w_{G,N}$ or $w_N$ when $G$ is clear from the context, is the word in $\Sigma^*$ reached from $N$ by successive applications of the rules of $G$.

Usual definitions of SCFG require it to be in Chomsky normal form. We do not keep this restriction to ease the presentation. But note that our SCFG can be converted into the more standard ones by a linear transformation.

**Definition 2.2.** A singleton tree grammar (STG) is a 4-tuple $G = (TN, CN, \Sigma, R)$, where $TN$ is a set of tree/term non-terminals, or non-terminals of arity 0, $CN$ is a set of context non-terminals, or non-terminals of arity 1, and $\Sigma$ is a signature of function symbols (the terminals), such that the sets $TN$, $CN$, and $\Sigma$ are pairwise disjoint. The set of non-terminals $N$ is defined as $N = TN \cup CN$. The rules in $R$ may be of the form:

- $A \to f(A_1, \ldots, A_m)$, where $A, A_i \in TN$, and $f \in \Sigma$ with $\text{ar}(f) = m$.
- $A \to C_1C_2$ where $A, A_2 \in TN$, and $C_1 \in CN$.
- $C \to \setminus$ where $C \in CN$.
- $C \to C_1C_2$, where $C_1 \in CN$. 

4
\[ C \rightarrow f(A_1, \ldots, A_i, C_i, A_{i+1}, \ldots, A_m), \quad \text{where} \quad A_1, \ldots, A_i, A_{i+1}, \ldots, A_m \in TN, \ C_i, C_i \in CN, \ \text{and} \ f \in \Sigma \ \text{with} \ \text{arity}(f) = m. \]

\[ A \rightarrow A_i, \ (\lambda \text{-rule}) \ \text{where} \ A \ \text{and} \ A_i \ \text{are term non-terminals}. \]

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 \( N \) 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 \( w_G \). We will write \( w_N \) when \( G \) is clear from the context.

Note that we use \( \Sigma \) instead of \( F \) to also allow first-order variables as constants in \( \Sigma \), which are not in \( F \). \( \lambda \)-rules are not necessary for compression. However, they will be useful when applying substitutions of variables to terms represented by STGs.

**Definition 2.3.** The size \( |G| \) of an STG (SCFG) \( G \) is the sum of the sizes of its rules, where the size of a rule \( N \rightarrow \alpha \) is \( 1 + |\alpha| \). The depth within \( G \) of a non-terminal \( N \) is defined recursively as \( \text{depth}(N) := 1 + \max\{\text{depth}(N') \mid N' \text{ is a non-terminal occurring in } \alpha \text{ where } N \rightarrow \alpha \in G\} \) and the empty maximum is assumed to be 0. The depth of a grammar is the maximum of the depths of all non-terminals, denoted as \( \text{depth}(G) \).

Term dags can efficiently be represented in STGs by considering an empty set of context non-terminals, which corresponds to the commonly used implementation of dags by adjacency lists. However, STG-represented terms may have exponential depth in the size of the grammar in contrast to dags, which only allow for a linear depth in the (notational) size of the dags.

**Example 2.4.** Let \( G = \{ \{ A, A_1 \}, \{ C', C_0, C_1, C_2, \ldots, C_{n-1}, C_n \}, \{ f, a, x \}, R \} \), where \( R = \{ A \rightarrow C_0 A_1 A_1 \rightarrow x, C_0 \rightarrow f(C'), C' \rightarrow [ ], C_1 \rightarrow C_0 C_0, C_2 \rightarrow C_1 C_1, \ldots, C_n \rightarrow C_{n-1} C_n \} \).

\[ \text{depth}(G) = \text{depth}(A) = n + 3. \]

The term non-terminal \( A \) of \( G \) represents the term \( w_{G,A} = f^n(x) \), whose height and size are exponential with respect to \( |G| \).

Plandowski [Pla95] proved decidability in polynomial time of the word problem for SCFG, i.e., given an SCFG \( P \) and two non-terminals \( A \) and \( B \) of \( P \), to decide whether \( w_{P,A} = w_{P,B} \). The best complexity for this problem has been recently obtained by Lifshits [Li97] with time \( O(|P|^4) \). 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 using the Lifshits result. Hence, we have the following.

**Theorem 2.5.** ([Li97, BLM05]) Given an STG \( G \), and two tree non-terminals \( A, B \) from \( G \), it is decidable in time \( O(|G|^4) \) whether \( w_A = w_B \).
We will use more specific information from Lifshits’ work [Lif7].

**Lemma 2.6.** [Lif7] Let \( G \) be an SCFG. Then a data structure can be computed in time \( O(|G|^3) \) which allows to answer to the following question in time \( O(|G|) \): 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 \( k \)?

The unification problem is a generalization of the word problem, since it allows occurrences of variables and substituting them in order to satisfy equality.

**Definition 2.7.** 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.

**Example 2.8.** Let \( G = (\{A_1, A_s, A, B, A', B'\},\{C_0, C_1, C_2, C_3, C_4, C', D\}, \{g,f,a,x\}, R) \), where \( R = \{A_1 \rightarrow g(B,A), A_s \rightarrow g(A,A), A \rightarrow C_4[A'], C_4 \rightarrow C_3C_3, C_3 \rightarrow C_2C_2, C_2 \rightarrow C_1C_1, C_1 \rightarrow C_0C_0, C_0 \rightarrow f(C'), C' \rightarrow [], A' \rightarrow a B \rightarrow D[B'], D \rightarrow C_3C_2, B' \rightarrow x\} \), be an STG. Note that \( w_{G,A_s} = g(f^{12}(x), f^{16}(a)) \), and \( w_{G,A_t} = g(f^{16}(a), f^{16}(a)) \). Hence, \( (G, A_s, A_t) \) is an instance of first-order unification with STG. The goal is to find a substitution \( \sigma \) such that \( \sigma(w_{G,A_s}) = \sigma(w_{G,A_t}) \).

# 3 Basic operations with STG and SCFG

Usual term unification algorithms need to compute subterms, apply substitutions, look for the difference between two terms, look for the occurrences of a certain variable, etc. We need to perform these operations when the input terms are represented by an STG. We use SCFGs for concisely representing positions of a term represented with an STG. For clarity we call the non-terminals of this SCFG *position non-terminals*.

## 3.1 Known Results

We give a list of operations which are known to be computable in linear time for a given STG \( G \) generating terms, and an SCFG \( P \) generating positions.

- For every position non-terminal \( p \) of \( P \) and non-terminal \( N \) of \( G \), the numbers \( |w_p| \) and \( |w_N| \) are computable in time \( O(|P|) \) and \( O(|G|) \), respectively.
- An SCFG \( H_G \) can be computed in time \( O(|G|) \) for \( G \) such that, for every context non-terminal \( C \in G \), there exists a position non-terminal \( H_C \) of \( H_G \) which generates \( hp(w_C) \). Moreover, the depth of every \( H_C \) is the same as the one of its corresponding \( C \), and \( \text{depth}(H_G) \leq \text{depth}(G) \).
- Given a position non-terminal \( p \) of \( P \) and a number \( l \leq |w_p| \), the SCFG \( P \) can be extended in time \( O(|P|) \) with depth(\( p \)) new non-terminals such that one of them, called \( p' \) generates the prefix (suffix) of \( w_p \) of length \( l \). Moreover, \( \text{depth}(p') \leq \text{depth}(p) \), and the new SCFG \( P' \) satisfies \( \text{depth}(P') = \text{depth}(P) \).

We present in detail the case of the extension of \( G \) generating a certain suffix of \( w_C \), for a context non-terminal \( C \) of \( G \) given in [GGSS08], as a definition here.

**Definition 3.1.** Let \( G \) be an STG and let \( C \) be a context non-terminal of \( G \). Let \( l \) be a natural number such that \( l \leq \text{hp}(w_C) \). Then, we define \( \text{Suff}(G,C,l) \) as an extension of \( G \) recursively as follows.

- If \( l = 0 \), then \( \text{Suff}(G,C,l) := G \). In the next cases we assume \( l > 0 \).
- If \( (C \to C_1C_2) \in G \) and \( l < \text{hp}(w_{C_1}) \). Then \( \text{Suff}(G,C,l) \) includes \( \text{Suff}(G,C_1,l) \), which contains a non-terminal \( C'_1 \) generating the suffix of \( w_{C_1} \) with \( \text{hp}(w_{C'_1}) = \text{hp}(w_{C_1}) - l \), plus the rule \( C' \to C'_1C_2 \), where \( C' \) is an additional new non-terminal.
- If \( (C \to C_1C_2) \in G \) and \( l \geq \text{hp}(w_{C_1}) \), then, with \( l' := l - \text{hp}(w_{C_1}) \), we define \( \text{Suff}(G,C,l) \) as \( \text{Suff}(G,C_1,l') \).
- If \( (C \to f(A_1,\ldots,A_{i-1},C_1,A_{i+1},\ldots,A_m)) \in G \), then we define \( \text{Suff}(G,C,l) \) as \( \text{Suff}(G,C_i,l-1) \).
- In any other case \( \text{Suff}(G,C,l) \) is undefined.

**Lemma 3.2.** Let \( G \) be an STG describing first-order terms and contexts. Let \( C \) be a context non-terminal of \( G \), and let \( l \) be a natural number such that \( l \leq \text{hp}(w_C) \).

Then, \( G' = \text{Suff}(G,C,l) \) is computable in time \( O(|G|) \), it adds at most \( \text{depth}(C) \) new context non-terminals, and one context non-terminal \( C' \) of \( G' \) generates the suffix of \( w_C \) with \( \text{hp}(w_{C'}) = \text{hp}(w_C) - l \). Moreover, for every new non-terminal \( N \), \( \text{depth}_{C'}(N) \leq \text{depth}_C(C) \), and \( \text{depth}(G') = \text{depth}(G) \).

Extending \( G \) to generate \( w_N \), for a non-terminal \( N \) of \( G \) and a non-terminal position \( p \) of \( P \), is known to be computable in polynomial time. This was shown in [GGSS08]. We present such an extension as a definition here.

**Definition 3.3.** Let \( G \) be an STG describing first-order terms and contexts, and let \( P \) be an SCFG describing positions. Let \( p \) be a position non-terminal of \( P \) and \( N \) a non-terminal of \( G \). We recursively define \( \text{pExt}(G,N,p,P) \) as an extension of \( G \) as follows.

- If \( w_p = \lambda \) (the empty word), then \( \text{pExt}(G,N,p,P) := G \). In the next cases we assume \( w_p \neq \lambda \).
- If \( (N \to C_1N_2) \in G \) and \( w_p < \text{hp}(w_{C_1}) \), then \( \text{pExt}(G,N,p,P) \) includes \( \text{Suff}(G,C_1,[w_p]) \), which contains a non-terminal \( C'_1 \) generating the suffix of \( w_{C_1} \) with \( \text{hp}(w_{C'_1}) = \text{hp}(w_{C_1}) - |w_p| \), plus the rule \( N' \to C'_1N_2 \), where \( N' \) is an additional new non-terminal.
- If \((N \rightarrow C, N_0) \in G\) and \(w_p\) is disjoint from \(hp(w_{C_1})\), then 
  \(\text{pExt}(G, N, p, P) := \text{pExt}(G, C_1, p, P)\).
- If \((N \rightarrow C, N_0) \in G\) and \(hp(w_{C_1}) \leq w_p\), then extend \(P\) with \(\text{depth}(p)\) new non-terminals where one of them called \(p'\) generates the suffix of \(w_p\) of length \(|w_p| - |hp(w_{C_1})|\), and define 
  \(\text{pExt}(G, N, p, P) := \text{pExt}(G, N_0, p', P)\).
- If \((N \rightarrow f(N_1, \ldots, N_m)) \in G\), and \(i \leq w_p\) with \(1 \leq i \leq m\), then extend \(P\) with \(\text{depth}(p)\) new non-terminals where one of them called \(p'\) generates the suffix of \(w_p\) of length \(|p| - 1\), and define 
  \(\text{pExt}(G, N, p, P) := \text{pExt}(G, N_i, p', P)\).
- In any other case \(\text{pExt}(G, N, p, P)\) is undefined.

**Lemma 3.4.** Let \(G\) be an STG describing first-order terms and contexts, and \(P\) be an SCFG describing positions. Let \(p\) be a position non-terminal of \(P\), and \(N\) a non-terminal of \(G\). It can be checked in time \(O((|G| + |P|)^4)\) if \(w_p\) is a valid position of \(w_N\). Moreover, \(G' = \text{pExt}(G, N, p, P)\) is computable in time \(O((|G| + |P|)^4)\), it adds at most \(\text{depth}(N)\) new non-terminals, and one non-terminal of \(G'\) generates \(w_N|w_p\). Furthermore, for every new non-terminal \(N'\), \(\text{depth}_{G'}(N') \leq \text{depth}_{G}(N)\), and \(\text{depth}(G') = \text{depth}(G)\).

### 3.2 Finding the first different position of two terms

Given two terms \(s\) and \(t\), represented by term non-terminals \(A_s\) and \(A_t\) of an STG \(G\), we show how to efficiently construct a succinct SCFG \(P\) with a position non-terminal \(p\) generating the word \(w_p\), which represents the first different position of \(s\) and \(t\), i.e. the first one with a different root symbol found when we traverse them in preorder. Recall that such a word is a position in \(s\) and \(t\), thus a sequence of integers. The SCFG \(P\) is obtained in three steps detailed in the next three subsections. We first construct an SCFG \(\text{Pre}_G\) with non-terminals \(P_s\) and \(P_t\) generating the preorder traversals \(\text{pre}(s)\) and \(\text{pre}(t)\) of \(s\) and \(t\), respectively. This is based on the ideas of [BLM05]. Then, given \(P_s\) and \(P_t\), we describe a procedure to efficiently compute the first index \(k\) in which \(\text{pre}(s)\) and \(\text{pre}(t)\) differ. Finally, given the index \(k\) we show how to construct the desired SCFG \(P\).

**Computing the preorder traversal of a term.** 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 \([1, \ldots, |\text{pre}(t)|]\) of \(\text{pre}(t)\) and the positions \(\text{Pos}(t)\) of \(t\), which associates every position \(p \in \text{Pos}(t)\) to the index \(i \in [1, \ldots, |\text{pre}(t)|]\) you find at \(\text{root}(t)[p]\) while traversing the tree in preorder. We can recursively define the two mappings \(\text{pIndex}(t, p) \rightarrow [1, \ldots, |\text{pre}(t)|]\) and \(\text{iPos}(t, i) \rightarrow \text{Pos}(t)\) as follows. \(\text{pIndex}(t, \lambda) = 1\), \(\text{pIndex}(f(t_1, \ldots, t_m), i, p) = (1 + |t_1| + \ldots + |t_{i-1}|) + \text{pIndex}(t_i, p)\), \(\text{iPos}(t, 1) = \lambda\), and \(\text{iPos}(f(t_1, \ldots, t_m), 1 + |t_1| + \ldots + |t_{i-1}| + k) = i.\text{Pos}(t_i, k)\) for \(1 \leq k \leq |t_i|\).

In [BLM05] it is shown how to construct, from a given STG \(G\), an SCFG \(\text{Pre}_G\) representing the preorder traversals of the terms generated by \(G\). We reproduce
that construction here, presented in Figure 1 as a set of rules indicating, for
each term non-terminal \( A \) and its rule \( A \to \alpha \) of \( G \), which rule \( P_A \to \alpha' \) of \( \mathcal{P}_G \)
is required in order to make the non-terminal \( P_A \) of \( \mathcal{P}_G \) satisfy \( w_{\mathcal{P}_G}: \alpha' = \mathcal{P}(w_{[\alpha]}). \) To this end, for each context non-terminal \( C \) of \( G \) we also need non-terminals of \( \mathcal{P}_G \) generating the preorder traversal to the left of the hole \( (L_C) \),
and the preorder traversal to the right of the hole \( (R_C) \).

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 \( P_N \)
of \( \mathcal{P}_G \) generates \( \mathcal{P}(w_N) \).

**Example 3.5.** (Continuation of Example 2.8) The SCFG \( \mathcal{P}_G \) obtained by applying
the rules of Figure 1 to the STG \( G \) of Example 2.8 is \( \{ \mathcal{P}_A, \to \}
\begin{align*}
gP_BP_A,PA, & \to gP_APA,PA \to LC_1PA,RC_1PA,PA \to \alpha_B,PA \to LC_2PA,RC_2PA \to R_CPA \to R_CP_A \to RC_2PR_{C_1} \to P_{C_1} \to P_{C_2} \to P_{C_3} \to P_{C_4} \to P_{C_N} \to \alpha_B \to RC_1PR_{C_1} \to R_{C_1} \to R_{C_2} \to R_{C_3} \to R_{C_4} \to RC_1R_{C_1} \to RC_2R_{C_2} \to RC_3R_{C_3} \to RC_4R_{C_4} \to \alpha_B \}
\end{align*}
Note that \( w_{\mathcal{P}_A}, = g^{12}f^4j^{16}a \) and \( w_{\mathcal{P}_A}, = g^{16}f^{16}a \).

**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 first position \( k \) where \( w_{p_1} \) and \( w_{p_2} \) are different. In order to solve this problem, a linear search over the
generated words is not a good idea, since their sizes may be exponentially big with respect to the size of \( P \). But we can take advantage from Lemma 2.6 to make it faster. Thus, assume that the pre-computation of Lemma 2.6 has been done (in time \( O(|P|^3) \)) and hence we can answer whether a given \( w_{p_1} \) occurs in a
given \( w_{p_2} \) at a certain position in time \( O(|P|) \).

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
\[(N \to \alpha) \land (k = 1) \quad P_{N,k} \rightarrow \lambda\]

\[(N \to f(N_1,\ldots,N_m)) \land (1 + |w_{N_1}| + \ldots + |w_{N_{m-1}}| = k' > k \leq |w_{N_1}|) \quad P_{N,k} \rightarrow \iota P_{N_1,k-\kappa'}\]

\[(N \to C_1 N_2) \land (1 < k \leq |w_{LC_1}|) \quad P_{N,k} \rightarrow P_{C_1,k}\]

\[(N \to C_1 N_2) \land (k' = |w_{LC_1}| + |w_{N_2}|) \quad P_{N,k} \rightarrow H_{C_1} P_{N_2,k-\kappa'}\]

\[(N \to C_1 N_2) \land (|w_{LC_1}| + |w_{N_2}| < k) \land (k' = |w_{N_1}|) \quad P_{N,k} \rightarrow P_{C_1,k-\kappa'+1}\]

**Fig. 3.** Construction of the SCFG generating the position corresponding to the k-th index in pre(\(w_N\)).

\(w_n[1\ldots|w_p|]\) (with \(w[i]\) we denote the symbol occurring at position \(i\) in the word \(w\), and with \(w[i\ldots j]\) we denote the subword of \(w\) at position \(i\) and length \(j - i + 1\)). 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 \(w_{p_1} \neq w_{p_2}[(k' + 1)\ldots(k' + |w_{p_1}|)]\), 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 2, as can be shown inductively on the depth of \(p_1\). By Lemma 2.6, each call takes time \(O(|P|)\), and at most \(depth(P)\) 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 2.6, that is, \(O(|P|^3)\).

**Lemma 3.6.** 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 \(O(|P|^3)\).

**Example 3.7.** (Continuation of Example 3.5) The SCFG \(P_{SCG}\) is not in Chomsky Normal Form, but it is easy to adapt the algorithm of Figure 2 to this case. Thus, if we execute an adapted version of \(\text{index}(P_{A_1}, P_{A_1}, 0, \text{pre}_{G})\), the following sequence of calls is produced: \(\text{index}(P_{A_1}, P_{A_1}, 0, \text{pre}_{G})\), \(\text{index}(P_{B_1}, P_{A_1}, 1, \text{pre}_{G})\), \(\text{index}(P_{B_1}, P_{A_1}, 13, \text{pre}_{G})\). The third call returns 1, the second one returns 13, and the first one returns 14, which corresponds to the first different position of \(w_{A_1}\) and \(w_{A_1}\).

**Computing the first different position of two terms.** Using the index \(k\) from the previous subsection, we want to compute the SCFG \(P\) with a non-
terminal $p$ generating the word $w_p$, which represents the first different position of $s$ and $t$. Generalizing this, for a given non-terminal $N$ of $G$ and a given positive natural number $k$, we want to construct an SCFG with a non-terminal $P_{N,k}$ generating the position of the symbol corresponding to the $k$-th index of $w_N$. In particular, $P_{A_t,k}$ is the $p$ we are looking for. We show how to do that again with a set of inference rules in Figure 3. These inference rules have to be understood to act on demand, i.e. they are executed to generate new non-terminals if those non-terminals have been demanded by other inference rules, or correspond to the initially demanded $P_{A_t,k}$. Note that we make use here of the SCFG $H_G$, which can be constructed in linear time as commented in Section 3.1.

It is again straightforward to check, by induction on $\text{depth}(N)$, that every $P_{N,k}$ generates $\text{IPos}(w_N,k)$. Note that the grammar rule of every such $P_{N,k}$ demands the existence of a grammar rule for at most one new $P_{N',k'}$, and the corresponding $N'$ satisfies $\text{depth}(N') < \text{depth}(N)$. Therefore, at most $\text{depth}(G)$ of the inference rules are executed for constructing $P_{A,t,k}$.

The following lemma is a consequence of the three previous subsections.

**Lemma 3.8.** Let $G$ be an STG, and let $A_s$ and $A_t$ be term non-terminals of $G$ such that $w_{A_s} \neq w_{A_t}$. Then, an SCFG $P$ with a position non-terminal $p$ generating the first different position in $w_{A_s}$ and $w_{A_t}$, can be computed in time $O(|G|^3)$. Moreover, $|P| \leq |G|$ and $\text{depth}(P) \leq \text{depth}(G)$.

**Example 3.9.** (Continuation of Example 3.7) We compute now an SCFG $P$ with a position non-terminal $P_{A_{t,14}}$ generating the position in $w_{A_t}$ that corresponds to the 14th index in its preorder traversal. We use the fact that the SCFG $H_G$ presented in Section 3.1, can be constructed in linear time. The set of rules of $H_G$ is

\[
\{H_{C_1} \rightarrow H_{C_5}, H_{C_3}, H_{C_2} \rightarrow H_{C_2} H_{C_6}, H_{C_3} \rightarrow H_{C_3} H_{C_4}, H_{C_4}, H_{C_1}, \lambda \rightarrow 1H_{C'}, H_{C'}, \lambda \rightarrow H_{C_6}, H_{C_7}\}. \]

We construct $P$ using the inference system presented in this section with the STG $G$, $N = A_t$, and $k = 14$. The set of rules of $P$ is

\[
\{P_{A_{t,14}} \rightarrow 1P_{A_{13}}, P_{A_{13}} \rightarrow P_{C_{13}}, P_{C_{13}} \rightarrow H_{C_5} P_{C_8}, P_{C_8}, P_{C_5}, P_{C_2} \rightarrow H_{C_6} P_{C_{2,1}}, P_{C_{2,1}} \rightarrow \lambda\}. \]

Note that $w_{P_{A_{t,14}}} = 11^{14}1^4 \lambda = 1^{13}$.

### 3.3 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 replaced value. We define this notion of application of a substitution as follows.

**Definition 3.10.** 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 $X \rightarrow A$ to $G$, and converting $X$ into a term non-terminal.
Example 3.11. (Continuation of Example 3.9) We now apply three operations to the STG $G$ given as input. We first extend $G$ using the $\text{pExt}$ construction presented in Definition 3.3 such that a new non-terminal, called $A'_x$, generates $w_{A_1} \cdot w_{A_{x+4}}$. Then, we need to check that the variable $x$ does not occur in $w_{A'_x}$, which can be done in linear time. Finally, we perform the substitution $\{x \mapsto A'_x\} (G)$ as stated in Definition 3.10. The set of rules of the obtained grammar $G'$ after the $\text{pExt}$ construction and this assignment is $\{x \mapsto A'_x, A'_x \mapsto C_2 A', A_x \mapsto g(B, A), A_x \rightarrow g(A, A), A \rightarrow C_4[A'], A_4 \rightarrow C_3 C_3, C_3 \rightarrow C_2 C_2, C_2 \rightarrow C_1 C_1, C_1 \rightarrow C_0 C_0, C_0 \rightarrow f(C'), C' \rightarrow \_\}, A' \mapsto a B \rightarrow D[B'], D \rightarrow C_3 C_2, B' \rightarrow x\}$. The rules marked with bold correspond to the added non-terminals with respect to the initial STG $G$. Note that $w_{G', A'_x} = w_{G, A_x}|_{w_{p,p_{A_x}}} = w_{G, A_x}|_{113} = f^4(a)$, and thus, $w_{G', A'_x} = g(f^{12}(w_{G', A'_x}), f^{16}(a)) = g(f^{12}(w_{G, A'_x}), f^{16}(a)) = g(f^{12} f^4(a), f^{16}(a)) = g(f^{16}(a), f^{16}(a)) = w_{G', A'_x}$. Hence, we state unifiability.

The solution $\sigma$ is represented in the STG $G'$.

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 $\text{vDepth}$, which does not increase after an application of a substitution. It allows us to bound the final size increase of $G$. The notion of $\text{vDepth}$ 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 3.12. Let $G = (T,N,C,N', \Sigma, R)$ be an STG, and let $V$ be a subset of $T \cup N \cup \Sigma$. We say that $V$ is a $\lambda$-set for $G$ if for each term non-terminal $A$ in $V$, the rule of $G$ of the form $A \rightarrow a$ is a $\lambda$-rule.

Definition 3.13. Let $G = (T,N,C,N', \Sigma, R)$ be an STG and let $V$ be a $\lambda$-set for $G$. For every non-terminal $N$ of $G$, the value $\text{vDepth}_{G,V}(N)$, denoted also as $\text{vDepth}_{V}(N)$ or $\text{vDepth}(N)$ when $G$ and/or $V$ are clear from the context, is defined as follows (recall the convention that $\max(\emptyset) = 0$).

$\text{vDepth}(N) := 0$ for $N \in V$
$\text{vDepth}(N) := 1 + \max\{\text{vDepth}(N') \mid N'$ is a non-terminal occurring in $\alpha$, where $N \rightarrow \alpha \in G\}$, otherwise.

The $\text{vDepth}$ of $G$ is the maximum of the $\text{vDepth}$ 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 $\text{vDepth}$ provided $X \in V$ for the substitution $X \mapsto A$.

Lemma 3.14. 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 $\text{vDepth}_{G'}(N) = \text{vDepth}_{G}(N)$.

We also need the fact that $\text{vDepth}$ does not increase due to the construction of $\text{pExt}(G, A, p, P)$ from $G$. This is stated by the following two lemmas.
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 $p$ such that $\text{root}(s|_p) \neq \text{root}(t|_p)$.
If both $\text{root}(s|_p)$ and $\text{root}(t|_p)$ are function symbols; Then
Halt stating that the initial $s$ and $t$ are not unifiable
Here, either $s|_p$ or $t|_p$ is a variable $x$, say $s|_p$, and both are different.
If $x$ occurs in $t|_p$ Then
Halt stating that the initial $s$ and $t$ are not unifiable
Extend the compressed representation by the assignment $\{x \mapsto t|_p\}$
EndWhile
Halt stating that the initial $s$ and $t$ are unifiable

Fig. 4. Unification Algorithm of STG-Compressed Terms

Lemma 3.15. Let $G$ be an STG, let $C$ be a context non-terminal of $G$, let $V$ be a set of terminals and term non-terminals of $G$, let $l$ be a natural number smaller than $|\text{hpt}(w_C)|$, and let $G'$ be Suff($G, C, l$).
Then, for every non-terminal $N$ of $G$ it holds that $\text{Vdepth}_{C'}(N) = \text{Vdepth}_{C}(N)$, and for every new non-terminal $N$ in $G'$ and not in $G$, it holds that $\text{Vdepth}_{C'}(N) \leq \text{Vdepth}_{C}(C)$. Moreover, the number of new added non-terminals is bounded by $\text{Vdepth}_{C}(N)$. 

Lemma 3.16. Let $G$ be an STG and $P$ an SCFG, let $N$ be a non-terminal of $G$, let $V$ be a $\lambda$-set for $G$, let $p$ be a position non-terminal of $P$ such that $w_p \in \text{Pos}(w_N)$, and let $G'$ be pExt($G, N, p, P$).
Then, for every non-terminal $N'$ of $G$ it holds that $\text{Vdepth}_{C'}(N') = \text{Vdepth}_{C}(N')$, and for every new non-terminal $N''$ in $G'$ and not in $G$, it holds that $\text{Vdepth}_{C'}(N'') \leq \text{Vdepth}_{C}(N)$. Moreover, the number of new added non-terminals is bounded by $\text{Vdepth}_{C}(N)$.

4 A polynomial time algorithm for first-order unification with STG

From a high level perspective the structure of our algorithm given in Figure 4 is very simple and rather standard. Most algorithms for first-order unification are variants of the above 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. In the previous section we have already seen how to perform the basic required operations on STGs: look for the first position $p$ satisfying that $\text{root}(s|_p)$ and $\text{root}(t|_p)$ are different, construct the term $t|_p$, 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 \( \text{depth}(G) \) 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 \( V_{\text{depth}}(G) = n \). The value \( V_{\text{depth}}(G) \) is preserved to be \( n \) along the execution of the algorithm thanks to Lemmas 3.14 and 3.16. Moreover, by Lemma 3.16, 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 \( O(|G|^2) \). Thus we have proved:

**Theorem 4.1.** First-order unification of two terms represented by an STG can be done in polynomial time \( O(|V|(m + |V|n)^4) \), 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.

5 Conclusion and Further Research

We presented an instantiation-based first-order unification algorithm, that can be immediately executed on the compressed representation of large terms and runs in polynomial time on the size of the representation.

Further research is to investigate extensions of first-order unification on compressed terms, and to investigate optimizations. Perhaps it is possible to show an improved upper bound. We believe 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 [GGSST08].

**Acknowledgement.** We thank Markus Lohrey for valuable discussions on the subject of this paper.

**References**


