Algorithmic Compression of Finite Tree Languages by Rigid Acyclic Grammars

SEBASTIAN EBERHARD, GABRIEL EBNER, and STEFAN HETZL, TU Wien

We present an algorithm to optimally compress a finite set of terms using a vectorial totally rigid acyclic tree grammar. This class of grammars has a tight connection to proof theory, and the grammar compression problem considered in this article has applications in automated deduction. The algorithm is based on a polynomial-time reduction to the MaxSAT optimization problem. The crucial step necessary to justify this reduction consists of applying a term rewriting relation to vectorial totally rigid acyclic tree grammars. Our implementation of this algorithm performs well on a large real-world dataset.

CCS Concepts: • **Theory of computation → Grammars and context-free languages**; *Rewrite systems*; *Tree languages*;

Additional Key Words and Phrases: Grammar-based compression, finite tree languages, MaxSAT

ACM Reference format:
Sebastian Eberhard, Gabriel Ebner, and Stefan Hetzl. 2017. Algorithmic Compression of Finite Tree Languages by Rigid Acyclic Grammars. *ACM Trans. Comput. Logic* 18, 4, Article 26 (September 2017), 20 pages.
https://doi.org/10.1145/3127401

1 INTRODUCTION

Formal grammars are one of the standard tools for text compression, used in several popular algorithms (Storer and Szymanski 1982; Nevill-Manning and Witten 1997; Larsson and Moffat 1999; Kieffer and Yang 2000). The increased use of XML documents in computer science has fueled the interest in tree languages, and many compression techniques have been adapted and extended to provide compact tree representations (Yamagata et al. 2003; Lohrey et al. 2013; Busatto et al. 2008). Nowadays, the use of tree grammars constitutes a standard technique for compressing XML documents (Sakr 2009). Grammar-based compression also has the considerable practical advantage that many operations can be performed directly on the compressed representation (Lohrey 2012).

Such grammar-based algorithms typically compress a single string (or tree). In this article, we consider the problem of simultaneously compressing a finite set of trees by a vectorial totally rigid acyclic tree grammar (VTRATG, see Definition 2.1). These VTRATGs describe a class of languages similar to the one accepted by rigid tree automata introduced in Jacquemard et al. (2009, 2011). Our motivation for considering this problem is rooted in proof theory and automated deduction: as shown in Hetzl (2012), there is an intimate relationship between a certain class of proofs in first-order predicate logic and VTRATGs. This relationship has been exploited in Hetzl et al. (2012) to

This work is supported by the Vienna Science Fund (WWTF) project VRG12-004 and FWF project W1255-N23.
Authors' addresses: S. Eberhard, G. Ebner, and S. Hetzl, Institute of Discrete Mathematics and Geometry, TU Wien, Wiedner Hauptstrase 8-10/104, 1040 Wien, Austria; emails: sebastian.eberhard84@gmail.com, {gabriel.ebner, stefan.hetzl}@tuwien.ac.at.

https://doi.org/10.1145/3127401

devise an algorithm that analyzes the structure of a given input proof and computes a lemma whose introduction into the input proof reduces its size. This algorithm has been extended to the computation of an arbitrary number of lemmas in Hetzl et al. (2014a), to predicate logic with equality in Hetzl et al. (2014b), and to induction in Eberhard and Hetzl (2015b). The combinatorial gist of this algorithm is the problem of grammar-based compression considered in this article: a grammar with a minimal number of production rules will yield a proof with a minimal number of quantifier inference rules. This tight connection between the complexity of proofs and grammars has been used in Eberhard and Hetzl (2015a) to prove a lower bound on the length of certain proofs by proving a lower bound on the size of the corresponding grammars.

The proof-theoretic application of our work entails a shift of emphasis compared to traditional grammar-based compression in the following two respects: First, we do not have any freedom of choice regarding the type of grammar. VTRATGs have to be used because they can be translated to proofs afterward. In order to prevent a trivial solution, we are also looking for a VTRATG with a fixed number of nonterminals. Second, when compressing a finite set of terms L, we are looking for a minimal grammar G such that $L(G) \supseteq L$ instead of reproducing L perfectly. This is the case because L describes a disjunction that is required to be a tautology (a so-called Herbrand disjunction, see Herbrand (1930) and Buss (1995)), and if $L' \supseteq L$, then L' also describes a tautology.

Note that the condition $L(G) \supseteq L$ is similar to (but different from) the one imposed on cover automata (Câmpeanu et al. 1998, 2001): there an automaton A is sought such that $L(A) \supseteq L$, but in addition it is required that $L(A) \setminus L$ consists only of words longer than any word in L.

Another notion from the literature related to our work is that of the grammatical complexity of a finite language as defined in Bucher et al. (1981) and studied further in Bucher (1981), Alspach et al. (1983), Bucher et al. (1984), and Tuza (1987): the grammatical complexity of a finite language L is defined as the minimal number of productions of a grammar G with $L(G) = L$. Each class of grammars thus gives rise to a measure of descriptional complexity.

In this article, we present an algorithm that finds a compressing grammar that is minimal with respect to the number of productions. We will precisely formulate this as an optimization problem in Problem 2.6: the Parameterized Language Cover Problem consists of finding a grammar with the minimal number of productions that covers the input term set. Algorithm 1 then solves this problem in the following steps:

(1) Compute a larger grammar that covers the term set and contains a covering subgrammar of minimal size, in polynomial time.
(2) Produce a MaxSAT problem that encodes the minimization of this large grammar.
(3) Use a MaxSAT solver to obtain a solution to the MaxSAT problem, and return the minimal VTRATG corresponding to this solution.

In Section 2, we will define VTRATGs and their derivations, and study how language generation commutes with rewriting in Section 3. We then obtain a specific rewrite relation $\sqsubseteq_L$ describing the regularities of a set of terms L in Section 4, and define the so-called stable grammar $S_{N,L}$ in Section 5—this is the aforementioned large grammar. The computation of the stable grammar is described in Section 6, and we describe the encoding of the Grammar Minimization Problem as an instance of MaxSAT in Section 7. Finally, we evaluate and compare our implementation of the algorithm on a real-world dataset in Section 8.

2 TREE GRAMMARS

In this section, we will define a special class of tree grammars called vectorial totally rigid acyclic tree grammars, or VTRATGs for short. These are different from regular tree grammars (as presented, for example, in Comon et al. (2007)) in two ways: nonterminals are *vectors*, and

the derivation relation is highly restricted. (A similar restriction on the derivation relation was introduced in Jacquemard et al. (2009).) For simplicity, we will define VTRATGs and their generated languages using iterated substitutions here; these are equivalent to a more classical approach using restricted derivations, see Hetzl (2012) for the detailed correspondence.

We consider terms $t \in \mathcal{T}(\Sigma)$ over a signature Σ of function symbols with arity; constants are just nullary functions. We write $f/n \in \Sigma$ if the function symbol f has arity n. A vector of terms $\overline{t} = (t_1, \ldots, t_n)$ is a finite sequence of terms. Nonterminals are special variables, and variables are just nullary function symbols. We will write $\mathcal{T}(\Sigma \cup X \cup N)$ for the set of terms containing variables from X and nonterminals from N; the set of constants is always disjoint from the set of variables.

Positions p are finite sequences of natural numbers; $\mathrm{Pos}(t)$ is the set of positions in a term, $t|_p$ is the subterm of t at position p, and $t[s]_p$ is the term t where the position p is replaced by another term s. The set $\mathrm{st}(t) = \{t|_p \mid p \in \mathrm{Pos}(t)\}$ is the set of subterms of t. We also define the relations $t \trianglelefteq s$ if and only if $t \in \mathrm{st}(s)$, and $t \triangleleft s$ if and only if $t \in \mathrm{st}(s) \setminus \{s\}$. The set of subterms $\mathrm{st}(L)$ of a set of terms $L \subseteq \mathcal{T}(\Sigma)$ is given by $\mathrm{st}(L) = \bigcup_{t \in L} \mathrm{st}(L)$. A term t subsumes a term s if there exists a substitution σ such that $t\sigma = s$.

Definition 2.1. Let Σ be a set of function symbols with arity. A VTRATG is given by the tuple $\langle \alpha_{0,0}, N, \Sigma, P \rangle$:

- $\alpha_{0,0}$ is the start nonterminal.
- $N = ((\alpha_{0,0}), \overline{\alpha_1}, \ldots, \overline{\alpha_n})$ is a finite sequence of nonterminal vectors, such that $\overline{\alpha_i} = (\alpha_{i,0}, \ldots, \alpha_{i,k_i})$ for each $1 \leq i \leq n$. The nonterminals are pairwise distinct: $\alpha_{i,j} \neq \alpha_{k,l}$ if $(i,j) \neq (k,l)$.
- Σ is the signature of the grammar.
- P is a finite set of vectorial productions. A vectorial production is a pair $\overline{\alpha_i} \to \overline{t}$, where $\overline{\alpha_i} = (\alpha_{i,0}, \ldots, \alpha_{i,k_i}) \in N$ is a nonterminal vector and $\overline{t} = (t_1, \ldots, t_{k_i})$ is a vector of terms $t_i \in \mathcal{T}(\Sigma \cup \overline{\alpha_{i+1}} \cup \cdots \cup \overline{\alpha_n})$.

The last condition for the productions states the requirement that G is acyclic: for each vectorial production $\overline{\alpha_i} \to \overline{t}$, the right-hand side $\overline{t}$ only contains nonterminals from the nonterminal vectors $\overline{\alpha_{i+1}}, \ldots, \overline{\alpha_n}$.

Example 2.2. $G = \langle \alpha_{0,0}, N, \Sigma, P \rangle$ is a VTRATG where

$$
\begin{aligned}
N &= ((\alpha_{0,0}), (\alpha_{1,0}, \alpha_{1,1})),\\
\Sigma &= \{f/3, c/0, d/0, e/0\},\\
P &= \{(\alpha_{0,0}) \to (f(\alpha_{1,0}, \alpha_{1,1}, \alpha_{1,1})),\\
&\qquad (\alpha_{0,0}) \to (f(\alpha_{1,1}, \alpha_{1,0}, \alpha_{1,0})),\\
&\qquad (\alpha_{1,0}, \alpha_{1,1}) \to (c, d)\\
&\qquad (\alpha_{1,0}, \alpha_{1,1}) \to (d, e)\}.
\end{aligned}
$$

Let X be a set of variables. For a term t, the set $\mathrm{Vars}(t) = \mathrm{st}(t) \cap X$ is called the set of variables in t, and the term t is called ground if $\mathrm{Vars}(t) = \emptyset$. Variables can be substituted by terms: let $x_1, \ldots, x_k \in X$ be variables and $s_1, \ldots, s_k \in \mathcal{T}(\Sigma \cup X)$ be terms, and then a (parallel) substitution $\sigma = [x_1 \backslash s_1, x_2 \backslash s_2, \ldots, x_k \backslash s_k]$ is a finite map from variables to terms and is extended to all terms recursively. Each of the terms s_i may contain any variable, including any of the variables x_j. We write $t\sigma$ for the application of a substitution σ to a term t. Substitutions are extended to vectors as well by substituting each element: $\overline{t}\sigma = (t_1\sigma, \ldots, t_n\sigma)$. We will also define substitutions using vectors: $[\overline{x} \backslash \overline{t}] = [x_1 \backslash t_1, \ldots, x_n \backslash t_n]$.

The right-hand sides of productions are terms in $\mathcal{T}(\Sigma \cup N)$. A production substitutes the nonterminals on the left-hand side by the terms on the right-hand side:

Definition 2.3. Let $G = \langle \alpha_{0,0}, N, \Sigma, P\rangle$ be a VTRATG with the nonterminals $N = (\overline{\alpha_0}, \overline{\alpha_1}, \ldots, \overline{\alpha_n})$. A derivation δ of a term t is a substitution of the form $\delta = [\overline{\alpha_0}\backslash\overline{s_0}][\overline{\alpha_1}\backslash\overline{s_1}]\cdots[\overline{\alpha_n}\backslash\overline{s_n}]$ such that $\alpha_{0,0}\delta = t$ and $\overline{\alpha_i} \to \overline{s_i} \in P$ is a production for each $0 \le i \le n$. The language $L(G)$ generated by the grammar G is the set of all terms of which there is a derivation: $L(G) = \{\alpha_{0,0}[\overline{\alpha_0}\backslash\overline{s_0}]\cdots[\overline{\alpha_n}\backslash\overline{s_n}] \mid \overline{\alpha_i} \to \overline{s_i} \in P\}$.

A consequence of this definition is that the language $L(G)$ of a VTRATG does not contain any nonterminals.

Example 2.4 (Continuing Example 2.2). The VTRATG G generates the four terms $L(G) = \{f(c,d,d), f(d,e,e), f(d,c,c), f(e,d,d)\}$; the term $f(c,d,d) \in L(G)$ has the unique derivation $\delta = [(\alpha_{0,0})\backslash(f(\alpha_{1,0}, \alpha_{1,1}, \alpha_{1,1}))][(\alpha_{1,0}, \alpha_{1,1})\backslash(c,d)]$.

Now that we have defined what VTRATGs are, we can precisely state the optimization problem that we will solve in this article:

Definition 2.5. Let $G = \langle \alpha_{0,0}, N, \Sigma, P\rangle$ be a VTRATG. Its size $|G| = |P|$ is the number of its productions.

This notion of size corresponds to the one used by Bucher et al. (1981).

Problem 2.6 (Parameterized Language Cover Problem). Given a finite set of terms L and a sequence of nonterminal vectors N, find a VTRATG G of minimal size with nonterminal vectors N such that $L(G) \supseteq L$.

If $L(G) \supseteq L$, then we will say that G *covers* L. The parameter in the Parameterized Language Cover Problem refers to the sequence of nonterminal vectors N. If we do not keep this parameter fixed, then the reduction presented in this article will no longer be polynomial time. Furthermore, if we allow arbitrarily large N, then the Language Cover Problem would become trivial as every set of terms would have optimal exponential compression:

Theorem 2.7 (Eberhard and Hetzl 2017). *Let L be a finite set of terms and $l_0, \ldots, l_n$ be natural numbers such that $|L| \le \prod_i l_i$. Then there exists a VTRATG G of size $|G| = \sum_i l_i$ such that $L(G) = L$.*

In particular, for every set of terms L of size $|L| \le 2^n$ there exists a covering VTRATG of size $2n$. Due to the restriction on N, the Parameterized Language Cover Problem we consider here is not trivially solvable.

From a complexity point of view, we can also consider a decision version of the Parameterized Language Cover Problem: given a finite set of terms L, a sequence of nonterminal vectors N, and a bound $n > 0$, does there exist a VTRATG G with nonterminal vectors N such that $L(G) \supseteq L$ and $|G| \le n$? This problem lies in NP: given a VTRATG G and derivations of all the terms in the term set, we can check in polynomial time whether $|G| \le n$, the derivations are all correct, and the VTRATG actually covers the set of terms. It is unknown at the present time whether this problem is already NP-complete, although we conjecture that it is.

The notion of VTRATG originates in the proof-theoretic applications of Hetzl (2012), Hetzl et al. (2014b), Hetzl et al. (2014b), and Ebner et al. (2017): there the set of terms L corresponds to the weak quantifier inferences in a cut-free proof, and finding a covering VTRATG determines the weak quantifier inferences in a proof with universally quantified cuts. Furthermore, there is a tight connection between the size of the VTRATG and the quantifier complexity of the proof with cuts: they are bounded by constant factors. Each of the nonterminal vectors corresponds to a cut in the proof. The length of each nonterminal vector determines the number of universal quantifiers in the cut formula.

3 REWRITING GRAMMARS

The goal of this section is to show that term rewriting and language generation commutes; that is, instead of applying a rewriting relation to a grammar and then generating the language, we can also generate the language from the original grammar and apply the rewriting to the language. (Note that the rewrite relation we will be using is nonconfluent and that the results of rewriting are therefore not deterministic.)

$$\begin{array}{ccc} G & \xrightarrow{R} & G' \\ \downarrow & & \downarrow \\ L(G) & \overset{R}{\dashrightarrow} & L(G') \end{array}$$

Following Dershowitz and Plaisted (2001), we axiomatically define a (single-step) rewrite relation as binary relation that is closed under substitution and congruence.

Definition 3.1. Let $\to_R$ be a binary relation on $\mathcal{T}(\Sigma \cup X)$. Then $\to_R$ is called *monotonic* if $s \to_R t$ implies $u[s]_p \to_R u[t]_p$ for all $s, t, u \in \mathcal{T}(\Sigma \cup X)$ and $p \in \mathrm{Pos}(u)$. It is called *fully invariant* if $s \to_R t$ implies $s\sigma \to_R t\sigma$ for all $s, t \in \mathcal{T}(\Sigma \cup X)$ and substitutions σ. It is called a *rewrite relation* if it is both monotonic and fully invariant.

We will now show how to lift the rewriting from terms to productions, to derivations, to grammars, and then describe the effect the rewriting has on the generated language. For the rest of the section, let $\to_R$ be a fixed rewrite relation and $\to_R^*$ its reflexive and transitive closure. (After this section, we will only consider the specific rewrite relation $\to_R = \to_R^* = \sqsubseteq_L$, see Definition 4.2.)

Definition 3.2. Let $L \subseteq \mathcal{T}(\Sigma \cup X \cup N)$ be a set of terms, $G = \langle \alpha_{0,0}, N, \Sigma, P \rangle$ and $G' = \langle \alpha_{0,0}, N, \Sigma, P' \rangle$ be VTRATGs, $p = (\overline{\alpha_i} \to \overline{s}) \in P$, $p' = (\overline{\alpha_i} \to \overline{s'}) \in P'$ be productions, and $\delta = [\overline{\alpha_0} \backslash \overline{s_0}][\overline{\alpha_1} \backslash \overline{s_1}] \cdots [\overline{\alpha_n} \backslash \overline{s_n}]$, $\delta' = [\overline{\alpha_0} \backslash \overline{s'_0}][\overline{\alpha_1} \backslash \overline{s'_1}] \cdots [\overline{\alpha_n} \backslash \overline{s'_n}]$ be derivations.

We extend rewriting on terms to sets, derivations, productions, and grammars in the natural way as follows:

- $L \to_R^* L'$ if and only if for all $t' \in L'$ there exists a $t \in L$ such that $t \to_R^* t'$ and for all $t \in L$ there exists a $t' \in L'$ such that $t \to_R^* t'$.
- $\delta \to_R^* \delta'$ if and only if $s_{i,j} \to_R^* s'_{i,j}$ for all i and j.
- $(\overline{\alpha_i} \to \overline{s}) \to_R^* (\overline{\alpha_i} \to \overline{s'})$ if and only if $s_j \to_R^* s'_j$ for all j.
- $G \to_R^* G'$ if and only if for all $p' \in P'$ there exists a $p \in P$ such that $p \to_R^* p'$ and for all $p \in P$ there exists a $p' \in P'$ such that $p \to_R^* p'$.

The following two easy lemmas allow us to move rewriting out of the right-hand side of a production and to the end of a derivation, respectively:

Lemma 3.3. *Let t be a term and $[\overline{x} \backslash \overline{s}]$ a substitution. If $s_i \to_R^* s'_i$ for all i, then $t[\overline{x} \backslash \overline{s}] \to_R^* t[\overline{x} \backslash \overline{s'}]$.*

Proof. This follows from the fact that $\to_R^*$ is monotonic and reflexive-transitive. □

Lemma 3.4. *Let σ be a substitution and t, t' terms. If $t \to_R^* t'$, then $t\sigma \to_R^* t'\sigma$.*

Proof. The result is clear for single-step rewritings and then extends to the transitive closure. □

Using these two lemmas, we can now lift rewriting to derivations:

Lemma 3.5. *Let δ, δ' be derivations such that $\delta \to_R^* \delta'$. Then $\alpha_{0,0}\delta \to_R^* \alpha_{0,0}\delta'$.*

Proof. Let $\delta = [\overline{\alpha_0}\backslash\overline{s_0}][\overline{\alpha_1}\backslash\overline{s_1}]\cdots[\overline{\alpha_n}\backslash\overline{s_n}]$ and $\delta' = [\overline{\alpha_0}\backslash\overline{s'_0}][\overline{\alpha_1}\backslash\overline{s'_1}]\cdots[\overline{\alpha_n}\backslash\overline{s'_n}]$. We will now iteratively change the derivation δ to $\delta_i = [\overline{\alpha_0}\backslash\overline{s'_0}]\cdots[\overline{\alpha_{i-1}}\backslash\overline{s'_{i-1}}][\overline{\alpha_i}\backslash\overline{s_i}]\cdots[\overline{\alpha_n}\backslash\overline{s_n}]$ while maintaining the invariant that $\delta \to_R^* \delta_i$. At step i, we rewrite the substitution $[\overline{\alpha_i}\backslash\overline{s_i}]$. First we apply Lemma 3.3:

$$\begin{aligned} & \alpha_{0,0}[\overline{\alpha_0}\backslash\overline{s'_0}]\cdots[\overline{\alpha_{i-1}}\backslash\overline{s'_{i-1}}][\overline{\alpha_i}\backslash\overline{s_i}] \\ \to_R^* \quad & \alpha_{0,0}[\overline{\alpha_0}\backslash\overline{s'_0}]\cdots[\overline{\alpha_{i-1}}\backslash\overline{s'_{i-1}}][\overline{\alpha_i}\backslash\overline{s'_i}]. \end{aligned}$$

And then we apply Lemma 3.4:

$$\begin{aligned} & \alpha_{0,0}[\overline{\alpha_0}\backslash\overline{s'_0}]\cdots[\overline{\alpha_{i-1}}\backslash\overline{s'_{i-1}}][\overline{\alpha_i}\backslash\overline{s_i}][\overline{\alpha_{i+1}}\backslash\overline{s_{i+1}}]\cdots[\overline{\alpha_n}\backslash\overline{s_n}] \\ \to_R^* \quad & \alpha_{0,0}[\overline{\alpha_0}\backslash\overline{s'_0}]\cdots[\overline{\alpha_{i-1}}\backslash\overline{s'_{i-1}}][\overline{\alpha_i}\backslash\overline{s'_i}][\overline{\alpha_{i+1}}\backslash\overline{s_{i+1}}]\cdots[\overline{\alpha_n}\backslash\overline{s_n}]. \end{aligned}$$

At the end $\delta_n = \delta'$ and we have $\delta \to_R^* \delta'$. □

We can now prove that rewriting a grammar changes the generated language by rewriting as well:

Theorem 3.6. *Let $G = \langle \alpha_{0,0}, N, \Sigma, P\rangle$ and $G' = \langle \alpha_{0,0}, N, \Sigma, P'\rangle$ be VTRATGs. If $G \to_R^* G'$, then $L(G) \to_R^* L(G')$.*

Proof. Let $\alpha_{0,0}\delta \in L(G)$. Since $G \to_R^* G'$, there exists a derivation δ' in G' such that $\delta \to_R^* \delta'$. By Lemma 3.5, $\alpha_{0,0}\delta \to_R^* \alpha_{0,0}\delta' \in L(G)$. On the other hand, let $\alpha_{0,0}\delta' \in L(G')$. By a symmetric argument, there exists a δ such that $\alpha_{0,0}\delta \in L(G)$ and $\alpha_{0,0}\delta \to_R^* \alpha_{0,0}\delta'$. Thus, $L(G) \to_R^* L(G')$. □

In contrast to this result on rewriting VTRATGs, there is no corresponding result for regular tree grammars. There are important differences: VTRATGs only produce finite languages and we are concerned with the preservation of size, while for regular tree grammars the question is whether the resulting infinite language can be generated at all. However, the results of Gascón et al. (2009) suggest a correspondence: they show that the language obtained by rewriting a regular tree language can be recognized by a tree automaton with equality and disequality constraints. A VTRATG where every nonterminal vector has length 1 can also be recognized by a tree automaton with equality constraints of similar size.

4 STABLE TERMS

The set of terms L that we would like to cover often has some regularity. We will make use of this regularity to simplify grammars using a rewrite relation $\sqsubseteq_L$ derived from L. This rewrite relation consists of all the transformations that keep every subterm of L intact. We then obtain a characterization of the fully simplified grammars—those are called stable grammars.

Example 4.1. The following set of terms L has some obvious regularities: the first and second arguments of f are always the same, and in addition the third argument is always e:

$$L = \{f(c, c, e), f(d, d, e)\}.$$

This VTRATG $G = \langle \alpha_{0,0}, N, \Sigma, P\rangle$ covers L without making use of these regularities:

$$\begin{aligned} N &= ((\alpha_{0,0}), (\alpha_{1,0}), (\alpha_{2,0}), (\alpha_{3,0})) \\ \Sigma &= \{f/3, c/0, d/0, e/0\} \\ P &= \{(\alpha_{0,0}) \to (f(\alpha_{1,0}, \alpha_{2,0}, \alpha_{3,0})), \\ &\qquad (\alpha_{1,0}) \to (c), (\alpha_{1,0}) \to (d), \\ &\qquad (\alpha_{2,0}) \to (c), (\alpha_{2,0}) \to (d), \\ &\qquad (\alpha_{3,0}) \to (e)\}. \end{aligned}$$

Clearly the right-hand side of the production $(\alpha_{0,0}) \to (f(\alpha_{1,0}, \alpha_{2,0}, \alpha_{3,0}))$ is unnecessarily general; we could have simplified it to $f(\alpha_{1,0}, \alpha_{1,0}, e)$ (thus saving two nonterminals and three productions).

Definition 4.2. Let L be a set of terms and t, t' be any terms. Then $t \sqsubseteq_L t'$ if and only if $t\sigma \in \mathrm{st}(L)$ implies $t\sigma = t'\sigma$ for all substitutions σ. We also define the strict relation $\sqsubset_L$ by: $t \sqsubset_L t'$ if and only if $t \sqsubseteq_L t'$ and $t' \not\sqsubseteq_L t$.

Example 4.3. For $L = \{f(c,c,e), f(d,d,e)\}$, we have $f(x,y,e) \sqsubseteq_L f(x,x,e)$—this expresses the fact that the first two arguments to f are always equal in L. But also, less meaningful statements such as $c \sqsubseteq_L c$ are true, or even $f(f(x,y,e),z,e) \sqsubseteq_L c$. It is *not* the case that $f(x,y,e) \sqsubseteq_L f(c,c,e)$, as $\sigma = [x\backslash d, y\backslash d]$ is a counterexample: we have $f(x,y,e)\sigma = f(d,d,e) \in \mathrm{st}(L)$, but $f(x,y,e)\sigma = f(d,d,e) \neq f(c,c,e) = f(c,c,e)\sigma$.

It follows via routine arguments that $\sqsubseteq_L$ is a preorder and $\sqsubset_L$ a strict partial order.

Lemma 4.4. *Let L be a set of terms, and then the relation $\sqsubseteq_L$ is a rewrite relation.*

Proof. Let s and t be such that $s \sqsubseteq_L t$. For monotonicity, we need to show that $u[s]_p \sqsubseteq_L u[t]_p$ for all u and p. So let σ be such that $u[s]_p\sigma \in \mathrm{st}(L)$. Then clearly $s\sigma \in \mathrm{st}(L)$ as well, so we have $s\sigma = t\sigma$, and thus $u[s]_p\sigma = u[t]_p\sigma$.

To show that $\sqsubseteq_L$ is fully invariant, we need to show that $s\sigma \sqsubseteq_L t\sigma$ for all σ. So let τ be such that $(s\sigma)\tau \in \mathrm{st}(L)$, and then clearly $s(\sigma\tau) \in \mathrm{st}(L)$ and $s\sigma\tau = t\sigma\tau$ by assumption. □

Let $\to \in A \times A$ be a binary relation on a set A. An element t is called a normal form with respect to the relation $\to$ if there is no s such that $t \to s$, that is, if it cannot be reduced using $\to$.

Simplifying a production corresponds to taking normal forms under $\sqsubset_L$. Recall that the set of nonterminals is by definition a subset of the set of variables; in particular, $f(\alpha_{1,0}, \alpha_{2,0}, \alpha_{3,0})$ is not a ground term. The reason we are nevertheless considering more variables than just nonterminals in this section is twofold: first, we need a larger number of variables (since the number of nonterminals is bounded). Second, they are conceptually different—we can always rename the variables in these transformations, while this is not possible in productions.

Definition 4.5. A term t is said to be *stable* if t is in normal form with respect to $\sqsubset_L$. The set $S(L)$ consists of all stable terms.

For a stable term t there exists no term s such that $t \sqsubset_L s$.

Example 4.6. Let $L = \{f(c,c,e), f(d,d,e)\}$, and then the terms $f(z,z,e)$ and c are stable, but $f(x,y,e)$ is not.

Example 4.7. We can now simplify the grammar from Example 4.1: there we had the production $(\alpha_{0,0}) \to (f(\alpha_{1,0}, \alpha_{2,0}, \alpha_{3,0}))$. If we apply $f(x,y,e) \sqsubseteq_L f(x,x,e)$ to the right-hand side of this production, we obtain the production $(\alpha_{0,0}) \to (f(\alpha_{1,0}, \alpha_{1,0}, e))$, as promised.

Lemma 4.8. *Let L be a set of terms and $t \in S(L)$. Then t subsumes a subterm of L.*

Proof. If L is empty, then $s \sqsubseteq_L r$ for any terms s and r. Hence, $S(L) = \emptyset$, a contradiction to $t \in S(L)$. So let L be nonempty, witnessed by $t_0 \in L$. Assume toward a contradiction that t subsumes no subterm of L. Since $t\sigma \notin \mathrm{st}(L)$ for any substitution σ, we have $t \sqsubseteq_L z$, where z is a fresh variable. We also have $z \not\sqsubseteq_L t$, since $z[z\backslash t_0] \neq t[z\backslash t_0]$ even though $z[z\backslash t_0] = t_0 \in L$. Hence, $t \sqsubset_L z$ and $t \notin S(L)$. □

Example 4.9. Looking at $L = \{h(f(c,c,e)), h(f(d,d,e))\}$, we see that $f(z,z,e) \in S(L)$ subsumes $f(c,c,e) \trianglelefteq h(f(c,c,e)) \in L$, and $g(c) \notin S(L)$ does not subsume any subterm of L.

We will use the relation $\sqsubseteq_L$ to simplify VTRATGs that cover L. Recall that when rewriting a grammar, the generated language is rewritten using the same relation by Theorem 3.6. But if we rewrite using $\sqsubseteq_L$, then any term in L remains unchanged:

Lemma 4.10. *Let L be a set of terms, and then $L \subseteq S(L)$.*

Proof. Let $t \in L$. We need to show that $t \in S(L)$ as well. Assume toward a contradiction that $t \sqsubset_L s$, and let σ be the identity substitution. Then $t\sigma = t \in L \subseteq \mathrm{st}(L)$ and hence $t = t\sigma = s\sigma = s$, a contradiction. □

Let $\rightarrow \subseteq A \times A$ be a binary relation. Then an $a \in A$ is called a normal form if there is no $b \in A$ such that $a \rightarrow b$. The relation $\rightarrow$ is called weakly normalizing if for every $a \in A$ there is a normal form $a' \in A$ such that $a \rightarrow^* a'$, where $\rightarrow^*$ denotes the reflexive-transitive closure of $\rightarrow$. We will now show that $\sqsubset_L$ is weakly normalizing, from which we can then conclude that we can rewrite every term into a stable term.

Lemma 4.11. *Let L be a set of terms and $t \sqsubseteq_L s$. If t subsumes a subterm of L, then s subsumes that subterm as well, and* $\mathrm{Vars}(t) \supseteq \mathrm{Vars}(s)$.

Proof. Let σ be a substitution such that $t\sigma = r \in \mathrm{st}(L)$. By definition of $t \sqsubseteq_L s$, we then have $t\sigma = s\sigma = r$. For the second property, assume toward a contradiction that $x \in \mathrm{Vars}(s) \setminus \mathrm{Vars}(t)$. Let $z_1 \neq z_2$ be two distinct variables. Consider the substitutions $\sigma_i = [x \backslash z_i]\sigma$ for $i \in \{1, 2\}$. We have $t\sigma_1 = t\sigma_2 = r \in \mathrm{st}(L)$ but $s\sigma_1 \neq s\sigma_2$, and hence $t\sigma_1 \neq s\sigma_1$ or $t\sigma_2 \neq s\sigma_2$, in contradiction to $t \sqsubseteq_L s$. □

Theorem 4.12. *Let $L \neq \emptyset$ be a set of terms, and then $\sqsubset_L$ is weakly normalizing.*

Proof. Let t be a term. We need to show that there exists a term s such that $t \sqsubseteq_L s$ and $s \not\sqsubset_L r$ for any term r. If t does not subsume a subterm of L, then we have $t \sqsubseteq_L t'$ for any $t' \in L \subseteq S(L) \neq \emptyset$. Hence, assume without loss of generality that t subsumes a subterm $r \in \mathrm{st}(L)$. If t is a normal form of $\sqsubset_L$, then we are done. Otherwise, by Lemma 4.11, for any term s such that $t \sqsubset_L s$, it is the case that s subsumes r as well, and $\mathrm{Vars}(s) \subseteq \mathrm{Vars}(t)$. Since there are only finitely many such terms and $\sqsubset_L$ is transitive and irreflexive, at least one such term s is a normal form of t. □

The following result is crucial to our approach: we want to rewrite grammars into a form that we can effectively search for—namely, those grammars where all productions are stable. Hence, we need to show that we can rewrite every term into a stable term:

Corollary 4.13. *Let $L \neq \emptyset$ be a set of terms and t a term. Then there exists a term $t' \in S(L)$ such that $t \sqsubseteq_L t'$.*

Proof. Take any $\sqsubset_L$-normal form of t. □

5 STABLE GRAMMARS

Being able to rewrite grammars allows us to transform any grammar until all right-hand sides of productions are stable or, seen differently, to transform any grammar into a subgrammar of the grammar consisting of all stable productions—without increasing its size. The resulting subgrammar still covers the input term set: by Theorem 3.6, the language is rewritten with $\sqsubseteq_L$, but by Lemma 4.10, $\sqsubseteq_L$ keeps every $t \in L$ unchanged!

Definition 5.1. Let L be a set of ground terms and $N = (\overline{\alpha_0}, \ldots, \overline{\alpha_i})$ be a sequence of nonterminal vectors. Then the stable grammar $S_{N,L} = \langle \alpha_{0,0}, N, \Sigma, P \rangle$ contains all productions with stable right-hand sides:

$$P = \{\overline{\alpha_i} \rightarrow \overline{s} \mid s_i \in S(L) \text{ for all } i\}.$$

If L is finite, then $S_{N,L}$ has only finitely many productions: by Lemma 4.8, all right-hand sides of productions in $S_{N,L}$ subsume subterms of L, and there are only finitely many terms in $\mathcal{T}(\Sigma \cup N)$ that are smaller than some term in L (comparing the number of positions). Hence, $S_{N,L}$ is a well-defined and finite grammar. In Corollary 6.14, we will even see that $S_{N,L}$ is only polynomially larger than L for fixed N.

Definition 5.2. Let $G_1 = \langle \alpha_{0,0}, N, \Sigma, P \rangle$ and $G_2 = \langle \alpha'_{0,0}, N', \Sigma', P' \rangle$ be VTRATGs. G_1 is a subgrammar of G_2, written $G_1 \subseteq G_2$, if $P \subseteq P'$, $\alpha_{0,0} = \alpha'_{0,0}$, $N = N'$, $\Sigma = \Sigma'$.

We can now prove the main result about the stable grammar $S_{N,L}$:

Theorem 5.3. *Let $G = \langle \alpha_{0,0}, N, \Sigma, P \rangle$ be a VTRATG and L a set of terms such that $L \subseteq L(G)$. Then there exists a VTRATG $G' = \langle \alpha_{0,0}, N, \Sigma, P' \rangle$ such that:*

(1) $G \sqsubseteq_L G'$
(2) $G' \subseteq S_{N,L}$
(3) $|G'| \leq |G|$
(4) $L \subseteq L(G')$

Proof. Let P' be the set of productions that is obtained by $\sqsubseteq_L$-rewriting the right-hand side of each production in P to a stable term; this is possible by Corollary 4.13. We have $G \sqsubseteq_L G'$ and $G' \subseteq S_{N,L}$. Since this rewriting does not increase the number of productions, we also have $|G'| \leq |G|$. Let $t \in L \subseteq L(G)$. By Theorem 3.6, there is a $t' \in L(G')$ such that $t \sqsubseteq_L t'$. But t is in $L \subseteq S(L)$, so $t' = t$. Hence, $t \in L(G')$, and therefore, $L \subseteq L(G')$. □

Corollary 5.4. *Let L be a finite set of terms and N a sequence of nonterminal vectors. Then the VTRATG $S_{N,L}$ contains a subgrammar $H \subseteq S_{N,L}$ of minimal size covering L.*

6 COMPUTING ALL STABLE TERMS

in Section 7, we will minimize the grammar $S_{N,L}$ in order to produce a solution for the Parameterized Language Cover Problem. Hence, we need to compute $S_{N,L}$. Let $N = (\overline{\alpha_0}, \ldots, \overline{\alpha_n})$ be a sequence of nonterminal vectors, and $k_0, \ldots, k_n$ be the lengths of these vectors; then the right-hand side of a production in $S_{N,L}$ may contain up to $k_1 + \cdots + k_n$ different nonterminals.

The right-hand sides of the productions in $S_{N,L}$ are therefore included in the subset of $S(L)$ of terms with at most $k_1 + \cdots + k_n$ variables (recall that nonterminals are variables). In this section, we will show how to compute this subset from L. To this end, we will characterize stable terms as generalizations of the least general generalizations with injective matching. Substitutions are injective if and only if we cannot express one variable in terms of the others:

Lemma 6.1. *Let σ be a substitution. Then σ is injective on $\mathcal{T}(\Sigma \cup X)$ if and only if $u\sigma \neq x\sigma$ for all variables $x \in X$ and terms u such that* $\mathrm{Vars}(u) \subseteq X \setminus \{x\}$.

Proof. If there are u and x such that $u\sigma = x\sigma$, then clearly σ is not injective. For the converse, we prove that $t\sigma = s\sigma$ implies $t = s$ by induction on t: if t is a variable, we distinguish three cases: the first case is $t \notin \mathrm{Vars}(s)$, and here we have a contradiction to the assumption. The second (trivial) case is $t = s$. In the third case, both $t \neq s$ and $t \in \mathrm{Vars}(s)$, and then t is a strict subterm of s, a contradiction to $t\sigma = s\sigma$. If s is a variable, a symmetric argument applies.

If now t and s are both functions, we have two cases: first, if t and s share the same root symbol, then $t|_i\sigma = s|_i\sigma$ for all arguments, and hence $t|_i = s|_i$ by the inductive hypothesis, and $t = s$ by congruence. Second, if t and s have different root symbols, then already $t\sigma = s\sigma$ is a contradiction. □

The subsumption relation $\preceq$ is a preorder on the set of all terms $\mathcal{T}(\Sigma \cup X)$. Every preorder induces an equivalence relation where $t \approx s$ if and only if $t \preceq s$ and $s \preceq t$. In the subsumption order,

two terms are equivalent if they differ only by variable renaming. If t, s are terms without common variables, and σ their most general unifier, then $t \vee s = t\sigma = s\sigma$ is their supremum. The dual operation is called least general generalization (sometimes also "anti-unifier") and was introduced independently in Plotkin (1970), Plotkin (1971), and Reynolds (1970).

Definition 6.2. The least general generalization $t \wedge s$ of two terms t and s can be computed recursively, where $x(t, s)$ is a different variable for each pair (t, s):

$$f(t_1, \ldots, t_n) \wedge f(s_1, \ldots, s_n) = f(t_1 \wedge s_1, \ldots, t_n \wedge s_n)$$
$$f(t_1, \ldots, t_n) \wedge g(s_1, \ldots, s_m) = x(f(t_1, \ldots, t_n), g(s_1, \ldots, s_m)) \quad \text{if } f \neq g.$$

Example 6.3. $f(c, c, e) \wedge f(d, d, e) = f(x, x, e)$.

Note that the least general generalization is only unique up to variable renaming, and we will often choose the variables to be fresh. As an infimum of terms in the subsumption preorder, the least general generalization satisfies $t \wedge s \approx s \wedge t$, $(t \wedge s) \wedge r \approx t \wedge (s \wedge r)$, and also $t \wedge s \preceq t$ for all terms t, s, r. Since the least general generalization is associative-commutative, we will also write $\bigwedge L$ for the least general generalization of a set of terms L.

Definition 6.4. Let $t, s \in \mathcal{T}(\Sigma \cup X)$ be terms such that t subsumes s. We define s/t as the unique substitution such that $t(s/t) = s$, and $x(s/t) = x$ for all $x \in X \setminus \mathrm{Vars}(t)$.

The substitution s/t is often called a matching from t to s. With this definition, we can now proceed to characterize the stable terms: in Theorem 6.6, we will show that $t \in S(L)$ if and only if the matching to $\bigwedge L_t$ is injective on $\mathcal{T}(\Sigma \cup \mathrm{Vars}(t))$, where L_t is the set of subterms of L that t subsumes. We will then prove an even stronger result: it suffices to only consider bounded subsets of L_t, where the bound only depends on the number of variables in t. Since there are only polynomially many such bounded subsets, we will be able to effectively use this characterization to compute the stable terms with a bounded number of variables in Theorem 6.13.

Lemma 6.5. *Let $T \subseteq \mathcal{T}(\Sigma)$ be a nonempty set of ground terms and $r, s \in \mathcal{T}(\Sigma \cup X)$. Let $u = \bigwedge T$, and for every $t \in T$, let $\pi_t := t/u$. If $r\pi_t = s\pi_t$ for all $t \in T$, then $r = s$.*

Proof. By induction on r. Similar to Lemma 6.1, the critical case is when r is a variable and $r \notin \mathrm{Vars}(s)$. If $r \notin \mathrm{Vars}(u)$, then $r = r\pi_t$ for all t and hence s is necessarily a variable as well. When additionally $s \in \mathrm{Vars}(u)$, we can choose two terms t and t' such that $s\pi_t \neq s\pi_{t'}$, a contradiction to $s\pi_t = s\pi_{t'} = r$. Otherwise, $s \notin \mathrm{Vars}(u)$ and $s = s\pi_t = r\pi_t = r$ with any t.

Assume on the other hand that $r \in \mathrm{Vars}(u)$. We already handled the case that s is a variable, so we can assume that $s = f(s_1, \ldots, s_n)$. Choose terms t and t' such that $r\pi_t$ and $r\pi_{t'}$ have a different root function symbol. One of the root symbols is different from f, a contradiction. □

Theorem 6.6. *Let L be a set of ground terms and t a term that subsumes a subterm of L, and assume $\mathrm{Vars}(t) \cap \mathrm{Vars}(\bigwedge L_t) = \emptyset$. Then $t \in S(L)$ if and only if $\sigma = (\bigwedge L_t)/t$ is injective on $\mathcal{T}(\Sigma \cup \mathrm{Vars}(t))$.*

Proof. First, we show that $t \in S(L)$, assuming that σ is injective. Let s be any term such that $t \sqsubseteq_L s$. For every $r \in L_t$ let $\pi_r := r/\bigwedge L$. We have $t\sigma\pi_r = r \in \mathrm{st}(L)$, and $t\sigma\pi_r = s\sigma\pi_r$ by assumption. By Lemma 6.5, we obtain $t\sigma = s\sigma$, and hence $t = s$ by injectivity of σ, and thus $t \not\sqsubset_L s$ and $t \in S(L)$.

In order to show that σ is injective, we apply Lemma 6.1 and have x, u such that $x\sigma = u\sigma$ and $x \in \mathrm{Vars}(t) \setminus \mathrm{Vars}(u)$. First, we have $t[x\backslash u] \not\sqsubseteq_L t$ because of Lemma 4.11. We will now show that $t \sqsubseteq_L t[x\backslash u]$, which contradicts $t \in S(L)$ since $t[x\backslash u] \not\sqsubseteq_L t$. Let τ be a substitution such that $t\tau \in \mathrm{st}(L)$, and then $t\tau \in L_t$ as well, and there exists a substitution ρ such that $\tau = \sigma\rho$. We can now compute $t[x\backslash u]\tau = t[x\backslash u]\sigma\rho = t\sigma\rho = t\tau$, where $[x\backslash u]\sigma = \sigma$ because the variables in u and the domain of σ are disjoint. □

Theorem 6.7. *Let L be a set of ground terms, and $t \in S(L)$. Then there exists a subset $L' \subseteq L_t$ such that $\sigma' = (\bigwedge L')/t$ is injective on $\mathcal{T}(\Sigma \cup \mathrm{Vars}(t))$ and $|L'| \leq |\mathrm{Vars}(t)| + 1$.*

Proof. We construct L' in stages: we will define a sequence of $L_0 \subseteq L_1 \subseteq \cdots \subseteq L_n = L'$ such that $|L_i| \leq i + 1$. In each step, we have a substitution $\sigma_k = (\bigwedge L_k)/t$, and in the end $\sigma' = \sigma_n$. First, pick a term $s_0 \in L_t$ and set $L_0 = \{s_0\}$, and then we have $t\sigma_0 = s_0$. We can now order the variables $x_1, \ldots, x_n$ in t in such a way that $x_i\sigma_0 \lhd x_j\sigma_0$ implies $i < j$.

By Lemma 6.1, it will suffice to show that $x_i\sigma' \neq r\sigma'$ for any x_i and r such that $x_i \notin \mathrm{Vars}(r)$. Note that due to symmetry, we can assume without loss of generality that if $r = x_j$ is a variable as well, then $j < i$. Furthermore, if the disequality $x_i\sigma_k \neq r\sigma_k$ already holds for some σ_k, then it also holds for σ' since $\sigma_k = \sigma'(\bigwedge L_k / \bigwedge L')$.

In step i, we now ensure that there is no r such that $x_i\sigma_i = r\sigma_i$ and $x_i \notin \mathrm{Vars}(r)$, and $j < i$ in the case that $r = x_j$ is a variable. Assume that there is such an r with $x_i\sigma_{i-1} = r\sigma_{i-1}$ (otherwise, we can set $L_i = L_{i-1}$). With these restrictions, we can show that this r is unique: let $r' \neq r$ be another such term, and then we have $r\sigma_{i-1} = r'\sigma_{i-1}$. Similar to Lemma 6.1, we can assume that at least one of r or r' is a variable. If $r = x_j$ and $r' = x_k$ are both variables, then without loss of generality $j < k < i$, and hence $x_k\sigma_{i-1} = x_j\sigma_{i-1}$ is a contradiction to the inductive hypothesis. If r is a function and $r' = x_j$ is a variable (or vice versa), then $x_j\sigma_{i-1} = r\sigma_{i-1}$ is also a contradiction.

Now there is a unique r such that $x_i\sigma_{i-1} = r\sigma_i$ with the previous restrictions. We have $t \not\trianglelefteq_L t[x_i \backslash r]$ because of $t \in S(L)$ and Lemma 4.11. Hence, there exists a substitution τ such that $t\tau \in \mathrm{st}(L)$ and $t\tau \neq t[x_i \backslash r]\tau$. Furthermore, $t\tau \in L_t$ and $x_i\tau \neq r\tau$. Set $L_i = L_{i-1} \cup \{t\tau\}$. Now $\tau = \sigma_i(t\tau / \bigwedge L_i)$ and hence $x_i\sigma_i \neq r\sigma_i$. □

Example 6.8. Consider $L = \{f(g(a), g(a), e), f(g(b), g(b), e), f(g(c), g(c), e)\}$. The term $t = f(x, x, e)$ is in $S(L)$, and hence the substitution $\sigma = [x \backslash g(y)]$ is injective, where $t\sigma = \bigwedge L_t = f(g(y), g(y), e)$. But since t has only one free variable, there is a subset $L' \subseteq L_t$ with the same property and at most $1 + 1$ elements: for example, $L' = \{f(g(c), g(c), e), f(g(d), g(d), e)\}$. The ground term $t_2 = f(g(c), g(c), e)$ has zero variables, and hence there is a subset $L'_2 \subseteq L$ with at most $0 + 1$ elements such $t_2\sigma'_2 = \bigwedge L'_2$. This set L'_2 is necessarily the singleton $L'_2 = \{t_2\}$.

Our strategy for computing stable terms will be to compute all terms t such that σ is injective, where $t\sigma = L'$ for some subset $L' \subseteq \mathrm{st}(L)$. We enumerate all subsets of $\mathrm{st}(L')$ of the bounded size given by Theorem 6.7, and then infer the stable term t from L' by generalization. However, in general, there exists more than one term t that has an injective substitution σ satisfying $t\sigma = \bigwedge L'$: for example, with $L' = \{f(f(c)), f(f(d))\}$, all of the terms x, $f(x)$, and $f(f(x))$ have an injective substitution to $\bigwedge L'$.

Let $m : \mathcal{T}(X \cup \Sigma) \to X$ be a partial function from terms to variables. Then $R_m : \mathcal{T}(X \cup \Sigma) \to \mathcal{T}(X \cup \Sigma)$ denotes the replacement function that replaces all occurrences of the terms in the domain of m by the corresponding variables.

Example 6.9. Recall that the terms x, $f(x)$, and $f(f(x))$ are all in $S(L)$, where $L = \{f(f(c)), f(f(d))\}$. Let $m_1 = \{f(f(y))) \mapsto x\}$, $m_2 = \{f(y) \mapsto x\}$, and $m_3 = \{y \mapsto x\}$. If we consider the least general generalization $\bigwedge L = f(f(c)) \wedge f(f(d)) = f(f(y))$, then we can obtain the three stable terms from it using R_{m_i}: $R_{m_1}(\bigwedge L) = x$, $R_{m_2}(\bigwedge L) = f(x)$, and $R_{m_3}(\bigwedge L) = f(f(x))$.

If a partial function m is injective, its inverse $m^{-1} = \{(s \mapsto t) \mid (t \mapsto s) \in m\}$ is a partial function as well. If $m : X \to Y$ is a partial function and $Z \subseteq X$ is a subset of its domain, then $m \restriction Z = \{(s \mapsto t) \mid (s \mapsto t) \in m \wedge s \in Z\}$ is the restriction of m to Z.

Since terms are equivalent modulo variable renaming in the subsumption order, in the following lemmas we will assume without loss of generality that the variables in the least general

generalization $\bigwedge L$ are distinct from the variables in X. The following lemma now shows how least general generalizations and stable terms relate:

Lemma 6.10. *Let $k \in \mathcal{T}(\Sigma \cup X)$ and $k\sigma = \bigwedge L$, where σ is injective on $\mathcal{T}(\Sigma \cup X)$. Then $k = R_{(\sigma \restriction X)^{-1}}(\bigwedge L)$.*

Proof. Follows homomorphically from $R_{(\sigma \restriction \mathrm{st}(k))^{-1}}(r\sigma) = r$ for all $r \in \mathrm{st}(k) \cap X$. □

Example 6.11. Let $L = \{f(g(c)), f(g(d))\}$, $\bigwedge L = f(g(y))$, and $k = f(x)$. Then $k\sigma = \bigwedge L$ for $\sigma = [x \backslash g(y)]$ and indeed $R_{\{g(y) \mapsto x\}}(f(g(y))) = f(x)$.

We now have enough constraints on stable terms to enumerate all of them (with a given bound on the number of variables) in polynomial time by simply applying all possible replacements to all least general generalizations. However, this would also produce many terms that are not stable if we use replacements that correspond to noninjective substitutions. For example, take $L = \{f(c, e), f(d, e)\}$, $k = f(x, e) = \bigwedge L \preceq L$, and $X = \{y, z\}$. Using the replacement $m = (\sigma \restriction X)^{-1} = \{x \mapsto y, e \mapsto z\}$, we can obtain the term $R_m(f(x, e)) = f(y, z)$, which is not in $S(L)$.

Hence, it is important to check that the substitutions are injective:

Lemma 6.12. *Let σ be a substitution and X a set of variables. Then we can decide in polynomial time whether σ is injective on $\mathcal{T}(\Sigma \cup X)$.*

Proof. We define a binary predicate s on variables and terms:

$$s(x, t) :\leftrightarrow \exists u \in \mathcal{T}(\Sigma \cup X \setminus \{x\}),\ u\sigma = t.$$

This predicate can be computed recursively:

$$s(x,t) = \begin{cases} \top & \text{if } y\sigma = t \text{ for some } y \neq x \\ s(x,t_1) \wedge \cdots \wedge s(x,t_n) & \text{if } t = f(t_1, \ldots, t_n) \\ \bot & \text{otherwise.} \end{cases}$$

By Lemma 6.1, σ is injective if and only if $\forall x \in X\ \neg s(x, x\sigma)$. The runtime of s is quadratic in the size of t and X, and we iterate it for every $x \in X$; hence, it has polynomial runtime in X and σ. □

We can now give the algorithm that computes the stable terms and prove its correctness. The runtime of the algorithm depends on the symbolic complexity $|L|_s$ of the set of terms L: the symbolic complexity is the sum of the number of positions of each term in L.

Theorem 6.13. *Let L be a set of ground terms and X a set of variables, and then the set of all terms $k \in S(L)$ such that $\mathrm{Vars}(k) \subseteq X$ can be computed as follows:*

(1) For each subset $L' \subseteq \mathrm{st}(L)$ such that $|L'| \leq |X| + 1$:
(2) Compute $\bigwedge L'$.
(3) For each injective partial function $m : \mathrm{st}(\bigwedge L') \to X$:
(4) Check that m^{-1} is injective on $\mathcal{T}(\Sigma \cup X)$.
(5) Then output $k = R_m(\bigwedge L')$.

The runtime of this procedure is bounded by a polynomial in $|L|_s$ for fixed $|X|$.

Proof. Let us first show the correctness of the algorithm, that is, that it does indeed generate exactly the terms $k \in S(L)$ such that $\mathrm{Vars}(k) \subseteq X$. Assume that k is such a term. Then, by Theorem 6.6, there exists an injective substitution σ such that $k\sigma = \bigwedge L_k$. By Theorem 6.7, we then have an $L' \subseteq L_k$ such that $|L'| \leq |k| + 1 \leq |X| + 1$ and the substitution σ' such that $k\sigma' = \bigwedge L'$ is injective as well. By Lemma 6.10, the injective partial function $m := (\sigma \restriction \mathrm{st}(k))^{-1}$

satisfies $k = R_m(\bigwedge L')$. Hence, the algorithm outputs k. On the other hand, every term in the output is in $S(L)$ by Theorem 6.6.

In step (1), there are at most $\binom{\mathrm{st}(L)}{|X|+1} = O(|L|_s^{|X|+1})$ possible subsets; in step (3), there are at most $|L|_s$ positions in $\bigwedge L'$ and hence at most $O(|L|_s^{|X|+1})$ partial functions σ. Computing least general generalizations and replacements is linear in the input, and checking injectivity is also polynomial due to Lemma 6.12; therefore, the total runtime is polynomial in $|L|_s$. □

Corollary 6.14. *Let a sequence of nonterminals N be fixed. Then the grammar $S_{N,L}$ is polynomial-time computable from a finite set of ground terms L.*

Proof. apply Lemma 6.13 to $X = N$. □

7 MINIMIZATION

In Corollary 6.14, we have produced a polynomial-time computable VTRATG $S_{N,L}$ that is guaranteed to contain a subgrammar H covering L of minimal size. In particular, this subgrammar H solves the Parameterized Language Cover Problem for L and N. Since we can efficiently compute $S_{N,L}$, we have reduced the Parameterized Language Cover Problem to the Grammar Minimization Problem:

Problem 7.1 (Grammar Minimization Problem). Given a VTRATG G and a finite set of ground terms $L \subseteq L(G)$, find a subgrammar $H \subseteq G$ of minimal size such that $L \subseteq L(H)$.

We will reduce this problem to MaxSAT by giving a propositional formula expressing the property that the subgrammar H covers L. MaxSAT is an optimization variant of the Boolean satisfaction problem (SAT), for which a number of efficient off-the-shelf solvers exist; see the yearly MaxSAT competition (Argelich et al. 2008) for a list of solvers. In this article, we only consider the partial and unweighted variant of MaxSAT and simply call it MaxSAT:

Problem 7.2 (MaxSAT). Given two sets of propositional clauses H and S (so-called "hard" and "soft" clauses), find an interpretation I such that $I \models H$ and I maximizes the number of satisfied clauses in S.

We will encode "H covers L" by stating for each $t \in L$ that "there exists a derivation δ_t of t in H." The concrete encoding of "δ_t is a derivation of t in H" is based on a sparse encoding of the function $\alpha_{i,j} \mapsto \alpha_{i,j}\delta_t$, that is, encoding the function as a binary relation. One important observation about this function is that it usually returns only subterms of t. For example, consider the following derivation:

$$\delta = [\alpha_{0,0} \backslash f(\alpha_{1,0}, \alpha_{1,0})][\alpha_{1,0} \backslash d].$$

In this case, $t = \alpha_{0,0}\delta = f(d, d)$, and $\alpha_{1,0}\delta = d$ is indeed a subterm of t. However, this subterm property can fail in the presence of "unused" nonterminals, for example:

$$\delta_2 = [\alpha_{0,0} \backslash f(\alpha_{1,0}, \alpha_{1,0})][\alpha_{1,0} \backslash d, \alpha_{1,1} \backslash c].$$

Again, $t = \alpha_{0,0}\delta_2 = f(d, d)$ and $\alpha_{1,0}\delta_2 = d$ are subterms, but now $\alpha_{1,1}\delta_2 = c$ is not a subterm of $f(d, d)$. If $\alpha_{i,j}\delta$ is not a subterm of t, it will turn out to be irrelevant to the derivation, and hence we will ignore it and assign the dummy term $\bot$ to all such terms that are not subterms of t. This allows us to consider the smaller range $\mathrm{st}(t) \cup \{\bot\}$ for the function δ.

Definition 7.3 (Propositional encoding for $t \in L(H)$). Let G be a VTRATG and t a ground term. We use the following atoms:

— $\alpha_{i,j}\delta_t = r$, where $\alpha_{i,j}$ is a nonterminal and $r \in \mathrm{st}(t) \cup \{\bot\}$ a ground term.
— $p \in P'$, where $p \in P$ is a production.

We define the following abbreviations for formulas:

$$\overline{\alpha_i}\delta_t = \overline{r} \equiv \bigwedge_j \alpha_{i,j}\delta_t = r_j.$$

$$\mathbf{Match}_{G,t}(u,s) \equiv \begin{cases} \top & \text{if } u = \bot \\ \beta_1\delta_t = r_1 \wedge \cdots \wedge \beta_k\delta = r_k & \text{if } s[\beta_1 \backslash r_1, \ldots, \beta_k \backslash r_k] = u \\ \bot & \text{otherwise.} \end{cases}$$

$$\mathbf{Case}_{G,t}(i,\overline{u}) \equiv \overline{\alpha_i}\delta_t = \overline{u} \rightarrow \bigvee_{\alpha_i \rightarrow \overline{s} \in P} \left(\overline{\alpha_i} \rightarrow \overline{s} \in P' \wedge \bigwedge_j \mathbf{Match}_{G,t}(u_j, s_j) \right).$$

$$\mathbf{Func}_{G,t} \equiv \bigwedge_{i,j} \left(\bigvee_{s \in \mathrm{st}(t) \cup \{\bot\}} \alpha_{i,j}\delta_t = s \wedge \bigwedge_{s_1 \neq s_2 \in \mathrm{st}(t) \cup \{\bot\}} \neg(\alpha_{i,j}\delta_t = s_1 \wedge \alpha_{i,j}\delta_t = s_2) \right).$$

$$\mathbf{GenTerm}_{G,t} \equiv \alpha_{0,0}\delta_t = t \wedge \mathbf{Func}_{G,t} \wedge \bigwedge_i \bigwedge_{\overline{s} \in (\mathrm{st}(L) \cup \bot)^{k_i}} \mathbf{Case}_{G,t}(i,\overline{s}).$$

The formula $\mathbf{Match}_{G,t}(u,s)$ encodes $s\delta_t = u$, extending the $\alpha_{i,j}\delta_t = u$ atom to arbitrary terms s. In order to ensure the correctness of the whole encoding, we have to add implied constraints for each possible function value of δ_t and for each nonterminal vector $\overline{\alpha_i}$: the $\mathbf{Case}_{G,t}(i,\overline{u})$ formula encodes these constraints. Then $\mathbf{Func}_{G,t}$ states that $\delta : N \rightarrow \mathrm{st}(t) \cup \{\bot\}$ is a function, and $\mathbf{GenTerm}_{G,t}$ combines the other formulas to encode that δ is a derivation of t using only the productions from the subset $P' \subseteq P$, that is, those that are present in H.

Example 7.4. Consider $t = f(c,c,e)$, $N = \{((\alpha_{0,0}), (\alpha_{1,0})\}$, and $G = \langle \alpha_{0,0}, N, \Sigma, P \rangle$ with the following productions $P = \{p_1, p_2, p_3, p_4\}$:

$$p_1 = (\alpha_{0,0}) \rightarrow (f(\alpha_{1,0}, \alpha_{1,0}, \alpha_{1,0})) \qquad p_3 = (\alpha_{1,0}) \rightarrow (c)$$
$$p_2 = (\alpha_{0,0}) \rightarrow (f(\alpha_{1,0}, \alpha_{1,0}, e)) \qquad p_4 = (\alpha_{1,0}) \rightarrow (d).$$

Then we encode $t \in L(H)$ using the following formula $\mathbf{GenTerm}_{G,t}$ (slightly simplified propositionally):

$$\begin{aligned}
\mathbf{Case}_{G,t}(0, f(c,c,e)) &\equiv \alpha_{0,0}\delta_t = f(c,c,e) \rightarrow (p_2 \in P' \wedge \alpha_{1,0}\delta_t = c) \\
\mathbf{Case}_{G,t}(0,c) &\equiv \alpha_{0,0}\delta_t = c \rightarrow \bot \\
\mathbf{Case}_{G,t}(0,e) &\equiv \alpha_{0,0}\delta_t = e \rightarrow \bot \\
\mathbf{Case}_{G,t}(0,\bot) &\equiv \alpha_{0,0}\delta_t = \bot \rightarrow \top \\
\mathbf{Case}_{G,t}(1, f(c,c,e)) &\equiv \alpha_{1,0}\delta_t = f(c,c,e) \rightarrow \bot \\
\mathbf{Case}_{G,t}(1,c) &\equiv \alpha_{1,0}\delta_t = c \rightarrow (p_3 \in P') \\
\mathbf{Case}_{G,t}(1,e) &\equiv \alpha_{1,0}\delta_t = e \rightarrow \bot \\
\mathbf{Case}_{G,t}(1,\bot) &\equiv \alpha_{1,0}\delta_t = \bot \rightarrow \top \\
\mathbf{GenTerm}_{G,t} &\equiv \alpha_{0,0}\delta_t = f(c,c,e) \wedge \mathbf{Func}_{G,t} \wedge \bigwedge_{i,s} \mathbf{Case}_{G,t}(i,s).
\end{aligned}$$

Many of the formulas of $\mathbf{Case}_{G,t}(i,s)$ are of the form $(\cdots \rightarrow \bot)$; these correspond to derivations that are impossible: for example, we have $\alpha_{0,0}\delta_t = c \rightarrow \bot$ because there is no production from $(\alpha_{0,0})$ that has c as the root symbol. In this example, $\mathbf{GenTerm}_{G,t}$ entails $p_2 \in P' \wedge p_3 \in P'$, and hence, any covering subgrammar $H \subseteq G$ necessarily includes these two productions; the minimal covering subgrammar consists precisely of these two productions.

Lemma 7.5. *Let $G = \langle \alpha_{0,0}, N, \Sigma, P\rangle$ be a VTRATG, $H = \langle \alpha_{0,0}, N, \Sigma, P'\rangle \subseteq G$ a subgrammar of G, and t a term. Then, for every derivation δ of t in H, there exists a satisfying interpretation $I \models \mathbf{GenTerm}_{G,t}$ such that for all $p \in P$, $I \models p \in P'$ if and only if $p \in P'$.*

Conversely, if $I \models \mathbf{GenTerm}_{G,t}$ is a satisfying interpretation such that for all $p \in P$, $I \models p \in P'$ if and only if $p \in P'$, then there exists a derivation of t in H.

Proof. We need to construct a satisfying interpretation I for every derivation δ in H. So we would like to set $I \models \alpha_{i,j}\delta_t = r$ if and only if $\alpha_{i,j}\delta = r$; but this could fail if r is not a subterm of t. But the following assignment only results in $\bot$ or subterms of t, as required:

$$I \models \alpha_{i,j}\delta_t = r \quad \Leftrightarrow \quad \begin{cases} r = \alpha_{i,j}\delta & \text{if } \alpha_{i,j} \text{ is a subterm of } t \\ r = \bot & \text{otherwise.} \end{cases}$$

It remains to verify that $I \models \mathbf{Case}_{G,t}(i, \overline{s})$ for all i and $\overline{s}$, that is, $I \models \mathbf{Match}_{G,t}(\alpha_{i,j}\delta_t, s_{i,j})$ for all j, where $\overline{\alpha_i} \to \overline{s_i}$ is the chosen production in δ. This is trivial if $\alpha_{i,j}\delta$ is not a subterm of t—then $\mathbf{Match}_{G,t}(\alpha_{i,j}\delta_t, s_{i,j})$ expands to $\top$; in the other case, it is clear from the definition of a derivation.

Conversely, let I be a satisfying interpretation for $\mathbf{GenTerm}_{G,t}$ such that $I \models p \in P'$ if and only if $p \in P'$. We will now construct an actual derivation δ in H such that $\alpha_{0,0}\delta = t$. By $\mathbf{Func}_{G,t}$, there is a unique value for each $\alpha_{i,j}\delta_t$ in I; let $t_{i,j} \in \mathrm{st}(t) \cup \{\bot\}$ be terms such that $I \models \alpha_{i,j}\delta_t = t_{i,j}$. Choose the productions $\overline{\alpha_i} \to \overline{s_i}$ for δ such that $I \models \overline{\alpha_i} \to \overline{s_i} \wedge \bigwedge_j \mathbf{Match}_{G,t}(t_{i,j}, s_{i,j})$. This immediately ensures that δ really is a derivation in the subgrammar.

We will now verify that $t_{i,j} \neq \bot$ implies $\alpha_{i,j}\delta = t_{i,j}$ for all i and j by backward induction on i: if $t_{i,j} \neq \bot$, then $\mathbf{Match}_{G,t}(t_{i,j}, s_{i,j}) \equiv \bigwedge_l \beta_l\delta = r_l$. Because the nonterminals in $s_{i,j}$ can only be of the form $\alpha_{i',j'}$ with $i' > i$, computing $\alpha_{i,j}\delta$ first substitutes $\alpha_{i,j}$ with $s_{i,j}$, and then each β_l with r_l, since $\beta_l\delta = r_l$ per induction hypothesis. Now since $t_{0,0} = t \neq \bot$, we conclude that δ is a derivation of $\alpha_{0,0}\delta = t$. □

So far we have only considered the encoding for the derivability of a single term t; we will now turn to derive multiple terms.

Theorem 7.6. *The Grammar Minimization Problem is polynomial-time reducible to MaxSAT.*

Proof. Let $G = \langle \alpha_{0,0}, N, \Sigma, P\rangle$ be the VTRATG, and $L \subseteq L(G)$ the set of terms. We need to find a subset $P' \subseteq P$ of minimal size such that the grammar $H = \langle \alpha_{0,0}, N, \Sigma, P'\rangle$ still satisfies $L \subseteq L(H)$. By Lemma 7.5, any interpretation I satisfying $\bigwedge_{t \in L} \mathbf{GenTerm}_{G,t}$ corresponds to a covering VTRATG. Set the hard clauses of the MaxSAT problem to a CNF of this formula. (Such a CNF can be obtained in polynomial time as shown in Tseitin (1983).)

For each production $p \in P$, we add the soft clause $\neg(p \in P')$. The number of satisfied soft clauses is then exactly $|P| - |P'|$, the number of productions not included in the minimized VTRATG. Maximizing the number of soft clauses then minimizes the number of productions in the VTRATG given by the interpretation. From a solution I to this MaxSAT problem, we obtain the minimal covering VTRATG by setting $P' = \{p \mid I \models p \in P'\}$, which is the corresponding subgrammar according to Lemma 7.5. □

8 EXPERIMENTAL RESULTS

We have evaluated our implementation of Algorithm 1 on a real-world collection of 19,104 term sets, called the TSTP dataset. If we recall our proof-theoretic motivation for the Parametric Language Cover Problem, we extract term sets from cut-free proofs, and if we can find a covering grammar, then we can introduce cuts or lemmas into the cut-free proof. In this fashion, we have extracted term sets from the TSTP collection of proofs produced by automated theorem provers (Thousands of Solutions from Theorem Provers (Sutcliffe 2010)). Unfortunately, many of these

ALGORITHM 1: Solution of the Parameterized Language Cover Problem via MaxSAT

Input: Set of ground terms L and a sequence of nonterminals N.
Output: Minimal VTRATG H with nonterminals N such that $L \subseteq L(H)$.

G = stableGrammar(N, L)
φ = minimizationFormula(G, L)
I = maxSatSolver(φ, softClauses(G))
H = grammarFromAssignment(I)

proofs are in custom formats that we have been unable to import. However, from the total 137,989 proofs in the TSTP, we could import term sets from 36,494 proofs (26.45%). Of these 36,494 term sets, 17,390 (47.65%) cannot be compressed using VTRATGs as each term has a different root symbol. The remaining 19,104 term sets are the TSTP dataset.

The performance of Algorithm 1 is compared to the so-called delta-table algorithm (see Hetzl et al. (2014b)), which is to our knowledge the only other algorithm that can produce VTRATGs. The delta-table algorithm has two major limitations: First, it only produces grammars with nonterminals $N = ((\alpha_{0,0}), \overline{\alpha}_1)$. Depending on a parameter of the delta-table algorithm, either $|\alpha_1| = 1$ or $|\alpha_1|$ is determined by the algorithm. Since Algorithm 1 requires an upfront choice of N, we will compare both algorithms with the only common parameter setting, that is, $|\alpha_1| = 1, N = ((\alpha_{0,0}), (\alpha_{1,0}))$. The other limitation is that the delta-table algorithm does not always produce minimal grammars.

We compared the implementation of these two algorithms in a prerelease version of GAPT[1] 2.0 (Ebner et al. 2016) and used GNU parallel (Tange 2011) for scheduling. The MaxSAT solver we used is OpenWBO version 1.3.0 (Martins et al. 2014). The performance comparison was conducted on a Debian Linux system with an Intel i5-4570 CPU and 8GiB RAM.

Within a timeout of 60 seconds per term set, the delta-table algorithm finds grammars for 6,873 term sets. Algorithm 1 improves on this and finds grammars for 7,503 term sets. This is a significant improvement, since the additional 630 term sets are the most difficult instances solved, even though they only account for a 9% increase in solutions. Figure 1 shows a cactus plot illustrating the CPU runtime of both algorithms: a cactus plot[2] shows all the term sets that can be compressed within a given timeout; the runtimes are sorted, and beginning from left to right, the runtime for the nth fastest term set is plotted on the y-axis.

For easy term sets—those that take a short time to compress—the delta-table algorithm is faster; there the curve for the delta-table algorithm is below the one for Algorithm 1. As far as we can tell, the performance difference on the easy examples is due to the overhead of constructing the stable grammar and the minimization formula. On average, this reduction makes up for 72% of the runtime for term sets where the total runtime is less than 1 second, whereas the situation is almost reversed for term sets where the total runtime is more than 10 seconds: there about 69% of the runtime is spent inside the MaxSAT solver.

This dichotomy between easy and hard problems is apparent not only on the aggregate dataset but also for individual term sets: Figure 2 shows the CPU runtime ratio (x-axis) in comparison to the compression ratio (y-axis) and total runtime (size of the points). On the top right there is a cluster of small points: these are the easy problems, and while the delta-table algorithm is faster by a factor of 10, both algorithms can solve the problems in a reasonable time. Then there is a large

[1]The implementation is open source and available at `https://logic.at/gapt`.
[2]Cactus plots have been popularized by the SAT community to visualize the performance of different solvers on a benchmark set, and have also been adopted by competitions in other communities.

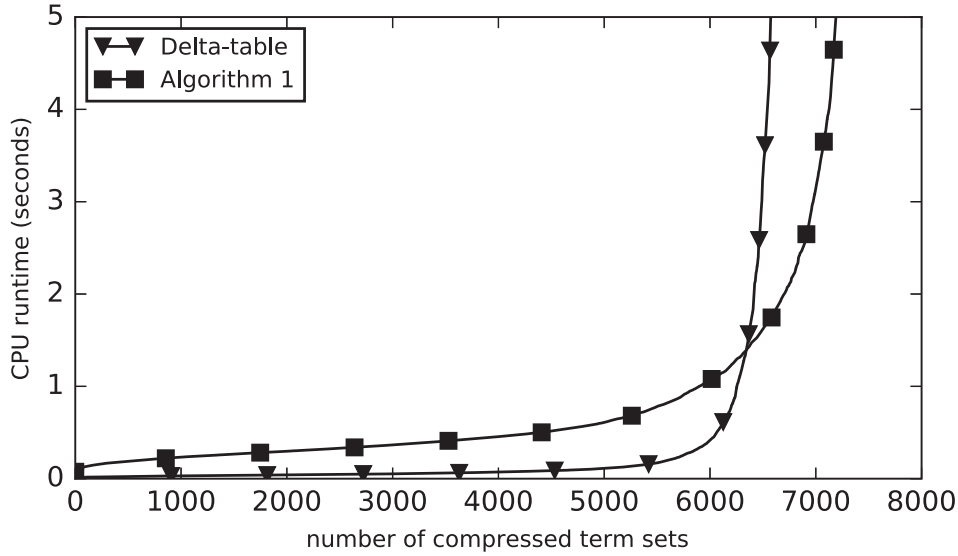

Fig. 1. Runtime of the delta-table algorithm and Algorithm 1 on the TSTP dataset.

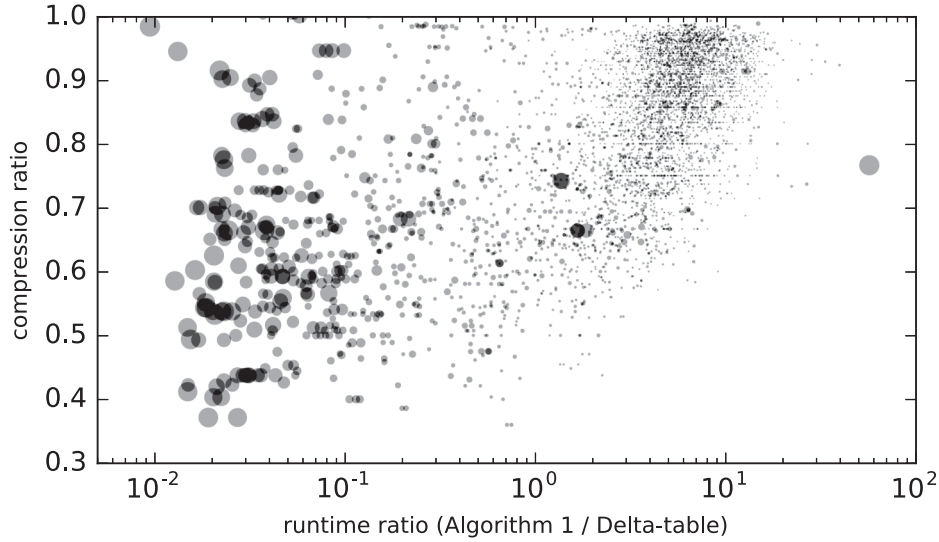

Fig. 2. Comparison of the performance difference to the compression ratio and overall runtime. Each point represents an input term set from the TSTP dataset. The size of a point is proportional to the combined runtime of both algorithms.

cluster on the left: these are the hard problems, which generally admit a better compression, and here Algorithm 1 is faster by a factor of 100.

In Figure 3, we then see the extent of how many more grammars are found by Algorithm 1: while it was previously infeasible with the delta-table algorithm to compress large term sets or obtain good compression, Algorithm 1 can not only compress much larger term sets but also find

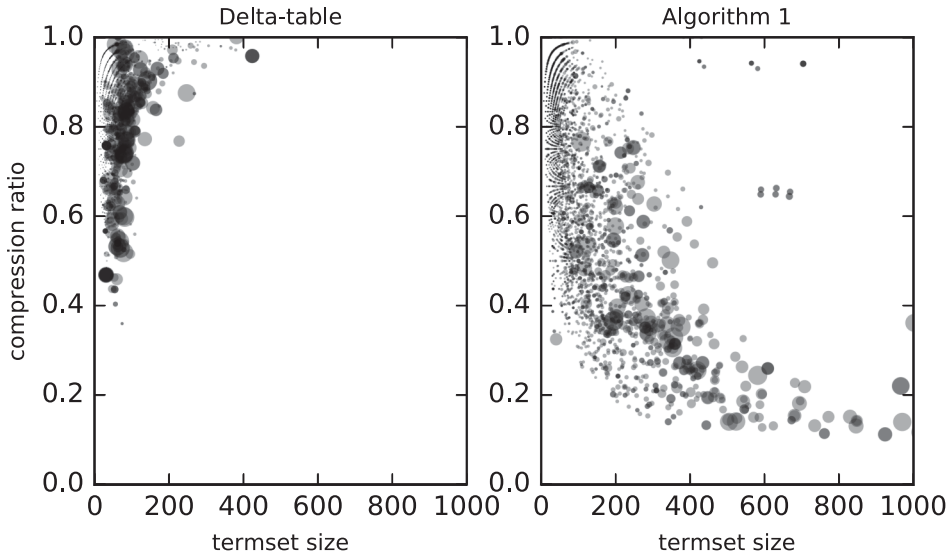

Fig. 3. Runtime performance comparison with the delta-table algorithm. As in Figure 2, each point represents an input term set, and its size is proportional to the runtime of each algorithm.

grammars with better compression. Of particular note are the small points on the bottom left: these mean that Algorithm 1 was able to find grammars with a high compression ($10\times$) for relatively large term sets (consisting of about 300 terms).

9 CONCLUSION

In this article, we have presented a practically efficient algorithm for compressing a finite tree language by a VTRATG with a fixed sequence of nonterminals. This Parameterized Language Cover Problem is the combinatorial gist of the lemma generation technique introduced in Hetzl et al. (2012) and further extended in Hetzl et al. (2014a), Hetzl et al. (2014b), and Ebner et al. (2017). The mathematical key insight is that there is a polynomial-time computable VTRATG that contains a minimal grammar as a subgrammar. This allows the polynomial reduction of the grammar compression problem to a MaxSAT problem for which highly efficient solvers are available.

The algorithm presented in this article has been integrated into our implementation (Ebner et al. 2016) of these lemma generation techniques. It considerably improves the simple algorithm introduced in Hetzl et al. (2014b) and treats a larger class of formal proofs including that considered in Hetzl et al. (2014a). The implementation also contains extensions of this algorithm for other types of grammars, such as the ones developed in Eberhard and Hetzl (2015b) describing a certain class of inductive proofs.

In principle, this algorithm could be adapted to any class of grammars that are closed under rewriting. We are also investigating how to extend the algorithm to other notions of grammar size: Eberhard and Hetzl (2017) introduce the tree complexity measure, which precisely corresponds to the number of weak quantifier inferences in a proof. The MaxSAT-reduction in Section 7 could be extended to other classes of grammars or automata as well, for example, to find tree automata with a fixed number of states and minimal number of transitions that accept a given finite set of terms. It is not yet clear how to extend the reduction to cover infinite sets of terms.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their suggestions, which have led to a considerable improvement of this article.

REFERENCES

Brian Alspach, Peter Eades, and Gordon Rose. 1983. A lower-bound for the number of productions required for a certain class of languages. *Discrete Applied Mathematics* 6, 2 (1983), 109–115.

Josep Argelich, Chu Min Li, Felip Manya, and Jordi Planes. 2008. The first and second max-SAT evaluations.*Journal on Satisfiability, Boolean Modeling and Computation* 4, 2–4 (2008), 251–278.

Walter Bucher. 1981. A note on a problem in the theory of grammatical complexity. *Theoretical Computer Science* 14 (1981), 337–344.

Walter Bucher, Hermann A. Maurer, and Karel Culik II. 1984. Context-free complexity of finite languages. *Theoretical Computer Science* 28 (1984), 277–285.

Walter Bucher, Hermann A. Maurer, Karel Culik II, and Detlef Wotschke. 1981. Concise description of finite languages. *Theoretical Computer Science* 14 (1981), 227–246.

Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. 2008. Efficient memory representation of XML document trees. *Information Systems* 33, 45 (2008), 456–474. Selected Papers from the 10th International Symposium on Database Programming Languages *(DBPL'05)*.

Samuel R. Buss. 1995. On Herbrand's theorem. In *Logic and Computational Complexity*. Lecture Notes in Computer Science, Vol. 960. Springer, 195–209.

Cezar Câmpeanu, Nicolae Santean, and Sheng Yu. 1998. Minimal cover-automata for finite languages. In *Proceedings of the 3rd International Workshop on Implementing Automata (WIA'98) (Lecture Notes in Computer Science)*, Jean-Marc Champarnaud, Denis Maurel, and Djelloul Ziadi (Eds.), Vol. 1660. Springer, 43–56.

Cezar Câmpeanu, Nicolae Santean, and Sheng Yu. 2001. Minimal cover-automata for finite languages. *Theoretical Computer Science* 267, 1–2 (2001), 3–16.

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. 2007. Tree Automata Techniques and Applications. (2007). Available on: http://www.grappa.univ-lille3.fr/tata.

Nachum Dershowitz and David A. Plaisted. 2001. Rewriting. In *Handbook of Automated Reasoning*. Vol. 1. 535–610.

Sebastian Eberhard and Stefan Hetzl. 2015a. Compressibility of finite languages by grammars. In *Proceedings of the 17th International Workshop on Descriptional Complexity of Formal Systems, (DCFS'15)*, J. Shallit and A. Okhotin (Eds.). 93–104.

Sebastian Eberhard and Stefan Hetzl. 2015b. Inductive theorem proving based on tree grammars. *Annals of Pure and Applied Logic* 166, 6 (2015), 665–700.

Sebastian Eberhard and Stefan Hetzl. 2017. On the compressibility of finite languages and formal proofs. http://www.dmg.tuwien.ac.at/hetzl/research/compressibility_proofs.pdf. To appear.

Gabriel Ebner, Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. 2017. On the generation of quantified lemmas. (2017). Submitted.

Gabriel Ebner, Stefan Hetzl, Giselle Reis, Martin Riener, Simon Wolfsteiner, and Sebastian Zivota. 2016. System description: GAPT 2.0. In *Proceedings of hte 8th International Joint Conference on Automated Reasoning (IJCAR'16)*.

Adriá Gascón, Guillem Godoy, and Florent Jacquemard. 2009. Closure of tree automata languages under innermost rewriting. *Electronic Notes in Theoretical Computer Science* 237 (2009), 23–38. *Proceedings of the 8th International Workshop on Reduction Strategies in Rewriting and Programming (WRS'08)*.

Jacques Herbrand. 1930. *Recherches sur la théorie de la démonstration*. Ph.D. Dissertation. Université de Paris.

Stefan Hetzl. 2012. Applying tree languages in proof theory. In *Language and Automata Theory and Applications (LATA'12) (Lecture Notes in Computer Science)*, Adrian-Horia Dediuand Carlos Martín-Vide (Eds.), Vol. 7183. Springer, 301–312.

Stefan Hetzl, Alexander Leitsch, Giselle Reis, Janos Tapolczai, and Daniel Weller. 2014b. Introducing quantified cuts in logic with equality. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14) (Lecture Notes in Computer Science)*, Stéphane Demri, Deepak Kapur, and Christoph Weidenbach (Eds.), Vol. 8562. Springer, 240–254.

Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. 2014a. Algorithmic introduction of quantified cuts. *Theoretical Computer Science* 549 (2014), 1–16.

Stefan Hetzl, Alexander Leitsch, and Daniel Weller. 2012. Towards algorithmic cut-introduction. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR'18) (Lecture Notes in Computer Science)*, Vol. 7180. Springer, 228–242.

Florent Jacquemard, Francis Klay, and Camille Vacher. 2009. Rigid tree automata. In *Language and Automata Theory and Applications (LATA'09) 2009 (Lecture Notes in Computer Science)*, Adrian Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide (Eds.), Vol. 5457. Springer, 446–457.

Florent Jacquemard, Francis Klay, and Camille Vacher. 2011. Rigid tree automata and applications. *Information and Computation* 209, 3 (2011), 486–512.

John C. Kieffer and En-hui Yang. 2000. Grammar based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory* 46 (2000), 737–754.

N. Jesper Larsson and Alistair Moffat. 1999. Offline dictionary-based compression. In *Proceedings of the Data Compression Conference, 1999 (DCC'99)*. 296–305.

Markus Lohrey. 2012. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology* 4, 2 (2012), 241–299.

Markus Lohrey, Sebastian Maneth, and Roy Mennicke. 2013. XML tree structure compression using RePair. *Information Systems* 38, 8 (2013), 1150–1167.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. 2014. Open-WBO: A modular MaxSAT solver. In *Theory and Applications of Satisfiability Testing (SAT'14)*. 438–445.

Craig G. Nevill-Manning and Ian H. Witten. 1997. Identifying hierarchical strcture in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research* 7 (1997), 67–82.

Gordon D. Plotkin. 1970. A note on inductive generalization. *Machine Intelligence* 5, 1 (1970), 153–163.

Gordon D. Plotkin. 1971. A further note on inductive generalization. *Machine Intelligence* 6 (1971), 101–124.

John C. Reynolds. 1970. Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence* 5, 1 (1970), 135–151.

Sherif Sakr. 2009. XML compression techniques: A survey and comparison. *Journal of Computer and System Sciences* 75, 5 (2009), 303–322.

James A. Storer and Thomas G. Szymanski. 1982. Data compression via textual substitution. *Journal of the ACM* 29, 4 (Oct. 1982), 928–951.

Geoff Sutcliffe. 2010. The TPTP world - Infrastructure for automated reasoning. In *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning (Lecture Notes in Artificial Intelligence)*, E. Clarkeand A. Voronkov (Eds.). Springer-Verlag, 1–12.

Ole Tange. 2011. GNU parallel - The command-line power tool. *;login: The USENIX Magazine* 36, 1 (Feb. 2011), 42–47.

Grigori S. Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of Reasoning: Classical Papers in Computational Logic*. Vol. 2. Springer, 466–483.

Zsolt Tuza. 1987. On the context-free production complexity of finite languages. *Discrete Applied Mathematics* 18, 3 (1987), 293–304.

Kazunori Yamagata, Tomoyuki Uchida, Takayoshi Shoudai, and Yasuaki Nakamura. 2003. An effective grammar-based compression algorithm for tree structured data. In *Inductive Logic Programming*, Tamás Horváthand Akihiro Yamamoto (Eds.). Lecture Notes in Computer Science, Vol. 2835. Springer, Berlin, 383–400.

Received April 2016; revised June 2017; accepted June 2017