Algorithmic compression of finite tree languages by rigid acyclic grammars

Sebastian Eberhard, Stefan Hetzl

Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Wiedner Hauptstraße 8-10, AT-1040 Wien, Austria

Abstract

We consider the problem of compressing a finite tree language by a totally rigid acyclic tree grammar. This class of grammars has a tight connection to proof theory and the grammar compression problem considered in this paper has applications in automated deduction.

We present a grammar compression algorithm whose crucial step consists of a polynomial translation of the finite input tree language into a \textit{MinCostSat} problem for which highly efficient solvers are available.

Keywords: 68P30 Coding and information theory (compaction, compression, models of communication, encoding schemes, etc.)

1. Introduction

Formal grammars are one of the standard tools for text compression, see e.g. [22, 20, 16, 14] for popular algorithms. 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, see for example [24], [1] and [18, 19]. Nowadays, the use of tree grammars constitutes a standard technique for compressing XML documents [21].

Such grammar-based algorithms typically compress a single string (or tree). In this paper we consider the problem of simultaneously compressing a finite set of trees by a totally rigid acyclic tree grammar (rigid tree languages have been introduced in [12, 13]). Our motivation for considering this problem is rooted in proof theory and automated deduction: as shown in [8] there is an intimate relationship between a certain class of proofs in first-order predicate logic and totally rigid acyclic tree grammars. This relationship has been exploited in [9] for devising an algorithm which analyses 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 several lemmas in [10] and to predicate logic with equality in [11]. The combinatorial gist of this algorithm is the problem of grammar-based compression considered in this paper: a grammar with a minimal number of production rules will yield a proof with a minimal number of quantifier inference rules.

The proof-theoretic application of our work entails a shift of emphasis w.r.t. traditional grammar-based compression in the following two respects: first, we do not have any freedom of choice regarding the type of grammar. Totally rigid acyclic tree grammars have to be used because they can be translated to proofs afterwards. And secondly, we are looking for a minimal grammar \( G \) s.t.
$\mathcal{L}(G) \supseteq L$ where $L$ is the finite input term set. This is the case because $L$ describes a disjunction which is required to be a tautology (a so-called Herbrand-disjunction, see [7, 2]) and if $L' \supseteq L$ then $L'$ also describes a tautology.

A first simple algorithm for compressing a tree language by a totally rigid acyclic grammar has been defined in [9]. However, this algorithm is quite limited for the following two reasons: first, it computes a grammar with only one non-terminal in addition to the start symbol and secondly the grammar fulfills $\mathcal{L}(G) = L$ which restricts the obtainable compression. In this paper we will present an algorithm that depends on a parameter $n \in \mathbb{N}$ and will be denoted as Decom$_n$. It improves the algorithm of [9] in the following crucial respects:

1. Decom$_n$ produces a grammar minimal among all with up to $n$ non-terminals.
2. Decom$_n$ produces a grammar $G$ with $\mathcal{L}(G) \supseteq L$ thus allowing stronger compression.
3. Decom$_n$ consists of a polynomial translation to an instance of MinCostSat for which highly efficient solvers are available.

Fix a $n \in \mathbb{N}$ and fix a language $L$ of terms which is to be compressed. Let us briefly sketch how the algorithm Decom$_n$ works on input $L$. The main problem to solve is that a minimal totally rigid acyclic grammar $G$ containing at most $n$ non-terminals with $L \subseteq \mathcal{L}(G)$ might contain terms as right side of rules from a set $\bar{S}$ with $|\bar{S}|$ being exponential in the size of $L$. Let us explain how this problem can be avoided: In section 3, we will define the set of terms $S_{L,n}$ via a certain normal form. We will prove that from a compression power point of view, it is sufficient to consider grammars only containing rules with right sides contained in $S_{L,n}$; as long as we are just interested in grammars containing at most $n$ non-terminals apart from their start symbol.

In section 4, we will prove that $S_{L,n}$ can be calculated in polynomial time in $L$ and $n$. The main computational challenge is to choose a minimal subset of all possible rules with right sides in $S_{L,n}$ such that the grammar containing exactly these rules produces all terms in $L$. In section 5, this selection problem will be efficiently reduced to an instance of MinCostSat for which optimised algorithms are known. In section 6, we propose such algorithms, and discuss which of them are most suitable to solve the instance of MinCostSat occurring in our setting.

The sketched strategy allows the definition of an algorithm Decom$_n$ reducing the problem of finding a minimal totally rigid acyclic grammar with $L \subseteq \mathcal{L}(G)$, containing at most $n$ non-terminals apart from the start symbol, in polynomial time to an instance of MinCostSat. This definition will be given in section 7.

2. Basic notations and definitions

In the following, we define the class of totally rigid acyclic tree grammars via the notion of regular tree grammars.

**Definition 1 (Regular tree grammar).** A regular tree grammar is a tuple $\langle \alpha_0, N, \Sigma, P \rangle$ composed of an axiom (start symbol) $\alpha_0$, a set $N$ of non-terminal symbols with arity 0 and $\alpha_0 \in N$, a term signature $\Sigma$ with $\Sigma \cap N = \emptyset$ and a set $P$ of productions of the form $\beta \rightarrow t$ where $t$ is a term build from $\Sigma \cup N \setminus \{\tau\}$ and $\beta \in N$.

Derivations in regular tree grammars $G$ are defined as expected (see e.g. [3]), giving rise to the language $\mathcal{L}(G)$ produced by $G$. 
Notation 2. For a term \( t \) its positions \( p \) are given as words composed of natural numbers as usual (see e.g. [3]). As usually, for a term \( t \) and a position \( p \) of \( t \) we write \( t|_p \) for the subterm of \( t \) with head at position \( p \).

Definition 3. A rigid tree grammar is a tuple \( \langle \alpha_0, N, R, \Sigma, P \rangle \) where \( \langle \alpha_0, N, \Sigma, P \rangle \) is a regular tree grammar and \( R \subseteq N \). We call \( R \) the set of rigid non-terminal symbols.

The notion of rigid tree automata and grammars has been introduced by Jacquemard, Klay, and Vacher in [12, 13].

The language produced by a rigid tree grammar is restricted to the set of terms which can be produced by derivations satisfying the rigidity condition given as follows.

Definition 4 (Rigidity condition for derivations). Let \( \delta := \alpha_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_m \rightarrow t \) be a derivation of a term \( t \) in the grammar \( \langle \alpha_0, N, \Sigma, P \rangle \). Assume \( R \subseteq N \). Then, the derivation \( \delta \) satisfies the rigidity condition relative to \( R \) exactly if the following condition holds: For all positions \( p_0 \) and \( p_1 \) and all non-terminals \( \beta \in R \) occurring at positions \( p_0 \) and \( p_1 \) in a term occurring in \( \delta \), we have \( t|_{p_0} = t|_{p_1} \). If this is the case, we call \( t|_{p_0} \) the \( \beta \)-rest of the rigid derivation \( \delta \).

Definition 5 (Language produced by rigid grammar). Let \( G := \langle \alpha_0, N, R, \Sigma, P \rangle \) be a rigid grammar. Then, \( \mathcal{L}(G) \) is given as the set of terms producible by derivations in \( \langle \alpha_0, N, \Sigma, P \rangle \) satisfying the rigidity condition relative to \( R \).

Definition 6. A rigid grammar \( \langle \alpha_0, N, R, \Sigma, P \rangle \) is called totally rigid exactly if \( N = R \).

Definition 7. Let \( G \) be a regular or rigid tree grammar with non-terminals \( N \) and productions \( P \). Define a binary relation \( <^1_G \) on \( N \) such that \( \beta <^1_G \gamma \) exactly if there is a term \( t \) such that \( \beta \rightarrow t \in P \) and \( \gamma \) occurs in \( t \), and write \( <_G \) for the transitive closure of \( <^1_G \). \( G \) is called acyclic if \( <_G \) is.

Notation 8. Totally rigid acyclic tree grammars are abbreviated as trat grammars in the following.

The following lemma, given as lemma 11 in Hetzl’s [8] nicely characterises trat grammars.

Lemma 9. If \( G \) is a trat grammar, then up to renaming of non-terminals \( G = \langle \alpha_0, \{\alpha_0, \cdots, \alpha_n\}, \{\alpha_0, \cdots, \alpha_n\}, \Sigma, P \rangle \) with \( \mathcal{L}(G) = \{ \alpha_0|_{[q_0]} \cdots |_{[q_n]} | \alpha_i \rightarrow q_i \in P \} \).

Notation 10. For all trat grammars \( G \), we assume that its non-terminals are named as in the previous lemma, i.e. they are given as \( \alpha_0, \alpha_1, \alpha_2, \cdots \) with \( <_G \) given such that \( \alpha_i <_G \alpha_j \) implies \( i < j \).

A trat grammar containing exactly non-terminals \( \alpha_0, \alpha_1, \cdots, \alpha_n \) and an axiom \( \alpha_0 \) is called a trat-n grammar. In such contexts, \( n > 0 \) is always assumed.

Notation 11. Let \( G \) be a grammar. The number of productions of \( G \) is denoted as size of \( G \), or symbolically as \( |G| \). When we talk of minimal grammars in the following, we refer to grammars of minimal size.

For a term \( t \), we write \( |t| \) for the number of nodes of the representation of \( t \) as a tree.
• Terms not containing non-terminals are called closed. A finite, non-empty set of closed terms is called a language\(^2\).

**Notation 12.** For a term \(k\), non-terminals \(\alpha_1, \ldots, \alpha_n\), and closed terms \(s_1, \ldots, s_n\) and \(t\) we write

\[
k \circ_{\alpha_1, \ldots, \alpha_n} \begin{pmatrix}
s_1 \\
\vdots \\
s_n
\end{pmatrix} = t
\]

for \(k \mid \alpha_1 \backslash s_1, \ldots, \alpha_n \backslash s_n \rangle = t\). Note that since \(s_1, \ldots, s_n\) are closed the above displayed substitutions can be applied in any order. Note that if \(k\) contains the non-terminals \(\alpha_1, \ldots, \alpha_n\) the terms \(s_1, \ldots, s_n\) are uniquely determined by the pair \(k, t\).

For a term \(k\) containing the non-terminals \(\alpha_1, \ldots, \alpha_n\) and closed terms \(s_{1,1}, \ldots, s_{n,1}, \ldots, s_{1,m}, \ldots, s_{n,m}\) we write

\[
k \circ_{\alpha_1, \ldots, \alpha_n} \left\{ \begin{pmatrix}
 s_{1,1} \\
 \vdots \\
 s_{n,1}
\end{pmatrix}, \ldots, \begin{pmatrix}
 s_{1,m} \\
 \vdots \\
 s_{n,m}
\end{pmatrix} \right\}
\]

for

\[
\left\{ k_i \circ_{\alpha_1, \ldots, \alpha_n} \begin{pmatrix}
 s_{1,i} \\
 \vdots \\
 s_{n,i}
\end{pmatrix} : 1 \leq i \leq m \right\}.
\]

**Definition 13 (Key of a language).** A term \(k\) containing non-terminals \(\alpha_1, \ldots, \alpha_n\) is key of a language \(L\) if there is a set \(R\) (of \(n\)-tuples) such that \(k \circ_{\alpha_1, \ldots, \alpha_n} R = L\).

**Definition 14 (Key of a grammar).** A term \(k\) is key of a grammar \(G\) exactly if \(G\) contains a production with right side \(k\).

**Definition 15 (Induced substitution).** Let \(k\) be a key for a language \(L\) containing the non-terminals \(\alpha_1, \ldots, \alpha_n\). Assume

\[
k \circ_{\alpha_1, \ldots, \alpha_n} \begin{pmatrix}
 s_1 \\
 \vdots \\
 s_n
\end{pmatrix} = t
\]

for a term \(t \in L\). Then, we write \(\sigma_{L,k,t}\) for the following composition of substitutions:

\[
[\alpha_1 \backslash s_1, \ldots, \alpha_n \backslash s_n]
\]

For \(n = 0\), \(\sigma_{L,k,t}\) is defined as the identity substitution.

In the following, if the language \(L\) and the term \(k\) are clear from the context, we write \(\sigma_t\) instead of \(\sigma_{L,k,t}\).

---

\(^2\)In the literature, the notion of language usually also refers to infinite sets of terms. However, in this paper we will only deal with finite sets of terms as trat grammars produce finite languages. Given a finite language \(L\) the existence of a trat grammar \(G\) with \(L(G) = L\) is trivial, the interesting question is that of finding a *minimal* \(G\).
Definition 16. Let \( k \) be a key for the language \( L \) containing non-terminals \( \alpha_1, \ldots, \alpha_n \). Let \( q_0, q_1 \) be terms containing at most non-terminals \( \alpha_1, \ldots, \alpha_n \). \( L \) fulfills the equation \( q_0 = q_1 \) relative to \( k \) exactly if for all \( t \in L \)
\[
q_0[\sigma_t] = q_1[\sigma_t].
\]
Assume that for a term \( t \in L \), we have
\[
q_0[\sigma_t] \neq q_1[\sigma_t].
\]
Then, \( t \) falsifies the equation \( q_0 = q_1 \) relative to \( k \).

Notation 17. In the following, if a key \( k \) of a language \( L \) is clear from the context, for equations \( s = t \), we just write that \( L \) satisfies \( s = t \) instead of \( L \) satisfies \( s = t \) relative to \( k \). Analogously for terms falsifying equations.

Example 18. Let the language \( L \) be given as \( \{f(g(c), g(c)), f(g(d), g(d))\} \), and the key \( k \) as \( f(\alpha_1, g(\alpha_2)) \). We have
\[
\begin{align*}
\sigma_{f(g(c), g(c))} &= [\alpha_1 \backslash g(c), \alpha_2 \backslash c] \text{ and } \\
\sigma_{f(g(d), g(d))} &= [\alpha_1 \backslash g(d), \alpha_2 \backslash d].
\end{align*}
\]
Accordingly, \( L \) fulfills the equation \( \alpha_1 = g(\alpha_2) \).

3. A sufficient set of keys

In the introduction, we mentioned a set \( S_{L,n} \) such that from a compression power point of view, it is sufficient to consider grammars only containing keys in \( S_{L,n} \); as long as we are just interested in compressing \( L \) using tract-\( n \) grammars. In this section, we define \( S_{L,n} \) via the notion of a new normal form which will be introduced below. Then, we prove that this set has the intended properties.

Definition 19 (Normal form of decomposition key). Assume that \( k \) is a key for the language \( L \). Assume that \( k \) contains at least the non-terminals \( \alpha_1, \ldots, \alpha_\ell \) for \( \ell \in \mathbb{N} \). Then \( k \) is in normal form relative to \( L \) and \( \alpha_1, \ldots, \alpha_\ell \) exactly if the following condition holds:

For any equation of the form \( \alpha_i = q \) for \( 1 \leq i \leq \ell \) for \( q \) being a subterm of \( k \) or \( q \) being a closed term if \( \alpha_i = q \) is satisfied by \( L \) then \( q \) equals \( \alpha_i \).

We say that \( k \) is in normal form relative to \( L \) exactly if \( k \) is in normal form relative to \( L \) and all non-terminals occurring in \( k \).

Notation 20. We stipulate that no term is in normal form relative to the empty set and any of its non-terminals.

Remark 21. • Intuitively, keys in normal form do not contain idle non-terminals: If \( L \) satisfies a non-trivial equation of the form \( \alpha_i = q \) for a key \( k \) given as above, it is easy to see that the key \( k[\alpha_i \backslash q] \) decomposes \( L \) as well. \( k[\alpha_i \backslash q] \) contains one non-terminal less than \( k \).

• Note that keys without occurrences of non-terminals are in normal form relative to any language.
Example 22. Let \( L \) and \( k \) be given as in example 18. \( k \) is not in normal form relative to \( L \) because \( L \) satisfies the equation \( \alpha_1 = g(\alpha_2) \) for \( g(\alpha_2) \) being a subterm of \( k \). E.g. the keys \( \alpha_1, f(\alpha_1, \alpha_1) \) and \( f(g(\alpha_1), g(\alpha_1)) \) are in normal relative to \( L \).

Remark 23. The key \( f(\alpha_1, \alpha_1) \) is in normal form relative to \( L \) even though it introduces \( \alpha_1 \) at a position where all heads of the terms of \( L \) are equal. In the terminology of [9, 23] a left-shift of the decomposition of \( L \) with left side \( f(\alpha_1, \alpha_1) \) is possible. This is contrary to the definition of normal form in [23] used in [11]. The reason for our more relaxed definition is to make lemmas 29 and 33 hold which would fail for the restricted definition of normal form in [23, 11].

In the following, we present easy lemmas about the normal form.

Lemma 24. Assume that \( k \) is in normal form relative to \( \{t\} \). Then \( k = t \).

Lemma 25. Assume that \( k \) is in normal form relative to \( L \) and the non-terminals \( \alpha_1, \ldots, \alpha_\ell \). Assume \( q_0 \neq q_1 \), where \( q_i \) for \( i = 0,1 \) is either a subterm of \( k \) containing at most non-terminals \( \alpha_1, \ldots, \alpha_\ell \) or a closed term. Then there is a term \( t \in L \) falsifying \( q_0 = q_1 \).

Proof. By induction on the added size of the terms \( q_0,q_1 \). Let \( q_0 \) be a non-terminal \( \alpha_i \) with \( 1 \leq i \leq \ell \). Then since \( q_1 \) is a subterm of \( k \) or a closed term, and \( k \) is in normal form relative to \( L \) and \( \alpha_1, \ldots, \alpha_\ell \), there must be a term \( t \in L \) falsifying \( q_0 = q_1 \). The induction step is obvious. \( \square \)

We can define the set \( S_{L,n} \) mentioned above.

Notation 26. For a language \( L \), let \( st(L) \) denote the set of subterms of terms in \( L \).

Definition 27 (\( S_{L,n} \)). Let \( L \) be a language, and \( n \) a natural number. Then, \( S_{L,n} \) is defined as the set of terms containing at most non-terminals \( \alpha_1, \ldots, \alpha_n \) being in normal form relative to some \( L' \subseteq st(L) \).

We introduce the notion of a sufficient set of keys which will allow the formalisation of the fact that the set \( S_{L,n} \) contains all keys necessary for an optimal compression of \( L \) by trat-\( n \) grammars.

Definition 28 (Sufficient set of keys). Let \( L \) be a language, and \( n \) a natural number. A set of terms \( S \) is called a sufficient set of keys of \( n \) non-terminals for \( L \) exactly if the following property holds:

Let \( m \in \mathbb{N} \) be the size of a minimal trat-\( n \) grammar \( G \) such that \( L \subseteq \mathcal{L}(G) \). Then there is a trat-\( n \) grammar \( G' \) with \( L \subseteq \mathcal{L}(G') \) and size \( m \) containing only keys in \( S \).

Theorem 29. Let \( L \) be a language. Let \( n \) be a natural number. Then, \( S_{L,n} \) is a sufficient set of keys of \( n \) non-terminals for \( L \).

We need an auxiliary lemma for the proof of the theorem.

Notation 30. We call an equation trivial if it is of the form \( s = s \) for a term \( s \).

Notation 31. Let \( k \) be a key of \( L \). We define \( E_L(k) \) as the set of non-trivial equations of the form \( \alpha_i = q \) satisfied by \( L \) with \( \alpha_i \) being a non-terminal occurring in \( k \), and \( q \) a subterm of \( k \) which is not closed. If \( L \) is clear from the context, we just write \( E(k) \).
Lemma 32. Assume $k \circ_{\alpha_1, \ldots, \alpha_n} R = L$ for a term $k$, a set of $n$-tuples of terms $R$, and a language $L$. Then, there is a key $k'$ in normal form relative to $L$ such that

$$k' \circ_{\alpha_1, \ldots, \alpha_n} R = L.$$  

($k'$ may contain less non-terminals than $k$, therefore some substitution terms in $R$ might be idle.)

Proof. We prove by induction on $|E(k)|$ that there is a term $\tilde{k}$ with $\tilde{k} \circ_{\alpha_1, \ldots, \alpha_n} R = L$ such that $E(\tilde{k}) = 0$. The searched term $k'$ in normal form is then easily obtained from $\tilde{k}$ by replacing non-terminals $\alpha_i$ by $q$ if equations of the form $\alpha_i = q$ are satisfied by $L$ for $q$ a closed term.

Assume that $\alpha_i = q$ is an element of $E(k)$. Let us define a key $\bar{k}$ with $|E(\bar{k})| < |E(k)|$. $\bar{k}$ is obtained from $k$ by replacing each subterm $q$ of $k$ by $\alpha_i$. It follows easily that $E(\bar{k})$ is a subset of $E'$ obtained from $E(k)$ by replacing all occurring subterms $q$ by $\alpha_i$, and by dropping the equation $\alpha_i = q$. Clearly, for the new key $\hat{k}$, we still have $\hat{k} \circ_{\alpha_1, \ldots, \alpha_n} R = L$. □

We are ready for the proof of Theorem 29

Proof.[of Theorem 29] Assume $L \subseteq L(G)$ for the trat-$n$ grammar $G$. We prove that there exists a trat-$n$ grammar $G'$ with $|G'| \leq |G|$ containing only keys in $S_{L,n}$ which immediately implies the theorem.

The proof uses induction over the number of rules of $G$ of the form $\alpha_i \rightarrow k$ for $0 \leq i \leq n$ with $k$ not being in normal form relative to any subset of $st(L)$. We fix such a $k$. Assume that $L'$ is the largest subset of $st(L)$ such that $k$ is a key of $L'$. We can assume that $L'$ is not the empty set since otherwise the rule $\alpha_i \rightarrow k$ can be dropped entirely. We have $k \circ_{\alpha_1, \ldots, \alpha_n} R = L'$ for a certain $R$. We apply the previous lemma to obtain a key $k'$ in normal form relative to $L'$ with $k' \circ_{\alpha_1, \ldots, \alpha_n} R = L'$. Let $\hat{G}$ be defined as $G$, but with the production $\alpha_i \rightarrow k'$ instead of the production $\alpha_i \rightarrow k$. We have to show that all terms in $L$ can still be derived in $\hat{G}$. Assume that in a derivation of a $t \in L$ in $G$, the rule $\alpha_i \rightarrow k$ was used. But then, we obtain a derivation of $t$ in $\hat{G}$ by using the rule $\alpha_i \rightarrow k'$ instead and by using the same derivations starting from the non-terminals in $k'$ as the ones used previously by $G$ in the derivation of $t$. The induction hypothesis can be applied to $\hat{G}$ yielding the desired result. □

4. Efficient computation of a sufficient set of keys

It is not obvious that the set $S_{L,n}$, introduced in the last section, can be produced efficiently from $L$ since terms in normal form relative to each subset of $st(L)$ are considered. Nevertheless, in this section we prove that it is already sufficient to consider terms in normal form relative to subsets of $st(L)$ of a size bounded by $n + 1$. This will be proved in the first subsection. In the second subsection, we present an algorithm efficiently constructing from an input language $L'$ all keys in normal form relative to $L'$. Finally, in the third subsection, the results of the first and second subsection will be combined to introduce an algorithm $\text{SuffKeys}_n$ efficiently producing $S_{L,n}$ from input $L$.

4.1. Normal forms and small subsets

This section is devoted to the proof of the following theorem.

Theorem 33. Let $k$, containing exactly the non-terminals $\alpha_1, \ldots, \alpha_n$, be in normal form relative to $L$ for $n \in \mathbb{N}$. Then there exists a set $L' \subseteq L$ with $|L'| \leq n + 1$ such that $k$ is in normal form relative to $L'$. 

7
We start with some auxiliary definitions.

**Notation 34.** For terms \( s, t, s \subseteq t \) is defined to hold exactly if \( s \) is a subterm of \( t \).

**Definition 35 (Preorder on non-terminals).** Let \( k \), containing exactly the non-terminals \( \alpha_1, \ldots, \alpha_n \), be in normal form relative to \( L \). For \( t \in L \), we assume \( \sigma_t = [\alpha_1 \setminus s_1, \ldots, \alpha_n \setminus s_n] \). Then, \( t \) induces the following preorder on \( \alpha_1, \ldots, \alpha_n \).

\[
\alpha_i \leq \alpha_j :\Leftrightarrow s_i \subseteq s_j
\]

**Proof.**[of Theorem 33] We pick an arbitrary \( t_0 \in L \). Let \( \leq \) denote the preorder on the non-terminals \( \alpha_1, \ldots, \alpha_n \) of \( k \) induced by \( t_0 \). Fix a sequence \( T_0, T_1, \ldots, T_m \) of subsets of \( \{\alpha_1, \ldots, \alpha_n\} \) such that the following properties hold for all \( 0 \leq i, j \leq n \) and all \( 0 \leq \ell, q \leq m \).

- \( T_0 = \emptyset \)
- \( T_m = \{\alpha_1, \ldots, \alpha_n\} \)
- \( q < \ell \Rightarrow T_q \subseteq T_\ell \)
- \( \alpha_i \in T_\ell \land \alpha_j \leq \alpha_i \land \neg(\alpha_i \leq \alpha_j) \Rightarrow (\exists q < \ell)(\alpha_j \in T_q) \)
- \( \alpha_i, \alpha_j \in T_{q+1} \setminus T_q \Rightarrow \alpha_i \leq \alpha_j \land \alpha_j \leq \alpha_i \)

We will construct a sequence \( S_0, S_1, \ldots, S_m \) such that the following properties hold for all \( 0 \leq i, j \leq n \) and all \( 0 \leq \ell, q \leq m \).

- \( S_0 = \{t_0\} \)
- \( q < \ell \Rightarrow S_q \subseteq S_\ell \)
- \( |S_i| \leq |T_i| + 1 \)
- \( k \) is in normal form relative to \( S_i \) and \( T_i \).

Clearly, if the sequence \( S_0, \ldots, S_m \) can be constructed, its third and its fourth property for \( S_m \) and \( T_m \) together imply the required lemma. We will construct the sequence \( S_0, S_1, \ldots, S_m \) recursively and prove that the required properties hold for each initial segment of \( S_0, S_1, \ldots, S_m \) already constructed. The first and second property will always be satisfied trivially, and therefore are not mentioned in the following.

All required properties clearly hold for \( S_0 \) and \( T_0 \). Let us define \( S_1 \). Assume for simplicity that \( T_1 \) contains exactly the non-terminals \( \alpha_1, \ldots, \alpha_n \). All of these non-terminals are replaced by the same closed term to produce \( t_0 \). We build the set \( S_1 \) by repeatedly adding terms from \( L \) to \( S_0 = \{t_0\} \).

In the following, we always deal with equations \( q_0 = q_1 \) with both terms being subterms of \( k \) or closed terms, and assume this property without explicitly mentioning it. We also always assume that the displayed equations are non-trivial.

The only equations we have to refute to obtain a term in normal form relative to \( T_1 \) are of the form \( \alpha_i = \alpha_j \) or \( \alpha_i = q \) for a closed term \( q \) since other equations are refuted by \( t_0 \). Because of \( k \) being in normal form relative to \( L \) there exists a term \( t_1 \in L \) such that \( t_1 \) refutes \( \alpha_1 = \alpha_2 \). The term \( t_1 \) is included into the set \( S_1 \). We partition the set \( \{\alpha_1, \ldots, \alpha_n\} \) into subsets \( R_1, \ldots, R_q \) such that \( \alpha_i, \alpha_j \) are in the same set exactly if \( \{t_1\} \) satisfies \( \alpha_i = \alpha_j \) for all \( 1 \leq i, j \leq n \). Let \( R_i \) be the first
set in the list $R_1, \ldots, R_\eta$ of size larger than 1. We include a term $t_2$ into $S_1$ falsifying $\alpha_{t_1} = \alpha_{t_2}$ for $\alpha_{t_1}, \alpha_{t_2}$ being two arbitrary members of $R_1$, and receive a new partition $R'_1, \ldots, R'_\eta$ such that $\alpha_i, \alpha_j$ are in the same set exactly if $\alpha_i = \alpha_j$ is satisfied by $\{t_1, t_2\}$ for all $1 \leq i, j \leq n$.

This strategy is continued analogously until the constructed partition only contains sets of size 1. When this is the case the preliminary set $S_1$ has at most size $|T_1| = \kappa$ because of combinatorial reasons. There is at most one $\alpha_i$ and at most one closed term $q$ such that all terms in the preliminary $S_1$ fulfil the equation $\alpha_i = q$. We add to $S_1$ a term $t_\kappa \in L$ falsifying this equation. This yields the final $S_1$. It can be checked easily, that $k$ is in normal form relative to $S_1$ and $T_1$.

For the induction step from $\nu$ to $\nu + 1$, first assume that $T_{\nu+1} \setminus T_\nu$ contains only one element $\alpha_\ell$. Then, the only equations that may hold for all elements in $S_\nu$ and especially for $t_0$ are of the form $\alpha_\ell = q$ for $q$ only containing non-terminals smaller than $\alpha_\ell$ relative to the preorder induced by $t_0$. Note that $T_\nu$ contains all non-terminals below $\alpha_\ell$ relative to this preorder. $q$ is unique because of lemma 25 since the key $k$ is in normal form relative to $S_\nu$ and $T_\nu$. To obtain $S_{\nu+1}$, we add a term falsifying the equation $\alpha_\ell = q$.

Next, assume that $T_{\nu+1} \setminus T_\nu$ consists of more than one element. We obtain a partition $R_1, \ldots, R_\eta$ such that $\alpha_i, \alpha_j$ are in the same set exactly if $\alpha_i = \alpha_j$ is satisfied by $S_\nu$ for $1 \leq i, j \leq n$. Because of lemma 25, and because $k$ is in normal form relative to $S_\nu$ and $T_\nu$ we deduce the following property:

For each $1 \leq \ell \leq \eta$, there is at most one $q$ such that $\alpha_i = q$ is satisfied by $S_\nu$ for all $\alpha_i \in R_\ell$. The non-terminals occurring in $q$ are contained in $T_\nu$.

We call the above mentioned equations the characteristic equations of $R_\ell$ for all $1 \leq \ell \leq \eta$. Then, as for the construction of $S_1$, we successively add terms to $S_\nu$ falsifying equations of the form $\alpha_i = \alpha_j$ between non-terminals of the same set $R_\ell$.

We reach a partition $R'_1, \ldots, R'_\eta$ with all sets having size one after adding at most $|T_{\nu+1} \setminus T_\nu| - \eta$ terms to $S_\nu$ for combinatorial reasons. This yields a preliminary $S_{\nu+1}$. Let us check whether the characteristic equations of $R_\ell$ for $1 \leq \ell \leq \eta$ still hold for the preliminary $S_{\nu+1}$. It easily follows, that for each of the original $R_\ell$ at most one characteristic equation is still satisfied in the preliminary $S_{\nu+1}$. Therefore, after adding $\eta$ suitable terms to the preliminary $S_{\nu+1}$, all of them are refuted. This yields the searched set $S_{\nu+1}$. Clearly, we added at most one term for each non-terminal in $(T_{\nu+1} \setminus T_\nu)$ which makes the third required property true for $S_{\nu+1}$. It is easy to see that $k$ is in normal form relative to $S_{\nu+1}$ and $T_{\nu+1}$ making the fourth required property true. By executing the induction until $\nu + 1 = m$, we obtain the required result.

Let us prove that the bound $n + 1$ in Theorem 33 is tight.

**Lemma 36.** For each $n \in \mathbb{N}$ with $n > 0$ there is term $k_n$ containing exactly $n$ non-terminals and a language $M_n$ such that all of the following properties hold.

- $k_n$ is a key in normal form relative to $M_n$.
- $k_n$ is not in normal form relative to any subset of $M_n$ having less than $n + 1$ elements.

**Proof.** Let $f$ be a function of an arity determined by the context. For all $n \in \mathbb{N}$, $M_n$ is given as

$$\{f(0,0,\ldots,0), f(1,0,\ldots,0), \ldots, f(1,1,\ldots,1)\}$$

9
Clearly, for each \( n > 0 \) the set \( M_n \) contains exactly \( n + 1 \) elements. We define \( k_n \) as follows for non-terminals \( \alpha_1, \alpha_2, \ldots, \alpha_n \):

\[
k_n := f(\alpha_1, \alpha_2, \ldots, \alpha_n)
\]

It is easy to see that the key \( k_n \) is in normal form relative to \( M_n \), but that it is not in normal form relative to any strict subset of \( M_n \). \( \square \)

4.2. Efficient computation of keys in normal form

Theorem 33 allows the definition of an efficient algorithm depending on a parameter \( n \in \mathbb{N} \) which from an input language \( L \) produces all keys in normal form relative to \( L \) containing at most \( n \) non-terminals \( \alpha_1, \ldots, \alpha_n \) for any \( n \in \mathbb{N} \). For \( n \in \mathbb{N} \), we will denote this algorithm with its parameter fixed as \( n \) as \( \text{NormForm}_n \). \( \text{NormForm}_n \) will be polynomial time computable in its input \( L \) for any \( n \in \mathbb{N} \). In the following, we fix an \( n \in \mathbb{N} \).

\( \text{NormForm}_n \) depends on the generalised delta vector operation \( gdv \) which was used by Tapolczai in [23] producing the maximal common part of an arbitrary input language \( L \). Thus, \( gdv(L) \) is a key of \( L \) for any language \( L \). Let us sketch \( gdv \) in the following.

Let us first sketch the behaviour of \( gdv \): We recursively descends through an input language \( L \) of terms in parallel to search for a maximal common term structure. Whenever we encounter a subset \( L' \) of \( st(L) \) not having a common head, a non-terminal is introduced. Its replacement by the terms in \( L' \) will allow to obtain the corresponding original terms in \( L \). This procedure yields a key \( k \) of \( L \). Finally, all pairs of non-terminals of \( k \) which can be merged will be merged which yields \( gdv(L) \).

Let us give a precise definition of \( gdv \). We need an auxiliary function \( gdv' \) which finds a maximal common part without executing any merging.

**Algorithm 1. \( gdv' \)**

Input: Language \( L := \{t_1, \ldots, t_n\} \)

Output: \( gdv'(L) \)

Execution:

- Compare the heads of the elements in \( L \):
  - If all terms \( t_i \) in \( L \) are of the form \( f(s_{i,1}, \ldots, s_{i,m}) \) for the same function symbol \( f \) and terms \( s_{k,\ell} \) for \( 1 \leq k \leq n \) and \( 1 \leq \ell \leq m \), we have
    \[
    gdv'(L) = f(gdv'({s_{1,1}, \ldots, s_{n,1}}), gdv'({s_{1,2}, \ldots, s_{n,2}}), \ldots, gdv'({s_{1,m}, \ldots, s_{n,m}})).
    \]
  - Else, we have \( gdv'(L) = \alpha_{\text{fresh}} \), where \( \alpha_{\text{fresh}} \) denotes a non-terminal which until now never occurred in any computation of \( gdv' \).

end

We are ready for the definition of \( gdv^3 \).

---

\(^3\) The algorithm \( gdv \) presented in [23] executes merging already during the construction of the key \( k \), and not only after having constructed it. Nevertheless, this does not make a difference for our arguments.
Algorithm 2. \textit{gdv}

Input: Language \( L := \{ t_1, \cdots, t_n \} \)
Output: \( gdv(L) \)
Execution:

\begin{itemize}
  \item Set \( k := gdv'(L) \)
  \item Successively, for all pairs \( \alpha_i, \alpha_j \) with \( i < j \) of non-terminals of \( gdv'(L) \):
    \begin{itemize}
    \item If \( \alpha_i = \alpha_j \) is satisfied by \( L \) store \( k[\alpha_j/\alpha_i] \) as new \( k \).
    \item Else, continue with the next pair.
    \end{itemize}
\end{itemize}

end

Example 37. We apply \( gdv \) to the term set \( L := \{ f(g(a), a), f(g(b), c), f(g(c), c) \} \).

\( gdv \) proceeds as follows.

\begin{itemize}
  \item The heads of the terms are compared. Since they are equal, the output of \( gdv' \) is of the form \( f(\cdot, \cdot) \).
  \item The subterms \( g(a), g(b), g(c) \) are compared. Since they have the same head, the output of \( gdv' \) is of the form \( f(g(\cdot), \cdot) \).
  \item The subterms \( a, c, c \) are compared. Since their heads are not equal, the output of \( gdv' \) is of the form \( f(g(\cdot), \alpha_1) \). After executing a final comparison step, we receive \( k = f(g(\alpha_2), \alpha_1) \).
  \item No merging of non-terminals of \( k \) will be applied since the non-terminals \( \alpha_1 \) and \( \alpha_2 \) are replaced differently to produce the term \( f(g(b), c) \) which refutes the equation \( \alpha_1 = \alpha_2 \). Therefore, \( gdv(L) = k \).
\end{itemize}

\( gdv(L) \) is a key of \( L \), we have \( gdv(L) \circ_{\alpha_1, \alpha_2} \left\{ \left( \begin{array}{c} a \\ a \end{array} \right), \left( \begin{array}{c} c \\ b \end{array} \right), \left( \begin{array}{c} c \\ c \end{array} \right) \right\} = L \).

Remark 38. In the following, for non-terminals occurring in generalised delta vectors, we often just write \( \alpha_i \) for \( a_i \in \mathbb{N} \) instead of \( \alpha_{\text{num}(L)} \) for a language \( L \).

Lemma 39. Let \( L \) be a language. Then \( gdv(L) \) is a key in normal form of \( L \).

Proof. Equations of the form \( \alpha_i = q \) for \( \alpha_i, q \in \text{st}(gdv(L)) \) can only be satisfied in \( L \) for \( q \) being a non-terminal. This is because otherwise, the heads compared at the stage of the execution of \( gdv(L) \) where \( \alpha_i \) was introduced were equal which is impossible. Equations of the form \( \alpha_i = \alpha_j \) do not hold because merging of non-terminals is executed whenever possible. \( \square \)

The following technical definition allows to present the algorithm \textit{NormForm}_n for the construction of keys in normal form in an understandable way.
**Definition 40.** Let $k$ be a term. Let $P$ be the set of positions $p$ of $k$ such that $k|_p$ contains a non-terminal. Then, the characteristic partition of $k$ is the partition of $P$ induced by the following equivalence relation $R$ on $P$:

$$R(p_0, p_1) :\iff k|_{p_0} = k|_{p_1}$$

Now, let us give a precise description of $\text{NormForm}_n$. Intuitively, for an input language $L$, $\text{NormForm}_n$ shifts the non-terminals of $gdv(L)$ upwards. In the terminology of [9, 23] it executes certain right-shifts on the decomposition of $L$ with left side $gdv(L)$.

**Algorithm 3.** $\text{NormForm}_n$

Input: Language $L$.
Output: The set of all keys containing at most non-terminals $\alpha_1, \ldots, \alpha_n$ which are in normal form relative to the set $L$.

Execution:
We let $k$ denote $gdv(L)$ with non-terminals $\alpha_i$ renamed by syntactically different non-terminals $\beta_i$ for all $i \in \mathbb{N}$. Let $\mathcal{P}$ denote the characteristic partition of $k$.

- Successively, for all ordered lists $P_1, P_2, \ldots, P_n$ with $P_i \in \mathcal{P} \cup \{\emptyset\}$ for all $1 \leq i \leq n$, and $P_i \neq P_j \lor P_i = \emptyset$ for all $1 \leq i \neq j \leq n$:
  - Initialise $\text{new-key}$ as $k$.
  - For $1 \leq i \leq n$
    - Successively, for all positions $p$ in $P_i$: If $p$ is a position of $\text{new-key}$, replace $\text{new-key}|_p$ by $\alpha_i$ in $\text{new-key}$. Store the modified $\text{new-key}$ as new $\text{new-key}$.
    - Else, no replacement is carried out.
    - Add the term stored as $\text{new-key}$ to the output set (initialised as the empty set) if $\text{new-key}$ does not contain a non-terminal of the form $\beta_i$ for $i \in \mathbb{N}$.

**Example 41.** Let us evaluate $\text{NormForm}_2$ on input

$$L := \{ f(g(a), g(a), g(a)), f(g(b), g(b), g(c)) \}.$$ 

We have

$$k = f(g(\beta_1), g(\beta_1), g(\beta_2)).$$

The characteristic partition of $gdv(L)$ is given as follows:

$$\{\epsilon\}, \{1, 2\}, \{3\}, \{11, 21\}, \{31\}$$

Let us calculate the output set. For all displayed keys, also the key with $\alpha_1$ and $\alpha_2$ switched is given as output of $\text{NormForm}_2$. If $\{\epsilon\}$ is in the chosen list, the trivial key $\alpha_1$ results. If $\{1, 2\}, \{3\}$ are in the chosen list, the key $f(\alpha_1, \alpha_1, \alpha_2)$ results. If $\{1, 2\}, \{31\}$ are in the chosen list, we obtain $f(\alpha_1, \alpha_1, g(\alpha_2))$. For $\{3\}, \{11, 21\}$, we obtain $f(g(\alpha_1), g(\alpha_1), g(\alpha_2))$. If $\{11, 21\}, \{31\}$ are in the chosen list, we obtain $f(g(\alpha_1), g(\alpha_1), g(\alpha_2))$. For other chosen lists output keys will not be produced since the accordingly substituted term $k$ will still contain $\beta_1$ or $\beta_2$.

It can be checked easily that the output set exactly consists of all keys in normal form relative to $L$. 

12
Remark 42. In the previous example, note that as soon as \{\varepsilon\} is chosen, the substitutions induced by other members of the characteristic partition are irrelevant. Reflections of this kind would allow the definition of versions of NormForm, with improved running time.

Lemma 43. NormForm, is polynomial time computable for each \( n \in \mathbb{N} \).

Proof. Note that \( gdv(L) \) can be calculated in polynomial time in \( L \). In addition, the size of the characteristic partition of \( gdv(L) \) is bounded by the size of the smallest term of \( L \). These facts easily imply the lemma. \( \square \)

Lemma 44. Let \( L \) be a language. If \( u \in \text{NormForm}_n(L) \), then \( u \) is in normal form relative to \( L \)

Proof. We denote the characteristic partition of \( gdv(L) \) as \( P_1, \ldots, P_m \). Assume that \( L \) satisfies the non-trivial equation \( \alpha_i = q \) for \( q \) being a subterm of \( u \) with head at position \( \bar{p} \) of \( u \). By definition of the algorithm NormForm, there is a \( P_j = \{p_0, \ldots, p_\nu\} \) with \( 1 \leq j \leq m \) such that \( \alpha_i \) occurs exactly at all positions of \( P_j \) occurring in \( u \). Since \( \bar{p} \) is not element of \( P_j \) we have

\[
(A) \quad gdv(L)|_{\bar{p}} \neq gdv(L)|_{p_i}
\]

for \( 1 \leq i \leq \nu \). Because of lemma 39 and lemma 25 this implies the existence of a term \( v \in L \) such that \( v \) falsifies equation \( (A) \) relative to key \( gdv(L) \). This immediately implies that \( v \) also falsifies the equation \( \alpha_i = q \) relative to key \( u \), contradicting the assumption. For \( q \) being a closed term, we argue similarly. \( \square \)

Lemma 45. Let \( u \) be in normal form relative \( L \) containing the non-terminals \( \alpha_1, \ldots, \alpha_n \). Let \( P_1, \ldots, P_m \) be the characteristic partition of \( gdv(L) \). Then the following two properties hold.

- Each \( \alpha_i \) for \( 1 \leq i \leq n \) only occurs at the positions of a unique \( P_j \) for \( 1 \leq j \leq m \).
- If \( \alpha_i \) occurs at a position of a \( P_j \) for \( 1 \leq j \leq m \) then it occurs at all positions of \( P_j \) occurring in \( u \).

Proof. Let us prove the first property. Assume that \( \alpha_i \) for \( 1 \leq i \leq n \) does not occur at a position of any \( P_j \) for \( 1 \leq j \leq m \). This implies that \( \alpha_i \) only occurs at positions \( p \) with \( gdv(L)|_p = r \) and \( r \) not containing any non-terminals. This implies that \( L \) satisfies \( \alpha_i = r \) contradicting the assumption. Assume that \( \alpha_i \) occurs at the positions \( p_0 \in P_j \) and \( p_1 \in P_k \) with \( j \neq k \). Since \( gdv(L) \) is in normal form there is a \( v \in L \) refuting \( gdv(L)|_{p_0} = gdv(L)|_{p_1} \) relative to \( gdv(L) \). This easily implies that \( u \) is not a key of \( \{v\} \) which is a contradiction.

We assume the first property. Let us prove the second property. Assume towards a contradiction that there is a \( P_j = \{p_0, \ldots, p_\nu\} \) for \( 1 \leq j \leq m \) such that \( \alpha_i \) for \( 1 \leq i \leq n \) does not occur at the position \( p_0 \) of \( u \) but occurs at other positions in \( P_j \) which are contained in \( u \). But then it can be seen easily that the equation \( \alpha_i = u|_{p_0} \) holds in \( L \) relative to \( u \) contradicting the assumption. \( \square \)

Lemma 46. Let \( u \) be in normal form relative \( L \) containing the non-terminals \( \alpha_1, \ldots, \alpha_n \). Then \( u \in \text{NormForm}_n(L) \).

Proof. The previous lemma easily implies that there is a list \( P_1, \ldots, P_n \) given as in the description of NormForm, such that the non-terminals \( \alpha_1, \ldots, \alpha_n \) of \( u \) occur exactly at the positions \( P_1, \ldots, P_n \) which occur in \( u \). The term new – key produced by inserting variables at these positions equals \( u \). \( \square \)
Lemma 47. Let $L$ be a language, and $n$ a natural number. Then, $\text{NormForm}_n$ satisfies its specification, i.e. $\text{NormForm}_n(L)$ contains exactly all keys in normal form relative to $L$ containing at most non-terminals $\alpha_1, \ldots, \alpha_n$.

Proof. The lemma is implied by lemmas 44 and 46. □

4.3. Definition of algorithm $\text{SuffKeys}_n$

We will define the algorithm $\text{SuffKeys}_n$ constructing $S_{L,n}$ efficiently from input $L$. The results of sections 3, and the results of this section easily imply that $\text{SuffKeys}_n$ is correct and computable in polynomial time.

Algorithm 4. $\text{SuffKeys}_n$

Input: Language $L$
Output: $S_{L,n}$
Execution:

• Calculate $st(L)$.
• For each subset $L'$ of $st(L)$ with $1 \leq |L'| \leq n + 1$, add all keys in the set $\text{NormForm}_n(L')$ to the output set initialised as $\emptyset$.

end

Lemma 48. For each $n \in \mathbb{N}$ and each language $L$, $\text{SuffKeys}_n$ is polynomial time computable, and $\text{SuffKeys}_n(L)$ is a sufficient set of keys for $L$

Proof. The first statement follows from the algorithm $\text{NormForm}_n$ being computable in polynomial time for each $n \in \mathbb{N}$ which was proved as lemma 43. The second statement follows from theorem 33, and the correctness of algorithm $\text{NormForm}_n$. □

5. Reduction to a $\text{MinCostSat}$ problem

In the last section, we showed how $S_{L,n}$ can be computed efficiently. $S_{L,n}$ contains all keys necessary to produce a minimal trat-n grammar $G$ with $L \subseteq \mathcal{L}(G)$. However, the most difficult problem in the computation of the searched minimal trat-n grammar $G$, from the complexity point of view, is the choice of an optimal set of rules with right sides contained in $S_{L,n}$. In this section, we show how this problem can be reduced to the $\text{MinCostSat}$-problem defined as follows:

Definition 49 ($\text{MinCostSat}$). An instance $\mathcal{P}$ of $\text{MinCostSat}$ is determined by a Boolean formula $F$ with $n$ atoms $x_1, \ldots, x_n$ with costs $c_i \in \mathbb{N}_{\geq 0}$ for $1 \leq i \leq n$. A solution of $\mathcal{P}$ is an evaluation $E$ such that $E(F) = \text{true}$ that minimises $c = \sum_{1 \leq i \leq n, E(x_i) = \text{true}} c_i$.

Remark 50. The question whether there exists a solution $E$ of an instance $\mathcal{P}$ of $\text{MinCostSat}$ such that $\sum_{1 \leq i \leq n, E(x_i) = \text{true}} c_i < n \in \mathbb{N}$ is NP-complete in its input $\mathcal{P}, n$. □
MinCostSat has a high practical relevance, and finding algorithms which solve MinCostSat efficiently is an active topic of research, see e.g. [5] and [17]. Many NP-optimisation problems are special cases of, or closely related to MinCostSat. E.g. the set covering problem is a special case of MinCostSat. In addition, MinCostSat can be easily reduced to partial MaxSat (see definition 62), and linear optimisation. Because of these reasons, even a broader set of optimised algorithms is available for MinCostSat. In the next section, we recommend some algorithms to solve the instances of MinCostSat which occur in the computation of Decom$\alpha$.

Now, let us define the algorithm MCS which executes the above mentioned reduction of a key selection problem to a MinCostSat instance.

**Algorithm 5. MCS**

Input: Language $L$, set of terms $S$ containing at most non-terminals $\alpha_1, \alpha_2, \ldots, \alpha_n$ for some $n > 0$ (the intended second input is a sufficient set of keys of $n$ non-terminals for the first input, e.g. $S_{L,n}$).

Output: A certain instance $P$ of MinCostSat.

Execution: Output as formula $F$ of $P$, given as in definition 49, the formula $C_{L,S}$ defined in definition 54. The cost function of $P$ is given as follows: For atoms $x_j$ of $C_{L,S}$ of the form $x_{i,k}$, we define $c_j := 1$. For all other atoms $x_j$, we define $c_j := 0$.

We will define the formula $D_{L,S}(t, \ell, q)$ which intuitively means that a term $t \in st(L)$ which is assumed to occur as $\ell$-rest in a derivation of $q \in L$ in $G_{L,S,E}$ ($\ell$-rest: defined in definition 4).

**Definition 51 ($D_{L,S}(t, \ell, q)$).** Assume that $L$ is a language, and $S$ a set of terms containing at most non-terminals $\alpha_1, \alpha_2, \ldots, \alpha_n$ for some $n > 0$. Assume $t \in st(L)$, $0 \leq \ell \leq n$, and $q \in L$. Assume that the following elements of $S$ only containing non-terminals $\alpha_i$ with $i > \ell$ are keys of $\{t\}$: $k_1, \ldots, k_s$. Assume $1 \leq j \leq s$. Let $\#nt(k_j)$ denote the number of non-terminals of $k_j$. Let $nt(i, k_j)$ denote the subindex of the $i$-th smallest\footnote{Relative to the usual ordering on the non-terminals.} non-terminal of $k_j$ for $1 \leq i \leq \#nt(k_j)$. Assume

$$k_j \circ_{t,1,j} \alpha_{nt(1,k_j)}, \ldots, \alpha_{nt(\#nt(k_j),k_j)} \begin{pmatrix} r_{1,1} \vdots \vdots \vdots r_{\#nt(k_j),j} \end{pmatrix} = t,$$

\begin{align*}
\text{where } r_{\#nt(k_j),j} &\text{ is the number of atoms of } C_{L,S} \text{ which are } (i,j)\text{-rest in a derivation of } q \in L \text{ in } G_{L,S,E}.
\end{align*}
respectively \( k_j = t \) if \( \#nt(k_j) = 0 \) for each \( 1 \leq j \leq s \). Then \( D_{L,S}(t, \ell, q) \) is defined as follows.

\[
\bigvee_{1 \leq j \leq s} x_{\ell,k_j} \land x_{(r_{1,j}),nt(1,k_j),q} \land \cdots \land x_{(r_{(\#nt(k_j)),j}),nt(\#nt(k_j),k_j),q}
\]

Note that in the case where \( k_j = t \), the disjunct for \( j \) in \( D_{L,S}(t, \ell, q) \) only consists of the conjunct \( x_{\ell,k_j} \).

The fact that the rigidity condition is satisfied for derivations of a certain \( q \in L \) is imposed by \( R_{L,S}(q) \) which will be defined in the following. \( R_{L,S}(q) \) expresses that there are not two different \( i \)-rests in derivations of \( q \).

**Definition 52 (\( R_{L,S}(q) \)).** Assume that \( L \) is a language, and \( S \) a set of terms containing at most non-terminals \( \alpha_1, \alpha_2, \ldots, \alpha_n \) for some \( n > 0 \). Assume that \( q \in L \). Then, we have

\[
R_{L,S}(q) := \bigwedge_{t_0, t_1 \in st(\{q\}), t_0 \neq t_1, 1 \leq i \leq n} -x_{t_0, i, q} \lor -x_{t_1, i, q}.
\]

The formula \( C_{L,S}(q) \) defined below states that \( q \) is decomposed correctly.

**Definition 53 (\( C_{L,S}(q) \)).** Assume that \( L \) is a language, and \( S \) a set of terms containing at most non-terminals \( \alpha_1, \alpha_2, \ldots, \alpha_n \) for some \( n > 0 \). Assume that \( q \in L \). Then, the formula \( C_{L,S}(q) \) is defined as follows.

\[
\left( \bigwedge_{i \in st(\{q\}), 1 \leq i \leq n} x_{t,i,q} \land D_{L,S}(t, i, q) \right) \land D_{L,S}(q, 0, q) \land R_{L,S}(q)
\]

Finally, the formula \( C_{L,S} \) can be defined. It claims that all elements of \( L \) are decomposed correctly.

**Definition 54.** Assume that \( L \) is a language, and \( S \) a set of terms containing at most non-terminals \( \alpha_1, \alpha_2, \ldots, \alpha_n \) for some \( n > 0 \). Then, the formula \( C_{L,S} \) is defined as follows.

\[
C_{L,S} := \bigwedge_{q \in L} C_{L,S}(q)
\]

**Lemma 55.** The algorithm MCS is polynomial time computable.

**Proof.** It can be seen easily that the size of \( C_{L,S} \) can be bounded polynomially in the size of the input \( L, S \) represented in a natural way. From this fact, we easily derive the lemma. \( \square \)

In the rest of the section, we will prove that the intended properties for \( \text{MCS}(L, S) \) hold if \( S \) is a sufficient set of keys of \( n \) non-terminals for \( L \) for any \( n \in \mathbb{N} \).

**Notation 56.**

- The formula \( F \) of the instance \( \text{MCS}(L, S) \) of MinCostSat is denoted as \( \text{MCS}(L, S)^* \).
- The atoms \( x_j \) of \( \text{MCS}(L, S)^* \) with \( c_j = 1 \) are called counting atoms.

**Definition 57.** Let \( S \) be a sufficient set of keys of \( n \) non-terminals for a language \( L \). Let \( E \) be an evaluation of the atoms occurring in \( \text{MCS}(L, S)^* \). Then, the \( t \)-rest grammar \( G_{L,S,E} \) is given as follows:

For each atom of the form \( x_{i,k} \) with \( E(x_{i,k}) = true \), \( G_{L,S,E} \) contains the production \( \alpha_i \rightarrow k \). \( G_{L,S,E} \) does not contain any other productions.
Theorem 58. Let $S$ be a sufficient set of keys of $n$ non-terminals for a language $L$. Assume that the instance $\text{MCS}(L,S)$ of $\text{MinCostSat}$ is solved by the evaluation $E$, evaluating $c \in \mathbb{N}$ counting atoms as true. Then, $L \subseteq L(G_{L,S,E})$ and $|G_{L,S,E}| = c$. In addition, $G_{L,S,E}$ is one of the minimal $\text{trat}$-$n$ grammars $G$ with $L \subseteq L(G)$.

To prove Theorem 58 which implies the correctness of $\text{MCS}(L,S)$, we need some auxiliary definitions and lemmas. The next lemma trivially follows from definition 57.

Lemma 59. Let $S$ be a sufficient set of keys of $n$ non-terminals for a language $L$. Let $E$ be an evaluation of the atoms occurring in $\text{MCS}(L,S)^\ast$. The number of true counting atoms of $\text{MCS}(L,S)^\ast$ relative to the evaluation $E$ equals the size of $G_{L,S,E}$.

Lemma 60. Let $S$ be a sufficient set of keys of $n$ non-terminals for a language $L$. Assume that for an evaluation $E$, we have $E(\text{MCS}(L,S)^\ast) = \text{true}$. Then, we have $L \subseteq L(G_{L,S,E})$.

Proof. Fix a term $q \in L$. $E(D_{L,S}(q,0,0)) = \text{true}$ implies that a certain set of counting atoms is true. From this set, we will produce a derivation of $q$ in $G_{L,S,E}$, and then prove that it is rigid. In the following, we sketch how this derivation can be constructed. $E_{L,S}(D(q,0,0)) = \text{true}$ implies that $E$ evaluates one the disjuncts of the form

$$x_{0,k_j} \land x_{(r_{1,j},nt(1,k_j),q} \land \cdots \land x_{(r_{(\#nt(k_j),j)},nt(\#nt(k_j),k_j),q}$$

as true. This implies that for a certain $1 \leq j \leq s$ the production $\alpha_0 \to k_j$ is available in $G_{L,S,E}$. We use it as first step of a derivation of $q$. The remaining task is to derive the terms $r_{1,j}, \cdots, r_{\#nt(k_j),j}$ from non-terminals $\alpha_{nt(1,k_j)}, \cdots, \alpha_{nt(\#nt(k_j),k_j)}$. We deduce that the atoms

$$x_{(r_{1,j},nt(1,k_j),q} \cdots \land x_{(r_{(\#nt(k_j),j)},nt(\#nt(k_j),k_j),q}$$

are true. Because of $x_{t,i,q} \to D_{L,S}(t,i,q)$ for $t \in \text{st}(\{q\})$ and $1 \leq i \leq n$ we derive that $E$ evaluates disjuncts as true which code a decomposition of each of the terms $r_{1,j}, \cdots, r_{\#nt(k_j),j}$ using some keys $k'_1, \cdots, k'_{\#nt(k_j)}$ of $S$. Accordingly, the derivation of $q$ is extended by replacing all occurrences of $\alpha_{nt(i,k_j)}$ in $k_j$ by $k'_i$ for all $1 \leq i \leq \#nt(k_j)$. These replacements can be executed in any order.

We will again deduce the truth of atoms of the form $x_{r,nt,q}$ and extend the derivation analogously as described before. The derivation is extended further until all non-terminals will finally be replaced by closed terms. The resulting derivation $\delta$ clearly is a derivation of $q$.

We have to show that $\delta$ is rigid. Towards a contradiction assume that we have two different $i$-rests $t_0$ and $t_1$ for $1 \leq i \leq n$. But this immediately implies $E(x_{t_0,i,q}) = \text{true}$ and $E(x_{t_1,i,q}) = \text{true}$ which contradicts $E(R_{L,S}(q)) = \text{true}$. $\square$

Lemma 61. Let $S$ be a sufficient set of keys of $n$ non-terminals for a language $L$. Let $G$ be a $\text{trat}$-$n$ grammar with $L \subseteq L(G)$, with $G$ only containing keys in $S$. Then there is an evaluation $E$ with $E(\text{MCS}(L,S)^\ast) = \text{true}$ with $G = G_{L,S,E}$.

Proof. The previous proof yields a method to obtain a derivation from an evaluation. This method can be easily inverted, yielding an evaluation from a derivation. Now, for each $q \in L$, fix a (rigid) derivation of $q$ in $G$, and evaluate the corresponding atoms true. In addition, for each rule of the form $\alpha_i \to k$ in $G$, evaluate $x_{i,k}$ as true. It is easy to see that the obtained evaluation has the required properties. $\square$
We are ready to prove Theorem 58.

Proof. The required properties $L \subseteq L(G_{L,S,E})$ and $|G| = c$ follow from lemmas 60, and 59. The minimality of $G_{L,S,E}$ follows from lemma 61, 59, and the definition of sufficient sets of keys (see definition 28).

6. Solving the MinCostSat instance

As third step of Decom$n$, an instance $MCS(L,S)$ of MinCostSat has to be solved which is obtained from $L$ and a sufficient set of keys $S$ produced from $L$ in the first step of Decom$n$. In this section, we propose algorithms which efficiently solve such problems. Efficient algorithms for MinCostSat have been produced e.g. in Fu and Malik’s [5]. In experiments they have solved instances with several thousands of variables and clauses, and found minimal evaluations with several hundreds of variables of non-zero cost evaluated as true.

The instances $P$ of MinCostSat occurring in the calculation of Decom$n$ only have costs $c_i$ equal to 0 or 1. Such instances of MinCostSat can be translated particularly easily into instances of the partial MaxSat problem, defined as follows.

Definition 62 (Partial MaxSat). An instance $P$ of partial MaxSat is determined by a pair $(F,G)$ of Boolean formulas with $G$ being in CNF. A solution of $P$ is an evaluation $E$ with $E(F) = true$ which maximises the number of clauses $G_i$ of $G$ with $E(G_i) = true$.

The above mentioned translation for a MinCostSat instance $P$ given as in definition 49 with costs 0 and 1 is achieved by defining the formula $F$ of the corresponding instance of the partial MaxSat problem as $F$, and the formula $G$ as the set of clauses of the form $\{\neg x_i\}$ with $c_i = 1$.

For the partial MaxSat problem efficient algorithms have been produced in Fu and Malik’s [4], in Heras, Larrosa, and Oliveras’ [6], and in Koshimura, Zhang, Fujita, and Hasegawa’s [15].

Obviously, MinCostSat problems can be reduced to linear optimisation problems, i.e. linear programming problems, for explanations see e.g. [17, pages 6 and 7]. Especially for MinCostSat problems for which a satisfying assignment can be found trivially, and the main challenge therefore represents the minimisation of the cost function, algorithms from the linear optimisation community seem to be more efficient than algorithms from the Sat community (see [5], pages 857 and 858). This is because the algorithms from the Sat community derive their strength from the conflict driven clause learning. In cases where a satisfying assignment can be found trivially, only few such conflicts will occur.

Clearly, the main challenge for the instances of MinCostSat occurring in the computation of Decom$n$ is the minimisation of the cost function and not the satisfaction of the formula $MCS(L,S)^*$ for all $n \in \mathbb{N}$. This suggests that Decom$n$ will achieve the best performance using an efficient algorithm for linear optimisation in its third step, as e.g. algorithms from cplex.

7. Definition of algorithm Decom

We are now in a position to define Decom$n$ for $n \in \mathbb{N}$.

---

5The first two cited sources present algorithms for the slightly more general weighted MaxSat problem, for its definition see e.g. [6].

6Cplex refers to the IBM ILOG CPLEX Optimization Studio, which is i.a. a package for linear optimisation. For details, please visit http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
Algorithm 6. Decomp\textsubscript{n}

Input: Language \(L\).
Output: A minimal trat-n grammar \(G\) with \(L \subseteq \mathcal{L}(G)\).
Execution:

\begin{itemize}
  \item Calculate \(S := \text{SuffKeys}\textsubscript{n}(L)\).
  \item Calculate \(\text{MCS}(L,S)\).
  \item Find a solution \(E\) of \(\text{MCS}(L,S)\) using e.g. algorithms from cplex.
  \item Output the grammar \(G_{L,S,E}\) obtained from \(E\) as described in definition 57.
\end{itemize}

Theorem 63. For each language \(L\) and each \(n \in \mathbb{N}\), \(\text{Decomp}\textsubscript{n}(L)\) outputs a minimal trat-n grammar \(G\) with \(L \subseteq \mathcal{L}(G)\).

Proof. The theorem follows immediately from theorem 29, the definition of \(\text{SuffKeys}\textsubscript{n}\), and theorem 58.

Remark 64. The running time of \(\text{Decomp}\textsubscript{n}\) grows at least exponentially in \(n\) because the running time of \(\text{SuffKeys}\textsubscript{n}\) grows exponentially in \(n\). The most expensive step of \(\text{Decomp}\textsubscript{n}\) for a fixed \(n\) is the calculation of the solution of \(\text{MCS}(L,S)\). However, optimised algorithms for such problems are available.

8. Conclusion

In this paper we have presented a practically efficient algorithm for compressing a finite tree language by a totally rigid acyclic tree grammar with a fixed number of non-terminal symbols. This grammar compression problem is the combinatorial gist of the lemma generation technique introduced in [9] and further extended in [10, 11]. The mathematical key insight is that a polynomial time computable subset of productions is sufficient for a minimal grammar. This allows the polynomial reduction of the grammar compression problem to a \textit{MinCostSat} problem for which highly efficient solvers are available.

The algorithm presented in the paper is about to become a central ingredient of our implementation\textsuperscript{7} of these lemma generation techniques. It considerably improves the simple algorithm introduced in [9] and we are convinced that it hence paves the way for the efficient treatment of larger classes of formal proofs such as that considered in [10].

References


\textsuperscript{7}http://www.logic.at/gapt/


