Congruence Closure of Compressed Terms in Polynomial Time

Manfred Schmidt-Schauss[1] and David Sabel[1] and Altug Anis[1]

Dept. Informatik und Mathematik, Inst. Informatik, Goethe-University,
PoBox 11 19 32, D-60054 Frankfurt, Germany,
{schauss,sabel,altug}@ki.informatik.uni-frankfurt.de

Abstract. The word-problem for a finite set of equational axioms between ground terms is the question whether for terms s, t the equation $s = t$ is a consequence. We consider this problem under grammar based compression of terms, in particular compression with singleton tree grammars (STGs) and with directed acyclic graphs (DAGs) as a special case. We show that given a DAG-compressed ground and reduced term rewriting system T, the T-normal form of an STG-compressed term s can be computed in polynomial time, and hence the T-word problem can be solved in polynomial time. This implies that the word problem of STG-compressed terms w.r.t. a set of DAG-compressed ground equations can be decided in polynomial time. If the ground term rewriting system (gTRS) T is STG-compressed, we show NP-hardness of T-normal-form computation. For compressed, reduced gTRSs we show a PSPACE upper bound on the complexity of the normal form computation of STG-compressed terms. Also special cases are considered and a prototypical implementation is presented.

Keywords: term rewriting, grammar based compression, singleton tree grammars, congruence closure

1 Introduction

This paper is dedicated to combining equational reasoning with grammar compression for terms. Automated deduction systems, formalizations of logical systems, systems for checking propositional logic and term rewriting systems [3,6] either are based on equational reasoning or may employ equational reasoning. The general form of equational reasoning using unrestricted sets of equational axioms is known to be very expressive, but to the price of undecidability of simple questions about derivability. A special case that leads to a decidable word problem occurs when all equational axioms are ground and thus quantifiers do not play any role. Congruence closure algorithms can solve this special kind of word problem in time $O(n \log n)$ ([20,26,13]). Extending SAT-solvers by theories leads to so-called SMT (SAT modulo theories) [21], which among other theories can also deal with equational theories defined by a set of ground equations.

Since terms in automated deduction systems may grow large during reasoning and search for a proof, compact or compressed representations of large

terms can be exploited to optimize the space usage, which must go hand-in-hand with specific algorithms that access the compact representations and process the compressed representation without too much decompression.

Instead of using specialized compression formats (e.g. Lempel-Ziv [29]), there are also investigations into a general mechanism: straight-line programs (SLP) for strings [23] and corresponding algorithms. An SLP here means an acyclic context free grammar, where every nonterminal has one rule, and thus can generate exactly one string. The generalization to terms is for example in [4,5] to compress XML-trees. The grammars for compressing terms were called *singleton tree grammars* (STG) [14]. STGs generalize directed acyclic graphs, since they are not limited to sharing subterms but they can also share contexts – terms with a single hole – and thus they allow to share parts of terms. For instance the grammar $A ::= C_3[B], B ::= b, C_0 ::= f([\cdot]), C_1 ::= C_0[C_0], C_2 ::= C_1[C_1], C_3 ::= C_2[C_2]$ is an STG where the nonterminal A generates the term $f(f(f(f(f(f(f(f(b)))))))))$.

Grammar-compression was also used for analyzing the complexity of unification algorithms: SLPs in [14,15] and STGs in [16,10,11]. A more general form of compression, employing terms with several holes, was shown to be polynomially equivalent to STGs [19].

A key algorithm is the equality check of two compressed strings (trees), which can be done in cubic time [17,22]. Almost all efficient algorithms on STGs use variants of the equality check of compressed words/terms. Implementations of related algorithms are described in [12].

Complexity results w.r.t. the compressed word and membership problem of confluent semi-Thue systems can be found in [18]. Note that ground TRSs on strings are like restricted semi-Thue systems, where the reduction relation is only applicable to the suffix of words.

The main result of this paper is an efficient normalization and thus an efficient solution of the word problem of STG-compressed terms w.r.t. DAG-compressed ground equations (Theorem 14). If the axioms are STG-compressed, then only partial results are obtained: For a non-confluent compressed gTRS, the question whether a term s reduces to a constant a is NP-hard (Proposition 17), and for reduced, confluent and STG-compressed gTRSs the normal form computation of STG-compressed terms is in PSPACE (Proposition 15). Also several special cases like one-rule gTRSs and monadic gTRSs are considered in the remainder of Section 4. Finally, in Section 5 a prototypical implementation, some examples, and experimental results are presented.

2 Preliminaries

In this section we briefly recall required notions and results on term rewriting systems and singleton tree grammars.

2.1 Term Rewriting Systems

A *signature* Σ is a set of function symbols, where every $f \in \Sigma$ has a fixed arity $\mathrm{ar}(f) \in \mathbb{N}_0$. Let $\mathcal{V}$ be a countably infinite set of variables. The set of *terms*

$\mathcal{T}(\Sigma,\mathcal{V})$ over the signature Σ and variables $\mathcal{V}$ is inductively defined as follows: for all $x \in \mathcal{V} : x \in \mathcal{T}(\Sigma,\mathcal{V})$; if $f \in \Sigma$ and $\mathrm{ar}(f) = 0$ then $f \in \mathcal{T}(\Sigma,\mathcal{V})$; and if $t_i \in \mathcal{T}(\Sigma,\mathcal{V})$, for $i = 1,\ldots,n$, $f \in \Sigma$, and $\mathrm{ar}(f) = n \geq 1$, then $f(t_1,\ldots,t_n) \in \mathcal{T}(\Sigma,\mathcal{V})$. With $Var(t)$ we denote the *set of variables* occurring in term t. A term $t \in \mathcal{T}(\Sigma,\mathcal{V})$ is called *ground* if $Var(t) = \emptyset$. A *substitution* σ is a mapping of variables to terms. Let $\mathrm{dom}(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. The extension σ_E of σ to terms is inductively defined as $\sigma_E(x) = x$ if $x \notin \mathrm{dom}(\sigma)$, $\sigma_E(x) = \sigma(x)$ if $x \in \mathrm{dom}(\sigma)$, $\sigma_E(f(t_1,\ldots,t_n)) = f(\sigma_E(t_1),\ldots,\sigma_E(t_n))$. In the following we do not distinguish between a substitution and its extension to terms. A *context* C is a term where the special constant $[\cdot]$ (the "hole") occurs exactly once (as a subterm). The term $C[t]$ is constructed by replacing the hole in C by term t. A context C_1 is a *prefix* of context C_2 if there exists a context C_3 such that $C_1[C_3] = C_2$.

A *term rewriting system* (TRS) T is a finite set of pairs of terms $\{(l_i, r_i) \mid i = 1,\ldots,m\}$, usually written $l_i \to r_i$, where we assume that for all i: l_i is not a variable and $Var(r_i) \subseteq Var(l_i)$ (see e.g. [3]). The term rewriting relation $\xrightarrow{T}$ is defined as: $t \xrightarrow{T} t'$, if $t = C[\sigma(l_i)]$ and $t' = C[\sigma(r_i)]$ for some i, some substitution σ, and some C. The transitive and reflexive-transitive closures of $\xrightarrow{T}$ are written as $\xrightarrow{T,+}$ and $\xrightarrow{T,*}$, respectively. A term t is *T-irreducible* or a *T-normal form*, iff it cannot be further reduced using the rules of T. If the TRS is interpreted as a set of equations $E := \{l_i = r_i \mid l_i \to r_i \in T\}$, then the equality $=_E$ is the equational theory on the terms w.r.t. a signature Σ. Operationally one can define $s =_E t$ iff $s \xrightarrow{E,*} t$ by permitting the equational axioms as rewrite rules in both directions. Alternatively, $=_E$ can be defined as the smallest congruence relation with $\sigma(l_i) =_E \sigma(r_i)$ for all substitutions σ and all $i = 1,\ldots,n$. The *word problem* is to decide for given terms s, t, whether $s =_E t$. A TRS is called *terminating*, if there are no infinite reduction sequences of $\xrightarrow{T}$, and it is called *confluent* iff whenever $t_1 \xleftarrow{T,*} t \xrightarrow{T,*} t_2$, there exists a term t_3 with $t_1 \xrightarrow{T,*} t_3 \xleftarrow{T,*} t_2$. A TRS that is confluent and terminating is also called *canonical*. Canonical TRSs permit to compute unique *normal forms* of terms by rewriting them exhaustively. For a canonical TRS T the word problem is decidable by rewriting the terms s, t to their normal form and then comparing the normal forms for syntactic equality.

In this paper we are interested in ground equations and ground term rewriting systems (gTRS), i.e. when the equations in E (the rules in T, respectively) consist of ground terms. A term rewriting system T is *reduced* if every right hand side r_i is a T-normal form, and every left hand side l_i is irreducible for the system $T \setminus \{l_i \to r_i\}$. It is well-known that every reduced gTRS is canonical. It is also well-known that the word problem for ground equations E is decidable [20,26,13]. The algorithms are usually variants of the so-called congruence closure computation on term graphs. This can be computed in time $O(n \log n)$, where it is essential that DAGs are used.

In [27,28,8] it is shown that the computation of a DAG-representation of a canonical gTRS can be done in time $O(n \log n)$. The computed canonical gTRS

$T = \{l_i \to r_i \mid i = 1, \ldots, n\}$ has the additional property that it is reduced and thus T is canonical.

2.2 Grammar Compressed Terms and Term Rewriting Systems

For compression of (ground) terms we use singleton tree grammars:

Definition 1 ([14]). *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$, where $\mathcal{TN}$ are tree/term nonterminals, or nonterminals of arity 0, $\mathcal{CN}$ are context nonterminals, or nonterminals 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 nonterminals $\mathcal{N}$ is defined as $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. The rules in R may be of the form:*

- $A ::= f(A_1, \ldots, A_m)$, *where $A, A_i \in \mathcal{TN}$ for $i = 1, \ldots, m$, and $f \in \Sigma$ with* $\mathrm{ar}(f) = m$.
- $A ::= C_1[A_2]$ *where $A, A_2 \in \mathcal{TN}$, and $C_1 \in \mathcal{CN}$.*
- $C ::= [\cdot]$ *where $C \in \mathcal{CN}$.*
- $C ::= C_1[C_2]$, *where $C, C_1, C_2 \in \mathcal{CN}$.*
- $C ::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_m)$, *where $C \in \mathcal{CN}$, $A_j \in \mathcal{TN}$ for $j = 1, \ldots, i-1, i+1, \ldots m$, and $f \in \Sigma$ with* $\mathrm{ar}(f) = m$.
- $A ::= A'$, *where A and A' are term nonterminals*

Each nonterminal X appears as a left-hand side of exactly one rule of R. The transitive closure $\xrightarrow{+}_G$ of the relation $\to_G$ over $\mathcal{N}$ is terminating, where $X \to_G Y$, iff $X ::= r$ is a rule in G, and $Y \in \mathcal{N}$ occurs in r. The term (or context) generated by a nonterminal N of G, denoted by $\mathrm{val}_G(N)$ *or* $\mathrm{val}(N)$ *when G is clear from the context, is the term (or context) over Σ reached from N by successive and exhaustive applications of the rules of G. More rigorously:*

$$
\begin{array}{ll}
\mathrm{val}_G(A) = f(\mathrm{val}_G(A_1), \ldots, \mathrm{val}_G(A_m)), & \text{if } A ::= f(A_1, \ldots, A_m) \\
\mathrm{val}_G(A) = \mathrm{val}_G(C_1)[\mathrm{val}_G(A_2)] & \text{if } A ::= C_1[A_2] \\
\mathrm{val}_G(A) = \mathrm{val}_G(A') & \text{if } A ::= A' \\
\mathrm{val}_G(C) = [\cdot] & \text{if } C ::= [\cdot] \\
\mathrm{val}_G(C) = \mathrm{val}_G(C_1)[\mathrm{val}_G(C_2)] & \text{if } C ::= C_1[C_2] \\
\mathrm{val}_G(C) = f(t_1, \ldots, t_{i-1}, [\cdot], t_{i+1}, \ldots, t_m) & \\
\qquad \text{if } C ::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_m) \text{ and} & \\
\qquad t_j = \mathrm{val}_G(A_j) \text{ for } j = 1, \ldots, i-1, i+1, \ldots, m &
\end{array}
$$

The cdepth *of a context nonterminal D is the maximal n of all sequences $D \to_G D_1 \to_G D_2 \ldots \to_G D_n$, where only rules of the form $C ::= C_1[C_2]$ are taken into account, and the cdepth of the STG G is defined as the maximum of all cdepths of nonterminals. The* size $|G|$ *of a grammar G is the sum of the sizes of all right hand sides, where the hole $[\cdot]$ counts as 1. If the arity of function symbols is $O(1)$, then we could have also used the number of the rules of G.* □

Note that for every nonterminal N of G, the term or context $\mathrm{val}_G(N)$ is defined.

Example 2. For $n \in \mathbb{N}$ let G_n be the STG $(\{A, B\}, \{C_0, \ldots, C_n\}, \{a, f\}, R)$ where R is the following set of productions:
$A ::= a$; $B ::= C_n[A]$; $C_0 ::= f([\cdot])$; $C_{i+1} ::= C_i[C_i]$ for $i = 0, \ldots, n-1$.
Then $\mathrm{val}_{G_n}(B) = f^{(2^n)}(a)$ and $|G_n| = 2n + 5$. The cdepth of C_i is i.

Proposition 3 ([4,24,5,17,22]). *For an STG G and two term nonterminals A_1, A_2 it can be decided in $O(|G|^3)$ whether $\mathrm{val}_G(A_1) = \mathrm{val}_G(A_2)$ holds.*

Directed acyclic graphs (DAGs) can be represented by STGs. Since DAGs only share subterms, context nonterminals must not be used to represent DAGs.

Definition 4. *A* DAG *G is an STG where $\mathcal{CN} = \emptyset$. A DAG is called* optimally compressed*, if for nonterminals $A, B : A \neq B \implies \mathrm{val}(A) \neq \mathrm{val}(B)$.*

Note that for every non-optimal DAG there exists either a production $A_1 ::= A_2$ or at least two productions $A_1 := r$ and $A_2 ::= r$. For the following complexity analyses and lemma we assume that the signature is fixed, and that the arity of function symbols is $O(1)$.

Lemma 5 ([20,26,13]). *A DAG G can be transformed into an optimally compressed DAG in time $O(|G| \cdot \log(|G|))$. The size is not increased by this operation.*

Proof. Though this appears to be well-known, we give a sketch: First we use topological sorting to produce in time $O(|G|)$ a list L of nonterminals of G where $A > A'$, if $A \xrightarrow{+}_G A'$. We operate on the list in reverse order. Assume that we construct the optimally compressed grammar from G by scanning the list. During the reconstruction, two data structures are used: (i) a data base with keys $f(A_1, \ldots, A_n)$, nonterminals as entries, and $O(\log m)$ access time if the data base has m entries; (ii) a function *on* mapping nonterminals to their optimal node. Let A be the current nonterminal, G' be the constructed new DAG and G'' be the remaining rules of the grammar G. For the current given nonterminal A with rule $A ::= f(A_1, \ldots, A_k)$, there are two cases: (i) If there is an entry A' under key $f(on(A_1), \ldots, on(A_k))$, then define $on(A) := A'$, and remove the rule for A. (ii) If there is no entry under key $f(on(A_1), \ldots, on(A_k))$, then define $on(A) := A$, and let the new rule be $A ::= f(on(A_1), \ldots, on(A_k))$ and insert A in the data base with key $f(on(A_1), \ldots, on(A_k))$.

Finally, the start symbol can be replaced using function *on*. Since the sum of the number of nonterminals of G' and G'' is at most $|G|$, the time per nonterminal is $O(\log(|G|))$, and the list is also of length $O(|G|)$, hence this can be done in time $O(|G| \cdot \log(|G|))$. The size of the DAG is not increased. □

We say a term t is *STG-compressed* (or *DAG-compressed*, respectively), if there is an STG (or a DAG, respectively) G and a term-nonterminal A of G such that $\mathrm{val}_G(A) = t$. A TRS T is called *STG-compressed* (or *DAG-compressed*, respectively), if all terms l_i, r_i of the rewriting relation are represented by term-nonterminals L_i, R_i of an STG (or a DAG, respectively) G such that $\mathrm{val}_G(L_i) = l_i$ and $\mathrm{val}_G(R_i) = r_i$. We use the analogous notions also for sets of equations.

3 The Word-Problem for STG-Compressed Terms with DAG-Compressed Ground Equations

In this section we show that the word problem for ground equations and STG-compressed terms s, t can be solved in polynomial time w.r.t. the size of the compressed representation.

3.1 A Normalizing Algorithm for Compressed Terms

First we describe an algorithm for T-normalizing all terms represented by term nonterminals in an STG, where T is a given reduced, canonical gTRS. The idea is to modify the STG such that there is no reducible subterm of any represented term in the STG, for any nonterminal.

We assume that a reduced, canonical TRS T is given and that T is optimally DAG-compressed, i.e. the terms l_i and r_i for $i = 1, \ldots, m$ are represented in the (optimally compressed) DAG $G_T = (\mathcal{TN}_T, \emptyset, \Sigma_T, R_T)$, such that there are nonterminals $L_i, R_i \in \mathcal{TN}_T$ with $\mathrm{val}_{G_T}(L_i) = l_i$ and $\mathrm{val}_{G_T}(R_i) = r_i$ for $i = 1, \ldots, n$. For convenience we sometimes write $T = \{L_1 \to R_1, \ldots, L_m \to R_m\}$ for the TRS. Note that the nonterminals $R_1, \ldots, R_m$ are not necessarily distinct. We assume that the to-be-normalized terms are STG-compressed, i.e. there is an input STG $G_{inp} = (\mathcal{TN}_{inp}, \mathcal{CN}_{inp}, \Sigma_{inp}, R_{inp})$, such that $\mathcal{TN}_{inp} \cap \mathcal{TN}_T = \emptyset$. Let G be the union of G_T and G_{inp}.

For the TRS T and its corresponding DAG G_T we define the sets $\mathrm{subterms}_{\mathrm{NT}}(T)$ and $\mathrm{subterms}(T)$ as follows, where $\top$ is an extra symbol:

$$\begin{aligned} \mathrm{subterms}_{\mathrm{NT}}(T) &:= \{A \mid L_i \xrightarrow{+}_{G_T} A,\ i = 1, \ldots, m\} \cup \{R_i \mid i = 1, \ldots, m\} \cup \{\top\} \\ \mathrm{subterms}(T) &:= \{\mathrm{val}(A) \mid A \in \mathrm{subterms}_{\mathrm{NT}}(T) \setminus \{\top\}\} \end{aligned}$$

The set $\mathrm{subterms}_{\mathrm{NT}}(T)$ comprises all nonterminals that are referenced by left hand sides L_i of T, the nonterminals R_i, and a distinguished constant $\top$. Every proper subterm of a left-hand side l_i is represented by one nonterminal in $\mathrm{subterms}_{\mathrm{NT}}(T)$, a nonterminal for every right-hand side r_i is in $\mathrm{subterms}_{\mathrm{NT}}(T)$, and $\top$ represents the other terms.

Note that $\{L_i \mid i = 1, \ldots, m\} \cap \mathrm{subterms}_{\mathrm{NT}}(T) = \emptyset$, and that for every $A \in \mathrm{subterms}_{\mathrm{NT}}(T) \setminus \{\top\}$ the term $\mathrm{val}(A)$ is T-irreducible, since T is reduced.

The algorithm below will modify the grammar G bottom-up, where only the rules of G_{inp} as a sub-STG of G are modified by replacing the right hand sides of the grammar rules, and also some rules for constructing context-nonterminals will be added. For the term nonterminals of G_{inp} the resulting STG G' will only represent normalized terms, i.e. for every nonterminal A of G_{inp}: $\mathrm{val}_{G'}(A)$ is the T-normal form of $\mathrm{val}_G(A)$. The following algorithms are designed to avoid the Plandowski-equality check (Proposition 3) and perform all checks for equality either on the name of the symbols or checking in the DAG, which requires that the DAG for T is optimally compressed.
First we define an algorithm that computes two functions ϕ_0, ϕ_1 such that:

– For every $A \in \mathcal{TN}_{inp}$, we have $\phi_0(A) \in \text{subterms}_{\text{NT}}(T) \subseteq \mathcal{TN}_{inp}$. If $\text{val}(A) \in \text{subterms}(T)$ then $\phi_0(A) \in \text{subterms}_{\text{NT}}(T) \setminus \{\top\}$, and the term $\text{val}(\phi_0(A))$ will be the normal form of $\text{val}(A)$. Otherwise, $\phi_0(A)$ will be $\top$.
– For every $C \in \mathcal{CN}_{inp}$, we have $\phi_1(C) : \text{subterms}_{\text{NT}}(T) \to \text{subterms}_{\text{NT}}(T)$. This function computes the mapping behavior of the context $\text{val}(C)$ on $\text{subterms}(T)$ after normalization: For every nonterminal A in $\text{subterms}_{\text{NT}}(T)$, if the T-normal form of $\text{val}(C)[\text{val}(A)]$ is in $\text{subterms}(T)$, then $\phi_1(C)(A) \in \text{subterms}_{\text{NT}}(T) \setminus \{\top\}$ and $\text{val}(\phi_1(C)(A))$ is the T-normal form of $\text{val}(C)[\text{val}(A)]$, otherwise $\phi_1(C)(A)$ will be $\top$.

The following subalgorithm dagNode is required, which computes for nonterminals A_i from G_T the node in G_T for $f(A_1, \ldots, A_n)$ if it exists. Formally, for $A_i \in \mathcal{TN}_T \cup \{\top\}$ and a function symbol $f \in \Sigma_T \cup \Sigma_{inp}$ of arity n let $\text{dagNode}(f, A_1, \ldots, A_n)$ be defined as follows:

$$\text{dagNode}(f, A_1, \ldots, A_n) := \begin{cases} N, \text{ if } N ::= f(A_1, \ldots, A_n) \in R_T \\ \bot, \text{ otherwise} \end{cases}$$

Algorithm 6 (The ϕ-Computation Algorithm). The algorithm incrementally computes ϕ_0 and ϕ_1 by inspecting the production rules of G_{inp} in bottom-up order, i.e. in the order reverse to $\to_{G_{inp}}$.

The treatment of production rules is defined by a case analysis:

1. $A ::= f(A_1, \ldots, A_n)$, with $n \geq 0$. If $\text{dagNode}(f, \phi_0(A_1), \ldots, \phi_0(A_n)) = \bot$ then define $\phi_0(A) := \top$.
   Otherwise, let $\text{dagNode}(f, \phi_0(A_1), \ldots, \phi_0(A_n)) = N$ and define
   $$\phi_0(A) := \begin{cases} R_i, \text{ if } N = L_i \text{ for some left hand side of a rule } L_i \to R_i \\ N, \text{ if } N \neq L_i \text{ and } N \in \text{subterms}_{\text{NT}}(T) \\ \top, \text{ otherwise} \end{cases}$$
2. $C ::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_n)$. We compute the function $\phi_1(C)$ for all arguments $B \in \text{subterms}_{\text{NT}}(T)$:
   The first case is $\phi_0(A_j) = \top$ for some $j \neq i$. Then $\phi_1(C)(B) := \top$ for all $B \in \text{subterms}_{\text{NT}}(T)$.
   The other case is $\phi_0(A_j) \neq \top$ for all $j \neq i$. Let $\phi_1(C)(B) := \top$, if $B = \top$ or $\text{dagNode}(f, \phi_0(A_1), \ldots, \phi_0(A_{i-1}),\ B, \phi_0(A_{i+1}), \ldots, \phi_0(A_n)) = \bot$. Otherwise, let $N := \text{dagNode}(f, \phi_0(A_1), \ldots, \phi_0(A_{i-1}),\ B, \phi_0(A_{i+1}), \ldots, \phi_0(A_n))$ and define:
   $$\phi_1(C)(B) := \begin{cases} R_i, \text{ if } N = L_i \text{ for some left hand side of a rule } L_i \to R_i \\ N, \text{ if } N \neq L_i \text{ and } N \in \text{subterms}_{\text{NT}}(T) \\ \top, \text{ otherwise} \end{cases}$$
3. $A_1 ::= A_2$. Then define $\phi_0(A_1) := \phi_0(A_2)$.
4. $C ::= [\cdot]$. Then $\phi_1(C)$ is the identity function on $\text{subterms}_{\text{NT}}(T)$.
5. $C ::= C_1[C_2]$. Since the algorithm proceeds bottom-up, we already have computed the functions $\phi_1(C_1), \phi_1(C_2)$, and so we can compute the function $\phi_1(C)$ as the composition $\phi_1(C_1) \circ \phi_2(C_2)$.

6. $A ::= C[B]$. Then $\phi_0(A) := \phi_1(C)(\phi_0(B))$.

Lemma 7. *Let G be an STG and T be a reduced and confluent gTRS. Then for every $A \in \mathcal{TN}_{inp}$ the following holds: If $\phi_0(A) = \top$, then $\mathrm{val}(A)$ is not a subterm of any left hand side of T. Consequently, there are no superterms of $\mathrm{val}(A)$ that are redexes. This also holds during the whole reduction process, hence also the T-normal form s of $\mathrm{val}(A)$ is not a subterm of any left hand side of T.*

Example 8. We consider the TRS T with one rule $f(f(a)) \to a$ represented by $\{L_1 \to R_1\}$ where the corresponding DAG G_T has the production rules $L_1 ::= f(F),\ F ::= f(R_1),\ R_1 ::= a$. Then $\mathrm{subterms}_{\mathrm{NT}}(T) = \{F, R_1, \top\}$. We consider the STG of Example 2 and compute ϕ_0, ϕ_1 as follows:

$$\phi_0(A) = R_1 \quad \left| \begin{array}{l} \phi_1(C_0)(\top) = \top \\ \phi_1(C_0)(F) = R_1 \\ \phi_1(C_0)(R_1) = F \end{array} \right. \quad \left| \begin{array}{l} \phi_1(C_1)(\top) = \top \\ \phi_1(C_1)(F) = \phi_1(C_0)(\phi_1(C_0)(F)) = F \\ \phi_1(C_1)(R_1) = \phi_1(C_0)(\phi_1(C_0)(R_1)) = R_1 \end{array} \right.$$

For $i = 1, \ldots, n-1$ we have $\phi_1(C_{i+1}) = \phi_1(C_i) \circ \phi_1(C_i)$ and thus $\phi_1(C_i)$ is the identity on $\mathrm{subterms}_{\mathrm{NT}}(T)$ for $i \geq 1$. Finally, we can compute $\phi_0(B) = \phi_1(C_n)(\phi_0(A)) = \phi_1(C_n)(R_1) = R_1$.

The normalization algorithm uses the functions ϕ_0 and ϕ_1 to compute an STG that represents all T-normal forms of terms represented by G_{inp}. Usually, it only changes productions for term nonterminals of the input grammar. The difficult case is a production of the form $A ::= C[B]$ where a subterm of $\mathrm{val}(C)[\mathrm{val}(B)]$ which is a proper superterm of $\mathrm{val}(B)$, is indicated by ϕ_0 as having a normal form representable by some nonterminal A' in $\mathrm{subterms}_{\mathrm{NT}}(T)$, and which is maximal. Then a new context nonterminal C' used to generate the normal form of $\mathrm{val}(C)[\mathrm{val}(B)]$ as $\mathrm{val}(C')[\mathrm{val}(A')]$ must be found and added to the grammar.

Algorithm 9 (Normalization Algorithm). The algorithm has $G = G_{inp} \cup G_T$ as input and uses the (algorithmically defined) functions ϕ_0, ϕ_1. It iterates over all productions of G_{inp} and modifies them according to the following cases.

1. The rules for context nonterminals are unchanged.
2. If the rule is $A_1 ::= A_2$, then the rule is unchanged.
3. If the rule is $A ::= f(A_1, \ldots, A_n)$, and $\phi_0(A) \neq \top$, then replace this rule by $A ::= \phi_0(A)$. If $\phi_0(A) = \top$, do not change the rule.
4. Let the rule be $A ::= C[B]$. Then
   (a) If $\phi_0(B) = \top$, then do not change the rule.
   (b) If $\phi_0(A) \neq \top$, then replace the rule for A by $A ::= \phi_0(A)$.
   (c) If $\phi_0(B) \neq \top$, but $\phi_0(A) = \top$, then the normalization stops somewhere between $\mathrm{val}(C[B])$ and $\mathrm{val}(B)$. The compressed normal form of $\mathrm{val}_G(C[B])$ is constructed as follows: we construct a context nonterminal C' such that $\mathrm{val}_G(C')$ is a prefix of $\mathrm{val}_G(C)$, and have to find a term nonterminal $B' \in \mathrm{subterms}_{\mathrm{NT}}(T) \setminus \{\top\}$ such that $C'[B']$ represents the normal form of $\mathrm{val}(C[B])$ and $\mathrm{val}(C'[B'])$ is irreducible. This is done top-down starting from C by the algorithm *Prefix* defined below. We

perform $\mathit{Prefix}(C, \phi_0(B))$ (that may add rules for context nonterminals as a side-effect) and obtain a result (C', D), which we use to replace the rule for A by the rule $A ::= C'[D]$.

Now we describe $\mathit{Prefix}(C, D)$, where we assume that $D \in \mathrm{subterms}_{\mathrm{NT}}(T) \setminus \{\top\}$. The cases are:

1. $C ::= [\cdot]$ is the rule for C, then return (C, D).
2. $C ::= f(A_1, \ldots, A_{i-1}, [\cdot], A_{i+1}, \ldots, A_n)$ is the rule for C. If $\phi_1(C)(D) \neq \top$, then return $([\cdot], \phi_1(C)(D))$. Otherwise, return (C, D).
3. $C ::= C_1[C_2]$ is the rule for C. Then there are two cases.
   (i) If $\phi_1(C_2)(D) = \top$, then let (C_2', D_2') be the result of $\mathit{Prefix}(C_2, D)$. Construct $C' ::= C_1[C_2']$ in the grammar and return (C', D_2').
   (ii) If $\phi_1(C_2)(D) = D' \neq \top$ then return the result of $\mathit{Prefix}(C_1, D')$.

Example 10. We again consider Example 2 and the gTRS of Example 8. The normalization algorithm produces the grammar G_{out} with the productions:

$$\begin{array}{llll} L_1 ::= f(F) & F ::= f(R_1) & R_1 ::= a & C_{i+1} ::= C_i[C_i] \\ A ::= R_1 & B ::= R_1 & C_0 ::= f([\cdot]) & \text{for } i = 0, \ldots, n-1 \end{array}$$

As expected we have $\mathrm{val}_{G_{out}}(B) = a$.

Lemma 11. *Algorithm 9 is correct: It computes a new STG $G' := G'_{inp} \cup G_T$, where for every term nonterminal A of G_{inp}: $\mathrm{val}_G(A) =_T \mathrm{val}_{G'}(A)$, and $\mathrm{val}_{G'}(A)$ is T-irreducible.*

Proof. (sketch) It is easy to verify that every modification in the grammar retains equality w.r.t. the equational theory $=_T$. It is also straightforward to check that the only irreducible expressions are represented by the STG after the normalization process has finished. □

Now we estimate the complexity of the normalization algorithm where we use $|G|$, $|G_{inp}|$, and $|G_T|$ as parameters and where we assume that $|\Sigma|$, as well as the arity of function symbols is $O(1)$.

Lemma 12. *The normalization increases the grammar G by at most $O(|G|^2)$ and requires time $O(|G|^2 + |G| \cdot |G_T| \cdot \log(|G_T|))$.*

Proof. The cardinality of $\mathrm{subterms}_{\mathrm{NT}}(T)$ is at most $|G_T|$ since every subterm of left hand sides and every right hand side is represented by a node in the DAG. The grammar is increased during the construction as follows. There is no cdepth-increase of context nonterminals during constructing the prefix of context nonterminals. *Prefix* returns a context and a term nonterminal. The only possibility for the returned term nonterminal for *Prefix* is the input term nonterminal, or a term nonterminal from $\mathrm{subterms}_{\mathrm{NT}}(T) \setminus \{\top\}$. The size increase by one normalization step is at most $\mathrm{cdepth}(G_{inp})$, since the size increase by *Prefix* depends only on the cdepth of context nonterminals. Thus the size increase is $|G_{inp}| \cdot \mathrm{cdepth}(G_{inp})$, which is $O(|G|^2)$.

Concerning the running time, note that all equality comparisons only require to compare nonterminals from $\mathrm{subterms}_{\mathrm{NT}}(T)$, since other comparisons are prevented by $\top$. Hence a single comparison can be done in constant time.

Assuming that ϕ_0, ϕ_1 are stored in an efficient data structure, the computation of the function ϕ_1 corresponding to the context nonterminals requires time $|G_T| \cdot \log(|G_T|)$, which has to be done for every context nonterminal of the initial grammar, thus it requires time $O(|G_{inp}| \cdot |G_T| \cdot \log(|G_T|))$. □

3.2 Deciding the Word Problem

Using the construction in the last subsection, we show how to decide the equality of two STG-compressed terms s_1, s_2 given a set of DAG-compressed ground equations, which is more general than considering plain equations. The following steps provide such a decision algorithm:

The input is a DAG G_E and equations $L_1 = R_1, \ldots, L_n = R_n$, where $\mathrm{val}(L_i)$, $\mathrm{val}(R_i)$ are ground (and all the symbols L_i, R_i are different) and an STG G that represents the terms s_1, s_2 by the nonterminals S_1, S_2.

1. Compute a DAG G_T that represents a reduced gTRS T which is equivalent to G_E using Snyder's algorithm ([27,28,8]). Note that Snyder's algorithm can also be used for DAG-compressed ground equations.
2. Optimally compress the DAG G_T using Lemma 5.
3. Use the algorithm in the previous subsection to construct an STG G' that represents the STG-compressed normal forms of all term nonterminals, in particular the normal forms of s_1, s_2 by the nonterminals S_1, S_2.
4. Use the Plandowski-Lifshits algorithm (Proposition 3) to decide whether S_1, S_2 represent the same terms.

Lemma 13. *[27,28] For a set of ground equations E represented by a DAG G_E (with different symbols for the terms in the equations) one can compute a reduced gTRS T, with $=_T \; = =_E$, represented by a DAG G_T in time $O(|G_E| \cdot \log^2 |G_E|)$ where $|G_T| = O(|G_E|)$.*

Proof. We analyze the steps of the algorithm in [28]: The first step is to generate a DAG for a given set of ground equations E. This step is not necessary for our claim, since E is already represented by G_E. All other steps in the algorithm of [28] are performed on the DAG, and the dominating cost is computing the congruence closure, which can either be done in time and space $O(|G_E| \cdot \log |G_E|)$ or in time $O(|G_E| \cdot \log^2 |G_E|)$ and $O(|G_E|)$ space [7]. □

Using this result and the previous results on normalization we obtain:

Theorem 14. *Given a set of ground equations E, represented by a DAG G_E (with different symbols for the terms in the equations), and two terms s_1, s_2 represented by nonterminals S_1, S_2, respectively, of an STG G_{inp}, a reduced, canonical gTRS T, equivalent to E, and the STG G' representing the T-normal forms of S_1, S_2 can be computed in time $O(|G|^2 + |G| \cdot |G_E| \cdot \log |G_E| + |G_E| \cdot \log^2 |G_E|)$, and $\mathrm{val}(S_1) =_E \mathrm{val}(S_2)$ can be decided in time $O(|G|^6 \cdot \log^3 |G|)$, where $G = G_{inp} \cup G_E$.*

Proof. The construction of the gTRS can be done in time $O(|G_E| \cdot \log^2 |G_E|)$ and space $O(|G_E|)$. The STG G' can be computed in time $O(|G|^2 + |G| \cdot |G_E| \cdot \log |G_E|)$, which results in time $O(|G|^2 + |G| \cdot |G_E| \cdot \log |G_E| + |G_E| \cdot \log^2 |G_E|)$.
Since $G_E \subset G$, the estimation is $O(|G|^2 \cdot \log |G|)$ for the time of the construction, and $O(|G|^6 \cdot \log^3 |G|)$ to perform the equality decision using the Plandowski-Lifshits-algorithm (Proposition 3). □

4 STG-Compressed Ground Term Rewriting Systems

If the ground TRS is STG-compressed, then the normalization algorithms become more involved if we want efficient ones. It is obvious that there is an exponential upper bound on the running time for normalization and the word problem, since after decompression, which increases the size at most exponentially to $2^{|G|}$, we can use the well-known algorithms with $O(n \cdot \log(n))$ running time. In the following we look for improved bounds in special cases.

4.1 Complexity Bounds

Proposition 15. *Given a reduced, confluent gTRS T, represented as $T = \{L_1 \to R_1, \ldots, L_n \to R_n\}$ where L_i, R_i are from an STG G_T. Let s be a term with $\mathrm{val}(S) = s$ where S is a term nonterminal from the STG G. Then the T-normal form of s is computable in polynomial space depending on $|G| + |G_T|$.*

Proof. We show that there is a reduction sequence $s \to s_1 \to \ldots \to s_k \xrightarrow{*} s_n$ where s_n is the T-normal form of s, and where for every k the STG G_k representing s_k requires polynomial space. The claim is that for every k: G_k can be directly derived from G as follows:

- G_k contains the rules of G_T as well as (perhaps modified) rules of G, plus perhaps some additional rules.
- Some term nonterminals in right hand sides of the G-rules may be replaced by R_i for some i.
- Some right hand sides of G-rules of the form $C[A]$ are replaced by $C'[R_i]$, for some i, where $\mathrm{val}(C')$ is a prefix of $\mathrm{val}(C)$. G is extended by the rules generating C'.

Since prefixes of contexts can be generated by an at most polynomial enlargement of the grammar, and the prefixes can be added independently, the size of the STG G_k is at most polynomial in the size of G.

Note that this construction cannot be turned into an efficient algorithm, since the justifications where to replace would require the whole rewrite sequence $s \to s_1 \to \ldots \to s_k$.

It remains to show that G_{k+1} can be derived from G_k by a parallel rewriting step: If the rewriting replaces a term nonterminal in G_k, then the replacement constructs a grammar that can be immediately derived from G.
If the rewriting replaces a subterm of some right hand side $C'[R]$, then the

only possibility is that a context nonterminal C'' has to be constructed such that $\mathrm{val}(C'')$ is a prefix of $\mathrm{val}(C')$ and the right hand side $C'[R]$ is replaced by $C''[R']$, where R' is a right hand side of a rule in T. Hence the algorithm runs in polynomial space. □

Corollary 16. *Given a reduced, confluent gTRSs T, STG-compressed by G_T and two terms s, t also STG-compressed by G, and let $=_T$ be the equality relation derived from T. Then the word problem, i.e. whether $s =_T t$, is in PSPACE.*

We do not know more efficient algorithms for simplifying a compressed term by a reduced gTRS, or for making a compressed gTRS reduced. Determining the exact complexity of the word problem of STG-compressed terms w.r.t. a set of STG-compressed ground equational axioms is left for future research.

Proposition 17. *Let $\Sigma = \{f, b_1, \ldots, b_n, b_{n+1}\}$ be a signature (n a positive integer) where f is unary and b_i are constants. Given a gTRSs T over Σ, compressed by an STG G_T, such that the right hand sides of rewriting rules are constants from Σ, and a term s also compressed by an STG G, then the problem whether s has b_{n+1} as a normal form under T is NP-hard.*

Proof. We adapt the proofs in [23] for our specific problem. We use positive SUBSETSUM as an NP-hard problem ([9]). Given n (positive) integers $S := \{a_1, \ldots, a_n\}$, and another integer m. Then the question is whether there there is a subset $S' \subseteq S$, such that $\sum_{a \in S'} a = m$.
The uncompressed TRS T is constructed as follows: It has rules of the forms $f^{a_i}(b_i) \to b_{i+1}$ and $b_i \to b_{i+1}$ for $i = 1, \ldots, n$, and the term s is of the form $f^m(b_1)$. These terms can easily be compressed in polynomial space. The question is whether s can be reduced to b_n using the rules of T. Such a reduction corresponds to a sum as in SUBSETSUM. Hence the problem is NP-hard. □

Remark 18. Proposition 17 does not show that the compressed word-problem w.r.t. a ground equational theory is NP-hard. For example, the equational theory of the encoding in Proposition 17 can be decided in polynomial time: It can be reduced to the axiom $f^k(b_{n+1}) =_T b_{n+1}$, where k is the greatest common divisor of all the numbers a_i, and the axioms $b_i = b_{n+1}$. Then the word problem can be decided in polynomial time by a computation modulo k.

4.2 STG-Compressed gTRS with One Rule

We consider the problem of normalizing an STG-compressed term t using a single STG-compressed ground rule $L \to R$, where L does not occur in R, and hence for deciding the word problem w.r.t. $L \to R$.

A naive method is to perform step-by-step normalization. One step is to find all positions of $\mathrm{val}(L)$ in $\mathrm{val}(t)$, constructing the corresponding nonterminals and replacing them by a reference to R. Such a single step can be done in polynomial time. In general, this will lead to an exponential number of normalization steps: Let the term be $f^n(a)$, and the rewrite rule be $f^{m+1}(a) \to f^m(a)$, where $n > m$

are large numbers. Since $f^n(a)$ can be represented in an STG of size $\log(n)$, and since every rewrite step only reduces the exponent by 1, there will be $n - m$ rewrite steps during normalization, which may be exponentially large in $|G|$. For the following special cases normalization can be performed in polynomial time:

- If $\mathrm{val}(L)$ has no occurrences of $\mathrm{val}(R)$, then replacing all occurrences of L by R is sufficient. These may be explicit occurrences, when $\mathrm{val}(L) = \mathrm{val}(A)$ for a term nonterminal, or implicit occurrences, when $\mathrm{val}(L)$ occurs in $\mathrm{val}(C[A])$ as a subterm between A and $C[A]$. In this case there is only one such possibility, which can easily be constructed.
- For the (nontrivial) case that $\mathrm{val}(R)$ occurs exactly once in $\mathrm{val}(L)$ the occurrence can be found by ground submatching, also the representation $L'[R]$ for L such that $\mathrm{val}(L'[R]) = \mathrm{val}(L)$ with a context nonterminal L' can be easily constructed. The computation of a representation of all occurrences of the context L' in some C is in [25]. More exactly, the occurrences of the form $L'^n[R]$ with a maximal n have to be determined. Using the context-in-context table of [25], and using binary search for the maximal n, a polynomial algorithm can be constructed for this task. If the occurrences are found, (at most one per term nonterminal), then we can replace these occurrences by R, and obtain a normalized term t' for t.

4.3 Monadic Ground Term Rewriting Systems

We investigate the special case of monadic signatures $\Sigma := \{f_1, \ldots, f_m, a\}$ consisting only of unary function symbols f_i and a single constant. Assume a compressed and reduced (i.e., a confluent and terminating) gTRS $\{L_i \to a \mid i = 1, \ldots, n\}$ over monadic Σ. We compute the nonterminals $A_{i,j}$, and the corresponding rules, such that whenever $\mathrm{val}(R_i)$ occurs in $\mathrm{val}(L_j)$, then $\mathrm{val}(A_{i,j}(R_i)) = \mathrm{val}(L_j)$. Given a term s, the rewriting process identifies a left hand side $\mathrm{val}(L_i)$ occurring in s, and rewrites this to $\mathrm{val}(R_i)$. Since the signature is monadic, this can be interpreted as rewriting a string where every rewrite must replace a suffix. Since the gTRS is reduced, the rewriting process can also be seen as a computation that acts like a deterministic finite automaton: For instance, let $s = \mathrm{val}(A_{2,5}(A_{3,2}(A_{1,3}(R_1))))$. Then $\mathrm{val}(A_{1,3}(R_1)) = \mathrm{val}(L_3)$, hence it proceeds as $\mathrm{val}(A_{2,5}(A_{3,2}(R_3)))$. The next reduction steps are $\mathrm{val}(A_{2,5}(A_{3,2}(R_3))) = \mathrm{val}(A_{2,5}(L_2)) \to \mathrm{val}(A_{2,5}(R_2) = \mathrm{val}(L_5) \to \mathrm{val}(R_5)$. Since the TRS is confluent and reduced, the computation is deterministic. Translating this into a DFA-computation: the starting state is i, where L_i is the left hand side occurring in s, and the next state depends on the symbol $A_{i,j}$. We can also add an initial step $\varepsilon \to R_i$, where we can omit ambiguous steps. i.e. if an $\mathrm{val}(R_i)$ is a proper suffix of $\mathrm{val}(R_j)$, then we can omit this step in the automata. Every state of the DFA is accepting.

On the other hand, every DFA where all states are accepting can be interpreted as such a TRS: Let all $\mathrm{val}(R_i)$ be trivial, and let the left hand sides be 1, 10, 100, 1000, ... Then the TRS is reduced, and the question whether a compressed string can be reduced to ε can be solved looking at the DFA.

This implies:

n	time (sec)
50 000	2.15
100 000	5.77
250 000	20.79
500 000	61.91
1 000 000	221.66

Table 1. Test series 1

n	m	time (sec)
5 000	1 000	6.03
5 000	2 000	11.89
5 000	5 000	38.75
5 000	10 000	93.85
100 000	1 000	649.13

Table 2. Test series 2

n	k	time (sec)
5 000	1 000	6.06
5 000	2 000	13.20
5 000	5 000	36.65
5 000	10 000	81.19
100 000	1 000	666.34

Table 3. Test series 3

Lemma 19. *The question whether a term over a monadic signature can be reduced to a is polynomially equivalent to the question: given a DFA and a compressed word s, is there is a word w over context nonterminals accepted by the DFA such that* $\mathrm{val}(w) = \mathrm{val}(s)$.

However, note that for small $\mathrm{val}(A_{i,j})$, i.e. if $\mathrm{val}(A_{i,j})$ can be viewed as part of the input, there is a polynomial algorithm to solve this problem by using dynamic programming over the compressions of s. Thus the open question is the complexity of these problems for arbitrary $\mathrm{val}(A_{i,j})$.

5 Implementation and Tests

We implemented the normalization algorithm and the Plandowski-Lifshits equality check in the lazy functional programming language Haskell [1]. STGs are implemented as maps (available by the Haskell library `Data.Map`) where the right hand side of a production is mapped to its left hand side. Haskell's maps are based on size balanced binary trees [2] and provide selection and construction operations, like lookup, insertion, or deletion, in logarithmic time which makes our prototypical implementation reasonably fast. Including some example grammars and term rewriting systems our prototypical implementation consists of about 2000 lines of Haskell source code. A cabal[1] package is available under http://www.ki.informatik.uni-frankfurt.de/research/gbc/.

We performed several tests[2] on the grammar of Example 2 as G_{inp}.

For the first series of tests we used the gTRS of Example 8. Table 1 shows the runtime of the normalization algorithm for different values of n. We observe that the runtime grows by increasing n. Nevertheless our algorithm performs fast, since the gTRS has only one rule and the corresponding DAG is small.

The second series of tests uses TRSs with a single rule of the form $f^m(a) \to a$. The corresponding DAG is $G_T = \{\{L_1, A_1, \ldots, A_{m-1}, R_1\}, \emptyset, \{f, a\}, R\}$ where $R = \{L_1 ::= f(A_1), A_1 ::= f(A_2), \ldots, A_{m-1} ::= f(R_1), R_1 ::= a\}$. Table 2 shows runtimes for different m, n. Note that $|\mathrm{subterms}_{\mathrm{NT}}(T)|$ is much larger than in the first series of tests.

The third series of tests concerns a growing number of rules of the TRS. We used TRSs with k rules $f(a) \to b_1, f(b_1) \to b_2, \ldots f(b_{k-1}) \to a$ (represented by

[1] http://www.haskell.org/cabal/

[2] All tests have been compiled with the Glasgow Haskell compiler with optimization turned on, on a Linux machine with an Intel(R) Core(TM) i5 CPU 680 @ 3.60GHz with 4 MB cache processor and 8 GB main memory.

a DAG of size $3k$). The runtimes for normalization are given in table 3. This tables validates the expectation that the size of the DAGs is important, since the DAG-sizes in the first and second series are similar.

6 Conclusion and Further Work

We showed that STG-compression can advantageously be applied to the word problem for STG-compressed large terms w.r.t. DAG-compressed ground equational theories, which may have a potential use in deduction systems.

Further work is to attack some of the open questions: look for an efficient algorithm for solving the word problem for a set of ground equations or gTRSs under STG-compression or prove hardness results.

Acknowledgement

We thank Markus Lohrey for hints on the complexity of related problems. We also thank the anonymous referees for their helpful comments.

References

1. The Haskell Programming Language, 2011. `http://www.haskell.org`.
2. S. Adams. Efficient sets - a balancing act. *J. Funct. Program.*, 3(4):553–561, 1993.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.
4. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML documents. In G. M. Bierman and C. Koch (eds.), *10th DBPL*, *LNCS*, vol. 3774, pp. 199–216. Springer, 2005.
5. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33(4–5):456–474, 2008.
6. 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 October 2002.
7. P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27:758–771, 1980.
8. J.H. Gallier, P. Narendran, D.A. Plaisted, S. Raatz, and W. Snyder. An algorithm for finding canonical sets of ground rewrite rules in polynomial time. *J. ACM*, 40(1):1–16, 1993.
9. M.R. Garey and D.S. Johnson. *"Computers and Intractability": A guide to the theory of NP-completeness*. W.H. Freeman and Co., San Francisco, 1979.
10. A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context matching for compressed terms. In *23rd LICS*, pp. 93–102. IEEE Computer Society, 2008.
11. A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and matching on compressed terms. http://arxiv.org/abs/1003.1632v1, 2010.
12. A. Gascón, S. Maneth, and L. Ramos. First-Order Unification on Compressed Terms. In M. Schmidt-Schauß (ed.), *22nd RTA*, *LIPIcs*, vol. 10, pp. 51–60, 2011.
13. D. Kozen. Complexity of finitely presented algebras. In *9th STOC*, pp. 164–177. ACM, 1977.

14. J. Levy, M. Schmidt-Schauß, and M. Villaret. Bounded second-order unification is NP-complete. In F. Pfenning (ed.), *17th RTA*, *LNCS*, vol. 4098, pp. 400–414. Springer, 2006.
15. J. Levy, M. Schmidt-Schauß, and M. Villaret. Stratified context unification is NP-complete. In U. Furbach and N. Shankar (eds.), *3rd IJCAR*, *LNCS*, vol. 4130, pp. 82–96. Springer, 2006.
16. J. Levy, M. Schmidt-Schauß, and M. Villaret. On the complexity of bounded second-order unification and stratified context unification. http://www.ki.informatik.uni-frankfurt.de/papers/schauss/, 2011. To appear in Logic J. of the IGPL.
17. Y. Lifshits. Processing compressed texts: A tractability border. In B. Ma and K. Zhang (eds.), *18th CPM*, *LNCS*, vol. 4580, pp. 228–240. Springer, 2007.
18. M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210–1240, 2006.
19. M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction in grammar-compressed trees. In L. de Alfaro (ed.), *12th FoSSaCS*, *LNCS*, vol. 5504, pp. 212–226. Springer, 2009.
20. G. Nelson and D.C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980.
21. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t). *J. ACM*, 53(6):937–977, 2006.
22. W. Plandowski. Testing equivalence of morphisms in context-free languages. In J. van Leeuwen (ed.), *2nd ESA*, *LNCS*, vol. 855, pp. 460–470. Springer, 1994.
23. W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg (eds.), *Jewels are Forever*, pp. 262–272. Springer, 1999.
24. 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, 2005.
25. M. Schmidt-Schauß. Pattern matching of compressed terms and contexts and polynomial rewriting. Frank report 43, Institut für Informatik. Goethe-Universität Frankfurt am Main, 2011.
26. R. E. Shostak. An algorithm for reasoning about equality. *Commun. ACM*, 21(7):583–585, 1978.
27. W. Snyder. Efficient ground completion: An $o(nlogn)$ algorithm for generating reduced sets of ground rewrite rules equivalent to a set of ground equations e. In N. Dershowitz (ed.), *3rd RTA*, *LNCS*, vol. 355, pp. 419–433. Springer, 1989.
28. W. Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *J. Symb. Comput.*, 15(4):415–450, 1993.
29. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, 23(3):337–343, 1977.