Finite Automata for the Sub- and Superword Closure of CFLs: Descriptional and Computational Complexity[⋆]

Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund

Technische Universität München, {bachmeie,luttenbe,schlund}@in.tum.de

Abstract. We answer two open questions by (Gruber, Holzer, Kutrib, 2009) on the state-complexity of representing sub- or superword closures of context-free grammars (CFGs): (1) We prove a (tight) upper bound of $2^{\mathcal{O}(n)}$ on the size of nondeterministic finite automata (NFAs) representing the subword closure of a CFG of size n. (2) We present a family of CFGs for which the minimal deterministic finite automata representing their subword closure matches the upper-bound of $2^{2^{\mathcal{O}(n)}}$ following from (1). Furthermore, we prove that the inequivalence problem for NFAs representing sub- or superword-closed languages is only NP-complete as opposed to PSPACE-complete for general NFAs. Finally, we extend our results into an approximation method to attack inequivalence problems for CFGs.

1 Introduction

Given a (finite) word $w = w_1 w_2 \dots w_n$ over some alphabet Σ, we say that u is a *(scattered) subword or subsequence* of w if u can be obtained from w by erasing some letters of w. We denote the fact that u is a subword of w by $u \preccurlyeq w$, and alternatively say that w is a *superword* of u. As shown by Higman [10] in 1952 $\preccurlyeq$ is a well-quasi-order on Σ^*, implying that *every* language $L \subseteq \Sigma^*$ has a finite set of $\preccurlyeq$-minimal elements. This proves that both the subword (also: downward) closure $\nabla L := \{u \in \Sigma^* \mid \exists w \in L \colon u \preccurlyeq w\}$ and the superword (also: upward) closure $\Delta L := \{w \in \Sigma^* \mid \exists u \in L \mid u \preccurlyeq w\}$ are regular for *any* language L. While in general, we cannot effectively construct a finite automaton accepting ∇L resp. ΔL, for specific classes of languages effective constructions are known.

It is well-known that this is the case when when L is given as a context-free grammar (CFG). This was first shown by van Leeuwen [18] in 1978. Later, Courcelle gave an alternative proof of this result in [5]. Section 3 builds up on these results by Courcelle. We also mention that for Petri-net languages an effective construction is known thanks to Habermehl, Meyer, and Wimmel [9].

These results can be used to tackle undecidable questions regarding the ambiguity, inclusion, equivalence, universality or emptiness of languages by over-approximating one or both languages by suitable regular languages [13,12,7,9]:

[⋆] This work was partially funded by the DFG project "Polynomial Systems on Semirings: Foundations, Algorithms, Applications".

For instance, consider the scenario where we are given a procedural program whose runs can be described as a pushdown automaton resp. a CFG G_1 and a context-free specification G_2 of all safe executions, and we want to check whether all runs of the system conform to the safety specification $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$. As $\mathcal{L}(G_1) \cap \overline{\nabla\mathcal{L}(G_2)} \neq \emptyset \Rightarrow \mathcal{L}(G_1) \not\subseteq \mathcal{L}(G_2)$, we can obtain at least a partial answer to the otherwise undecidable question. Of course, in the case $\mathcal{L}(G_1) \subseteq \nabla\mathcal{L}(G_2)$ no information is gained, and one needs to refine the problem e.g. by using some sort of counter-example guided abstraction refinement as done e.g. in [12].

Contributions and Outline Our first results (Sections 3 and 4) concern the blow-up incurred when constructing a (non-)deterministic finite automaton (NFA resp. DFA) for the subword closure of a language given by a context-free grammar G where we improve the results of [8]: For a CFG G of size n, [8] shows that an NFA recognizing $\nabla\mathcal{L}(G)$ has at most $2^{2^{\mathcal{O}(n)}}$ states, and there are CFGs requiring at least $2^{\Omega(n)}$ states. (For linear CFGs the upper and lower bounds are both single exponential.) The upper bound of [8] is established by analyzing the inductive construction of [18]. We improve this result in Section 3 to $2^{\mathcal{O}(n)}$ by slightly adapting Courcelle's construction [5] (we also briefly discuss that naively applying Courcelle's construction cannot do better than $2^{\Omega(n \log n)}$ in general). This result of course yields immediately an upper bound of $2^{2^{\mathcal{O}(n)}}$ on the size of minimal DFA representing accepting $\nabla\mathcal{L}(G)$. In Section 4 we show this bound is tight already over a binary alphabet. To the best of our knowledge, so far only examples were known which showcase the single-exponential blow-up when constructing an NFA accepting the subword closure of a context-free grammar[8] resp. a DFA accepting the subword closure of a DFA or NFA [15]. We then study in Section 5 the equivalence problem for NFAs recognizing subword- resp. supword-closed languages. While for general NFAs this problem is PSPACE-complete, we show that it becomes coNP-complete under this restriction. We combine these results in Section 6 to derive a conceptual simple semi-decision procedure for checking language-inequivalence of two CFGs G_1, G_2: we first construct NFAs for $\nabla\mathcal{L}(G_1)$ and $\nabla\mathcal{L}(G_2)$, and check language-inequivalence of these NFAs; if the NFAs are inequivalent, we construct a witness of the language-inequivalence of G_1 and G_2; otherwise we refine the grammars, and repeat the test on the so obtained new grammars. This approach is motivated by the abstraction-refinement approach of [12] for checking if the intersection of two context-free languages is empty. We experimentally evaluate our approach by comparing it to *cfg-analyzer* of [2] which uses incremental SAT-solving to tackle the language-inequivalence problem.

2 Preliminaries

By Σ we denote a finite alphabet. For every natural number n, let $\Sigma^{\leq n}$ denote the words of length at most n over Σ. The empty word is denoted by ε; the set of all finite words by Σ^*.

We measure the *size* $|G|$ of a CFG G as the total number of symbols on the right hand sides of all productions. The size of an NFA is simply measured as the number of states (this is an adequate measure for a constant alphabet, since the number of transitions is at most quadratic in the number of states).

Throughout the paper we will always assume that all CFGs are reduced, i.e. do not contain any unproductive or unreachable nonterminals (any CFG can be reduced in polynomial time). Let X be a nonterminal in a CFG G. We define $\mathcal{L}(X)$ as the set of all words $w \in \Sigma^*$ derivable from X. If S is the start symbol of G, then $\mathcal{L}(G) := \mathcal{L}(S)$. Moreover, $\Sigma_X \subseteq \Sigma$ denotes the set of all terminals reachable from X. Overloading notation we sometimes write ∇X for $\nabla \mathcal{L}(X)$.

The dependency graph of a CFG G is the finite graph with nodes the nonterminals of G where there is an edge from X to Y if there is a production $X \to \alpha Y \beta$ in G. We say that X *depends directly on* Y (written as $X \rhd Y$) if $X \neq Y$ and there is an edge from X to Y. The reflexive and transitive closure of $\rhd$ is denoted by $\trianglerighteq^*$. We write $X \equiv Y$ if $X \trianglerighteq^* Y \wedge Y \trianglerighteq^* X$, i.e. if X and Y are located in a common strongly-connected component of the dependency graph. We say that G is strongly connected if the dependency graph is strongly connected.

From [5] we recall some useful facts concerning the subword closure:

Lemma 1. *For any nonterminals X, Y, Z in a CFG G it holds that:*

1. $\nabla(\mathcal{L}(X) \cup \mathcal{L}(Y)) = \nabla\mathcal{L}(X) \cup \nabla\mathcal{L}(Y)$
2. $\nabla(\mathcal{L}(X) \cdot \mathcal{L}(Y)) = \nabla\mathcal{L}(X) \cdot \nabla\mathcal{L}(Y)$
3. $X \equiv Y \Rightarrow \nabla X = \nabla Y$
4. *If* $X \to^* \alpha Y \beta Z \gamma$ *for* $Y, Z \equiv X$ *then* $\nabla X = \Sigma_X^*$

3 Computing the Subword Closure of CFGs

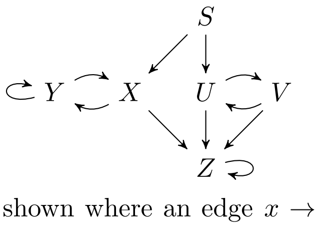
In this section we describe an optimized version of the construction in [5] to compute an NFA for the subword closure of a CFG G of size $2^{\mathcal{O}(|G|)}$, which is asymptotically optimal. We first illustrate the construction by a simple example.

As explained at the end of the next section, a naive implementation of the construction of [5] leads to an automaton of size $2^{\Omega(n)} n! = 2^{\Omega(n \log n)}$ whereas our approach achieves the (optimal) bound of $2^{\mathcal{O}(n)}$.

3.1 Construction by Example

Consider the grammar G with start symbol S defined by the productions:

$$\begin{array}{ll} S \to XaU \mid UaU \mid X & X \to ZbY \mid \varepsilon \\ Y \to XYa \mid b & U \to VZ \mid acb \\ V \to ZU \mid \varepsilon & Z \to cZ \mid bc \end{array}$$

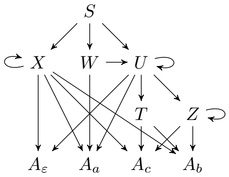


On the right-hand side, the dependency graph is shown where an edge $x \to y$ stands for $x \trianglerighteq y$. To simplify the construction, we first transform the grammar

G into a certain normal form G' (with $\nabla\mathcal{L}(G) = \nabla\mathcal{L}(G')$) and then construct an NFA from G'.

In the first step we compute the SCCs of G, here $\{X, Y\}$ and $\{U, V\}$. Since $Y \to XYa$ (with $Y \equiv X$ and $X \equiv X$), we know that $\nabla Y = \nabla X = \Sigma_X^* = \{a, b, c\}^*$. We therefore can replace any occurrence of Y by X (thereby removing Y from the grammar) and redefine the rules for X to $X \to aX \mid bX \mid cX \mid \varepsilon$. In case of the SCC $\{U, V\}$ the grammar is linear w.r.t. U and V, i.e. starting from either of the two we can never produce sentential forms in which the total number of occurrences of U and V exceeds one. Hence, we can identify U and V without changing the subword closure. Finally, we introduce unique nonterminals for each terminal symbol and restrict the right-hand side of each production to at most two symbols by introducing auxiliary nonterminals W and T:

$$\begin{array}{llll}
S & \to XW \mid UW \mid X & W & \to A_a U \\
X & \to A_a X \mid A_b X \mid A_c X \mid A_\varepsilon & U & \to UZ \mid ZU \mid A_a T \mid A_\varepsilon \\
T & \to A_c A_b & Z & \to A_c Z \mid A_b A_c \\
A_a & \to a & A_b & \to b \\
A_c & \to c & A_\varepsilon & \to \varepsilon
\end{array}$$



Note that the dependency graph of this transformed grammar is now acyclic apart from self-loops. Because of this, we can directly transform the grammar into an *acyclic* equation system (or straight-line program, or algebraic circuit) whose solution is a regular expression for ∇S:

$$\begin{array}{llll}
\nabla A_a & = (a + \varepsilon) & \nabla A_b & = (b + \varepsilon) \\
\nabla A_c & = (c + \varepsilon) & \nabla A_\varepsilon & = \varepsilon \\
\nabla Z & = c^*(\nabla A_b \nabla A_c) & \nabla T & = \nabla A_c \nabla A_b \\
\nabla U & = \Sigma_Z^*(\nabla A_a \nabla T)\Sigma_Z^* & \nabla W & = \nabla A_a \nabla U \\
\nabla X & = \Sigma_X^* & \nabla S & = \nabla X \nabla W + \nabla U \nabla W + \nabla X
\end{array}$$

In order to obtain an NFA for ∇S, we evaluate this equation system from bottom to top while re-using as many of the already constructed automata as possible. For instance, consider the equation:

$$\nabla S = \nabla X \nabla W + \nabla U \nabla W + \varepsilon \cdot \nabla X$$

Because of acyclicity of the equation system, we may assume inductively that we have already constructed NFAs $\mathsf{A}_{\nabla X}$, $\mathsf{A}_{\nabla W}$, and $\mathsf{A}_{\nabla U}$ for ∇X, ∇W, and ∇U, respectively. To construct the NFA for ∇S, we first make two copies $\mathsf{A}^{(1)}$, $\mathsf{A}^{(2)}$ of each of these automata. Automata with superscript (1) will be used exclusively for variable occurrences to the left of the concatenation operator, while automata with superscript (2) will be used for the remaining occurrences. We then read quadratic monomials, like $\nabla X \nabla W$, as an ε-transition connecting $\mathsf{A}_{\nabla X}^{(1)}$ with $\mathsf{A}_{\nabla W}^{(2)}$ as shown in Figure 1 where all edges represent ε-transitions.

We do not claim that this construction yields the smallest NFA, but it is easy to describe and yields an NFA of sufficiently small size in order to deduce in the following subsections an asymptotically tight upper bound on the number

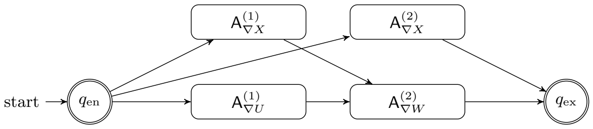

start
q_{en}
$\mathsf{A}^{(1)}_{\nabla X}$
$\mathsf{A}^{(2)}_{\nabla X}$
$\mathsf{A}^{(1)}_{\nabla U}$
$\mathsf{A}^{(2)}_{\nabla W}$
q_{ex}

Fig. 1: Efficient re-use of re-occuring NFAs in Courcelle's construction.

of states. We recall that using a CFG of size $3n+2$ to succinctly represent the singleton language $\{a^{2^n}\}$, the bound of $2^{\Theta(n)}$ follows [8].

In [1] it is remarked that a straight-forward implementation of Courcelle's construction yields an NFA "single exponential" size w.r.t. $|G|$. However, no detailed complexity analysis is given. Consider the CFG with start-symbol A_n and consisting of the rules $A_0 \to a$ and for all $1 \leq k \leq n: \quad A_k \to A_i A_j \quad \forall 0 \leq i, j \leq (k-1)$. If we compute an NFA for ∇A_n via the straight-forward bottom-up construction it will have size $a_n := |\mathsf{A}_{\nabla A_n}|$ with $a_n = 2 + \sum_{0 \leq i,j \leq (n-1)} (a_i + a_j)$. It is easy to show that $a_n \geq 2^n n! \in 2^{\Omega(n \log n)}$. Hence, the crucial part to achieve the optimal bound of $2^{\mathcal{O}(n)}$ is to reuse already computed automata. We just remark that one can also achieve similar savings, by factoring out common terms in the right hand side of the acyclic equations. A subsequent bottom-up construction leads to an NFA of size $2^{\mathcal{O}(n)}$ as well but the constant hidden in the $\mathcal{O}$ is larger and the analysis is more involved. Note that this also shows that we can construct a regular expression of size $2^{\mathcal{O}(n)}$ representing the subword closure.

3.2 Normal Form for Computing the Subword Closure

To simplify our construction, we will assume that our grammar has a special form which is similar to CNF but with unary rules allowed. Any CFG can be transformed into this form with at most linear blowup in size preserving its subword closure (but not its language).

Definition 2. *A CFG G is in quadratic normal form (QNF) if for every terminal $x \in \Sigma \cup \{\varepsilon\}$ there is a unique nonterminal A_x with the only production $A_x \to x$ and every other production is in one of the following forms:*

- *$X \to YX$ or $X \to XY$ (with $Y \neq X$)*
- *$X \to Y$ or $X \to YZ$ (with $Y, Z \neq X$)*

A grammar in QNF is called simple *if*

- *for all $X \to YX$ or $X \to XY$, we have $X \triangleright Y$*
- *for all $X \to Y$ or $X \to YZ$, we have $X \triangleright Y, Z$.*

Note that the dependency graph associated with a grammar in simple QNF is acyclic with the exception of self-loops.

First, we need a small lemma that allows us to eliminate all linear productions "within" some SCC, i.e. productions of the form $X \to \alpha Y \beta$ such that $X \neq Y$ but $Y \trianglerighteq^* X$.

Lemma 3. *Let G be a strongly connected linear CFG with nonterminals $\mathcal{X} = \{X_1, \ldots, X_n\}$ so that every production is either of the form $X \to \alpha Y \beta$ or $X \to \alpha$ for $\alpha, \beta \in \Sigma^*$.*

Consider the grammar G' which we obtain from G by replacing in every production of G every occurrence of a nonterminal X_i by Z.

We then have that $\nabla\mathcal{L}(Z) = \nabla\mathcal{L}(X_i)$ for all $i \in [n]$.

Using the preceding lemma, we can show that it suffices to consider only CFG in simple QNF in the following.

Theorem 4. *Every CFG G can be transformed into a CFG G' in simple QNF such that $\nabla\mathcal{L}(G) = \nabla\mathcal{L}(G')$ and $|G'| \in \mathcal{O}(|G|)$.*

Proof (sketch). First, we use Lemma 1 to simplify all productions involving an X with $X \Rightarrow^* \alpha X \beta X \gamma$. Then we apply Lemma 3 to contract SCCs to a single non-terminal. Finally, we introduce auxiliary variables for the terminals and we binarize the grammar (keeping unary rules like [11]).

Theorem 5. *For any CFG G in simple QNF with n nonterminals there is an NFA A with at most $2 \cdot 3^{n-1}$ states which recognizes the subword closure of G, i.e. $\nabla\mathcal{L}(G) = \mathcal{L}(\mathsf{A})$.*

Proof (sketch). Since the dependency graph of a grammar in simple QNF is a DAG (if we ignore self-loops), we can order the nonterminals according to a topological ordering of this graph. We proceed bottom-up to inductively build an NFA for $\nabla\mathcal{L}(G) = \nabla S$ as in section 3.1. Since our grammar is in QNF, at each stage we only have to produce at most two copies of every automaton representing the subword-closure of a "lower" nonterminals Y. Inductively, for each of these Y we can build NFAs with at most $2 \cdot 3^i$ many states where i is Y's position in the topological ordering. Using the "biparitite wiring" sketched in Figure 1 the size of the automaton for X can then be estimated as

$$|\mathsf{A}_S| \leq 2 + \sum_{Y \,:\, S \triangleright Y} 2 \cdot |\mathsf{A}_Y| \leq 2 + 4 \cdot \sum_{i=0}^{n-2} 3^i = 2 \cdot 3^{n-1}.$$

Corollary 6. *For every CFG G of size n there is an NFA A of size $2^{\mathcal{O}(n)}$ and a DFA D of size $2^{2^{\mathcal{O}(n)}}$ with $\nabla\mathcal{L}(G) = \mathcal{L}(A) = \mathcal{L}(D)$.*

4 CFG → DFA: Double-exponential Blowup

As seen in the preceding section, moving from a context-free grammar G representing a subword-closed language to a language-equivalent NFA $\mathcal{A}$, the size of the automaton is bounded from above by $2^{O(|G|)}$. For superword closures [8] prove the same upper bound for the size of the NFA. From both results we immediately obtain the upper bound $2^{2^{O(|G|)}}$ on the size of the minimal language-equivalent DFA recognizing the sub- or superword closure of a CFG G. This bound is essentially tight as witnessed by the family of finite languages

$$L_k = \bigcup_{j=1}^{k} \{0,1\}^{j-1}\{0\}\{0,1\}^{k}\{0\}\{0,1\}^{k-j}.$$

L_k contains exactly all those words $w \in \{0,1\}^{2k+1}$ which contain two 0s which are separated by exactly k letters. Using the idea of iterated squaring in order to succinctly encode the language $\{a^{2^n}\}$ as a context-free grammar (resp. straight-line program) of size $\mathcal{O}(n)$, also the language L_{2^n} can be represented by a context-free grammar of size $\mathcal{O}(n)$. One then easily shows that the Myhill-Nerode relation w.r.t. L_{2^n}, ∇L_{2^n}, and ΔL_{2^n}, respectively, has at least 2^{2^n} equivalence classes:

Theorem 7. *There exists a family of CFGs G_n of size $\mathcal{O}(n)$ (generating a finite language) such that the minimal DFA accepting either $L(G_n)$, or $\nabla L(G_n)$, or $\Delta L(G_n)$, has at least 2^{2^n} states.*

5 Equivalence of NFAs modulo Sub-/Superword Closure

As hinted at in the introduction, one application of the sub- resp. superword closure is (in-)equivalence checking of CFGs by regular over-approximation. For this, we must solve the equivalence problems for NFAs representing sub/superword closed languages. Naturally, the question arises how hard this is.

Let A and B denote NFAs over the common alphabet Σ, having n_{A} and n_{B} many states, respectively. Recall that the universality problem for NFAs, i.e. $\mathcal{L}(\mathsf{A}) \overset{?}{=} \Sigma^*$, and hence also the equivalence problem $\mathcal{L}(\mathsf{A}) \overset{?}{=} \mathcal{L}(\mathsf{B})$ are PSPACE-complete. Only recently, it was shown in [16] that these problems *stay* PSPACE-complete even when restricted to NFAs representing languages which are closed w.r.t. either prefixes or suffixes or factors. However, in [16] it was also shown that for subword-closed NFAs (i.e. $\nabla\mathcal{L}(\mathsf{A}) = \mathcal{L}(\mathsf{A})$), universality is decidable in linear time as $\mathcal{L}(\mathsf{A}) = \Sigma^*$ holds if and only if there is an SCC in A whose labels cover all of Σ. It is easily shown that a similar result also holds for superword-closed NFAs (i.e. $\Delta\mathcal{L}(\mathsf{A}) = \mathcal{L}(\mathsf{A})$): We have $\mathcal{L}(\mathsf{A}) = \Sigma^*$ if and only if $\varepsilon \in \mathcal{L}(\mathsf{A})$.

In this section we show that both equivalence problems, i.e. $\nabla\mathcal{L}(\mathsf{A}) \overset{?}{=} \nabla\mathcal{L}(\mathsf{B})$ and $\Delta\mathcal{L}(\mathsf{A}) \overset{?}{=} \Delta\mathcal{L}(\mathsf{B})$, are coNP-complete, hence are easier than in the general case (unless NP = PSPACE). In the following, we write more succinctly $\mathsf{A} \overset{?}{\equiv}_{\nabla} \mathsf{B}$ and $\mathsf{A} \overset{?}{\equiv}_{\Delta} \mathsf{B}$ for these two problems. The following lemma is easy to prove:

Lemma 8. *Let A be an NFA. Define A^{∇} as the NFA we obtain from A by adding for every transition $q \xrightarrow{a} q'$ of A the ε-transition $q \xrightarrow{\varepsilon} q'$. Similarly, define A^{Δ} to be the NFA we obtain by adding the loops $q \xrightarrow{a} q$ for every state q and every terminal $a \in \Sigma$ to A. Then $\nabla\mathcal{L}(\mathsf{A}) = \mathcal{L}(\mathsf{A}^{\nabla})$ and $\Delta\mathcal{L}(\mathsf{A}) = \mathcal{L}(\mathsf{A}^{\Delta})$.*

To prove that both $\mathsf{A} \overset{?}{\equiv}_{\Delta} \mathsf{B}$ and $\mathsf{A} \overset{?}{\equiv}_{\nabla} \mathsf{B}$ are coNP-complete we will give a polynomial bound on the length of a *separating word*, i.e. a word w in the symmetric difference of $\mathcal{L}(\mathsf{A}^{\nabla})$ and $\mathcal{L}(\mathsf{B}^{\nabla})$ resp. of $\mathcal{L}(\mathsf{A}^{\Delta})$ and $\mathcal{L}(\mathsf{B}^{\Delta})$.

We first show that the DFA obtained from A^{∇} resp. $\mathsf{A}^{\mathit{\Delta}}$ using the powerset construction has a particular simple structure:

Lemma 9. *Let* A *be an NFA. Let* $\mathsf{D}_{\mathsf{A}}^{\nabla}$ *(resp.* $\mathsf{D}_{\mathsf{A}}^{\mathit{\Delta}}$*) be the DFA we obtain from* A^{∇} *(resp.* $\mathsf{A}^{\mathit{\Delta}}$*) by means of the powerset construction. For any transition* $S \xrightarrow{a} T$ *of* $\mathsf{D}_{\mathsf{A}}^{\nabla}$ *(*$\mathsf{D}_{\mathsf{A}}^{\mathit{\Delta}}$*) it holds that* $S \supseteq T$ *(resp.* $S \subseteq T$*).*

Thus, the transition relation of $\mathsf{D}_{\mathsf{A}}^{\nabla}$ (disregarding self-loops) can be "embedded" into the lattice of subsets of the states of A, which has height at most $n_{\mathsf{A}} - 1$. Hence the DFA $\mathsf{D}_{\mathsf{A}}^{\nabla}$ has small diameter (although even the minimal DFA for the subword closure can be super-polynomially *larger* than an NFA [15]):

Corollary 10. *With the assumptions of the preceding lemma: The length of the longest simple path in* $\mathsf{D}_{\mathsf{A}}^{\nabla}$ *(resp.* $\mathsf{D}_{\mathsf{A}}^{\mathit{\Delta}}$*) is at most* $n_{\mathsf{A}} - 1$.

To bound the length of a shortest separating word w of two NFAs w.r.t. sub-/superword closure, consider the direct sum of the corresponding DFAs and observe that a run on w either has to "make progress" in the first, or in the second DFA:

Lemma 11. *Let* A *and* B *be two NFAs. If* $\mathsf{A} \not\equiv_{\nabla} \mathsf{B}$ *(resp.* $\mathsf{A} \not\equiv_{\mathit{\Delta}} \mathsf{B}$*), then there exists a separating word of length at most* $n_{\mathsf{A}} + n_{\mathsf{B}} - 2$.

Theorem 12. *The decision problems* $\mathsf{A} \overset{?}{\equiv}_{\nabla} \mathsf{B}$ *and* $\mathsf{A} \overset{?}{\equiv}_{\mathit{\Delta}} \mathsf{B}$ *are in* coNP.

To show coNP-hardness, recall the proof that the equivalence problem for star-free regular expressions is coNP-hard by reduction from TAUT: Given a formula ϕ in propositional calculus, we build a regular expression ρ (without Kleene stars) over $\Sigma = \{0, 1\}$ that enumerates exactly the satisfying assignments of ϕ. Hence, $\rho \in \mathsf{TAUT}$ iff $\mathcal{L}(\rho) = \Sigma^n$ iff $\nabla\mathcal{L}(\rho) = \Sigma^{\leq n}$, since the subword closure can only add new words of length less than n (analogously for $\mathit{\Delta}$).

Theorem 13. *The decision problems* $\mathsf{A} \overset{?}{\equiv}_{\nabla} \mathsf{B}$ *and* $\mathsf{A} \overset{?}{\equiv}_{\mathit{\Delta}} \mathsf{B}$ *are* coNP*-hard.*

6 Application to Grammar Problems

We apply our results to devise an approximation approach for the well-known undecidable problem whether $\mathcal{L}(G_1) = \mathcal{L}(G_2)$ for two CFGs G_1, G_2. Possible attacks on this problem include exhaustive search for a word in the symmetric difference $w \in (L_1 \oplus L_2) \cap \Sigma^{\leq n}$ w.r.t. some increasing bound n e.g. by using incremental SAT-solving [2]. Unfortunately, this quickly becomes infeasible for large problems. Previous work has successfully applied regular approximation for ambiguity detection [17,4] or intersection non-emptiness of CFGs [12].

A high-level description of our approach to (in-)equivalence-checking is given in Figure 2. Of course the procedure will not terminate if $\mathcal{L}(G_1) = \mathcal{L}(G_2)$, so in practice a timeout will be used after which the algorithm will terminate itself and output "Maybe equal". Steps (1) and (2) might take time (at most)

1. Compute NFAs A_1 and A_2 for the subword closures of G_1 and G_2, respectively.
2. Check, if $\mathcal{L}(\mathsf{A}_1) = \mathcal{L}(\mathsf{A}_2)$.
   (a) Case "Not equal": Generate a witness $w \in \mathcal{L}(G_1) \oplus \mathcal{L}(G_2)$.
   (b) Case "Equal": Refine the grammars and restart at **1.**

Fig. 2: Equivalence checking via subword closure approximation.

double exponential in the size of the grammars G_1 and G_2: Recall that the construction of Section 3 yields in the worst-case an NFA A_i whose number of states is exponential in the size of the given CFG G_i. To check if $\nabla\mathcal{L}(G_1) = \nabla\mathcal{L}(G_2)$, an on-the-fly construction of the power-set automaton for $\mathsf{A}_1 \times \mathsf{A}_2$ can be used which terminates as soon as a set of states is reached which contains at least one accepting state of, say, A_1 but no accepting state of A_2. Using Lemma 11, we can safely terminate the exploration of simple paths if their length exceeds the bound stated in Lemma 11. In the worst case this might take time exponential in the size of A_1 and A_2, so at most double exponential in the size of G_1 and G_2.

In the following, we describe in greater detail how we generate a separating word w' in $\mathcal{L}(G_1)$ or $\mathcal{L}(G_2)$ if we find a separating word $w \in \nabla\mathcal{L}(G_1) \oplus \nabla\mathcal{L}(G_2)$, resp. how we refine G_1 and G_2 if $\nabla\mathcal{L}(G_1) = \nabla\mathcal{L}(G_2)$.

6.1 Witness Generation for $\mathcal{L}(G_1) \neq \mathcal{L}(G_2)$

If our check in step (2) returns "Not equal" we know that $\nabla\mathcal{L}(G_1) \neq \nabla\mathcal{L}(G_2)$ and we obtain a word $w \in \nabla\mathcal{L}(G_1) \oplus \nabla\mathcal{L}(G_2)$, w.l.o.g. assume in the following $w \in \nabla\mathcal{L}(G_1) \setminus \nabla\mathcal{L}(G_2)$. This word has length linear in $|\mathsf{A}_1|$ and $|\mathsf{A}_2|$, i.e. at most exponential w.r.t. $|G_1|$ and $|G_2|$.

To obtain a (direct) certificate for the fact that $\mathcal{L}(G_1) \neq \mathcal{L}(G_2)$, we construct a superword $w' \succcurlyeq w$ with $w' \in \mathcal{L}(G_1)$ – such a w' is guaranteed to exist as it is the reason for $w \in \nabla\mathcal{L}(G_1)$. Straight-forward induction on w shows:

Lemma 14. *For $w \in \Sigma^*$ a DFA recognizing $\nabla\mathcal{L}(\{w\})$ resp. $\Delta\mathcal{L}(\{w\})$ and having at most $|w| + 2$ states can be constructed in time polynomial in $|w|$.*

We can therefore intersect G_1 with a DFA accepting $\Delta\mathcal{L}(\{w\})$, to obtain a new CFG G_1' whose size is at most cubic in $|w|$[3,14], i.e. exponential in the size of G_1. From this grammar, we can obtain in time linear in $|G_1'|$ a shortest word w' in $\mathcal{L}(G_1') = \mathcal{L}(G_1) \cap \Delta\mathcal{L}(\{w\})$. The length of w' is at most exponential in $|G_1'|$, i.e. at most double exponential in $|G_1|$.

In practice, shorter witnesses are preferable, so we construct the shortest word in $\overline{\mathcal{L}(\mathsf{A}_2)} \cap \mathcal{L}(G_1)$. In theory this might incur in a triple exponential blow-up resulting from complementing A_2, but we can find a separating word w' which is *not* a superword of w and hence is usually shorter.

6.2 Refinement

In case that the test in step (2) returns "Equal", we refine both grammars such that subsequent subword-approximations may find a counterexample to equality. Assume that our equivalence check yields $\nabla\mathcal{L}(G_1) = \nabla\mathcal{L}(G_2)$. A possible refinement strategy is to cover $L := \nabla\mathcal{L}(G_1)$ using a finite number of regular languages $L \subseteq L' := L_0 \cup L_1 \cup \cdots \cup L_k$ and then to repeat the equivalence check for all pairs of refined languages $\mathcal{L}(G_1) \cap L_i$ and $\mathcal{L}(G_2) \cap L_i$ for all i. The requirement $L' \supseteq L$ protects the refinement from cutting off potential witnesses.

A simple method is covering using prefixes: Here we generate all prefixes $p_1, \ldots, p_k$ of words in L of increasing length (up to some small bound d called the *refinement depth*) and set $L_i := p_i\Sigma^*$ and $L_0 = \nabla\{p_i \mid i \in [k]\}$. Since $\bigcup_i L_i \supseteq L$ this strategy preserves potential witnesses and since any counterexample eventually appears as a prefix, this yields a semi-decision procedure for grammar inequivalence. In our experiments we disregard the finite language L_0 (which can also be checked by enumeration) and only check refinement using the infinite sets $p_i\Sigma^*$ with the goal of quickly finding *some* (not the shortest) distinguishing word. This strategy is often able to tell apart different CFLs after few iterations as shown in the following.

6.3 Implementation and Experiments

We implemented the inequivalence check in an extension[1] of the FPsolve tool [6]. The additional code comprises roughly 1800 lines of C++ and uses libfa[2] to handle finite automata.

Our worst-case descriptional complexity results for the subword closure of CFGs (exponential sized NFA, double-exponential sized DFA) and our remarks on the length of possible counterexamples might suggest that our inequivalence checking procedure is merely of academic interest. Here we briefly show that this is not the case, and that overapproximation via subword closures is actually quite fast in practice.

The paper [2] presents cfg-analyzer, a tool that uses SAT-solving to attack several undecidable grammar problems by exhaustive enumeration. We demonstrate the feasibility of our approximation approach on several slightly altered grammars (cf. [19]) for the PASCAL programming language[3]. The altered grammars were obtained by adding, deleting, or mutating a single rule from the original grammar [19]. We used FPsolve and cfg-analyzer to check equivalence of the altered grammar with the original. Both tools were given a timeout of 30 seconds. We want to stress that we do not strive to replace enumeration-based tools like cfg-analyzer, but rather envision a combined approach: Use overapproximations like the subword closure (with small refinement depth) as a quick check and resort to more computationally demanding techniques like SAT-solving for a thorough test. Also note that it is not too hard to find examples where enumeration-based tools cannot detect inequivalence anymore, e.g. by considering grammars with large alphabet (like C# or Java) for which the shortest word in the language is

[1] The fork is available from `https://github.com/regularApproximation/newton`.

[2] http://augeas.net/libfa/

[3] Available from `https://github.com/nvasudevan/experiment/tree/master/grammars/mutlang/acc` .

already longer than 20 tokens. Here we just showcase an example where both approaches can be fruitfully combined.

Table 1 demonstrates that even if our tool uses the very simple prefix-refinement (which is the main bottleneck in terms of speed), we can successfully solve 100 cases where cfg-analyzer has to give up after 30 seconds and even in cases where both tools find a difference, FPsolve does so much faster.

scenario	# instances	# CA	t_{CA}	#FP	t_{FP}	$\#(CF \wedge FP)$	$t^{\wedge}_{\mathrm{CA}}$	$t^{\wedge}_{\mathrm{FP}}$
add	700	190	17.9	18	2.43	8	10.7	4.97
delete	284	61	17.8	34	0.424	10	14.4	0.464
empty	69	32	18.7	1	1.35	1	5.62	1.35
mutate	700	167	19.1	100	1.3	36	15.8	2.87
switchadj	187	16	20.5	2	5.46	1	9.68	0.34
switchany	328	35	18	9	3.72	8	9.09	2.84
$\sum$	2268	501	–	164	–	64	–	–

Table 1: Numbers of solved instances for different scenarios and respective average times: #CA: solved by cfg-analyzer, #FP: solved by FPsolve, $\#(CA \wedge FP)$: solved by both tools, $t^{\wedge}_{\mathrm{tool}}$: time needed by *tool* on instances from $(CA \wedge FP)$.

7 Discussion and Future Work

Motivated by the language-equivalence problem for context-free languages, we have studied the problems of the space requirements of representing the subword closure of CFGs by NFAs and DFAs, and the computational complexity of the equivalence problem of subword-closed NFAs. We have shown how to construct from a context-free grammar G an NFA accepting $\nabla\mathcal{L}(G)$ consisting of at most $2^{\mathcal{O}(|G|)}$ states – a small gap between the lower bound of $\Omega(2^{|G|})$ and our upper bound of $\mathcal{O}(3^{|G|})$ for grammars in QNF remains for future work. A further question is if this bound can be improved in the case of languages given by as deterministic pushdown automata. We have further shown that the upper-bound on the size of DFA accepting $\nabla\mathcal{L}(G)$ of $2^{2^{\mathcal{O}(|G|)}}$ is tight. Interestingly, a binary alphabet suffices for the presented languag family L_k: for instance the worst-case example of [15], which showcases the exponential blow-up suffered when constructing an DFA for the subword closure of a language given as DFA or NFA, requires an unbounded alphabet. We note that a unary context-free language cannot lead to this double exponential blow-up – this follows from the proof of Theorem 3.14 in [8] (see also Lemma 14 here). Regarding the language-equivalence problem, we have shown that it becomes coNP-complete when restricted to sub- resp. superword-closed NFAs. This is somewhat surprising given the fact that it stays PSPACE-complete for many related families (e.g. for prefix-, suffix-, or factor-closed languages). Finally, we have briefly described an approach to tackle the equivalence problem for CFGs using the presented results, though much work remains to turn our current implementation into a mature tool: In particular, since the intersection of two regular overapproximations is again a regular overapproximation, it could be fruitful to combine the subword closure (or variants like [12]) with other regular approximation techniques like [13]. We also need to improve the refinement of the approximations when scaling the problem size.

References

1. Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the Reachability Analysis of Acyclic Networks of Pushdown Systems. In *CONCUR 2008*, pages 356–371, 2008.
2. Roland Axelsson, Keijo Heljanko, and Martin Lange. Analyzing Context-Free Grammars Using an Incremental SAT Solver. In *ICALP (2)*, pages 410–422, 2008.
3. Yehoshua Bar-Hillel, M. Perles, and E. Shamir. On Formal Properties of Simple Phrase Structure Grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addrison-Wesley 1964, 116–150.
4. Claus Brabrand, Robert Giegerich, and Anders Møller. Analyzing Ambiguity of Context-Free Grammars. *Sci. Comput. Program.*, 75(3):176–191, 2010.
5. Bruno Courcelle. On Constructing Obstruction Sets of Words. *Bulletin of the EATCS*, 44:178–186, 1991.
6. Javier Esparza, Michael Luttenberger, and Maximilian Schlund. FPsolve: A Generic Solver for Fixpoint Equations over Semirings. In *CIAA*, volume 8587 of *LNCS*, pages 1–15. 2014.
7. Pierre Ganty, Rupak Majumdar, and Benjamin Monmege. Bounded underapproximations. *Formal Methods in System Design*, 40(2):206–231, 2012.
8. Hermann Gruber, Markus Holzer, and Martin Kutrib. More on the Size of Higman-Haines Sets: Effective Constructions. *Fundam. Inf.*, 91(1):105–121, January 2009.
9. Peter Habermehl, Roland Meyer, and Harro Wimmel. The Downward-Closure of Petri Net Languages. In *ICALP (2)*, pages 466–477, 2010.
10. Graham Higman. Ordering by Divisibility in Abstract Algebras. *Proc. London Math. Soc.*, s3-2(1):326–336, January 1952.
11. Martin Lange and Hans Leiß. To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm. *Informatica Didactica*, 8, 2009.
12. Zhenyue Long, Georgel Calin, Rupak Majumdar, and Roland Meyer. Language-Theoretic Abstraction Refinement. In *FASE*, pages 362–376, 2012.
13. Mehryar Mohri and Mark-Jan Nederhof. Regular Approximation of Context-Free Grammars through Transformation. In *Robustness in Language and Speech Technology*, volume 17 of *Text, Speech and Language Technology*, pages 153–163. 2001.
14. Mark-Jan Nederhof and Giorgio Satta. Probabilistic Parsing. In *New Developments in Formal Languages and Applications*, pages 229–258. 2008.
15. Alexander Okhotin. On the State Complexity of Scattered Substrings and Superstrings. *Fundam. Inform.*, 99(3):325–338, 2010.

16. Narad Rampersad, Jeffrey Shallit, and Zhi Xu. The Computational Complexity of Universality Problems for Prefixes, Suffixes, Factors, and Subwords of Regular Languages. *Fundam. Inform.*, 116(1-4):223–236, 2012.
17. Sylvain Schmitz. Conservative Ambiguity Detection in Context-Free Grammars. In *ICALP*, pages 692–703, 2007.
18. Jan van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237 – 252, 1978.
19. Naveneetha Vasudevan and Laurence Tratt. Detecting Ambiguity in Programming Language Grammars. In *Software Language Engineering*, volume 8225 of *LNCS*, pages 157–176. 2013.

A Missing proofs

Proof of Lemma 3

Proof. Since G is strongly connected, $\nabla X_i = \nabla X_j$ for all $i, j \in [n]$, hence it suffices to show the statement for X_1. Clearly, $\mathcal{L}(Z) \supseteq \mathcal{L}(X_1)$ hence also $\nabla Z \supseteq \nabla X_1$. For the other inclusion let $w \in \nabla Z$, i.e. we have a word w' with $w \preccurlyeq w' \in \mathcal{L}(Z)$ possessing some derivation $Z \Rightarrow u_0 Z v_0 \Rightarrow u_0 u_1 Z v_1 v_0 \Rightarrow \cdots \Rightarrow w'$. Since G is strongly connected there must be an X_{j_1} reachable from X_1 with $X_{j_1} \rightarrow u_0 X_{k_1} v_0$ for some Y. Continuing this reasoning we generate a superword of w' (with some "junk"-strings α_l, β_l) by following the derivation of w':

$$X_1 \Rightarrow^* \alpha_0 X_{j_1} \beta_0 \Rightarrow \alpha_0 u_0 X_{k_1} v_0 \beta_0 \Rightarrow^* \alpha_0 u_0 \alpha_1 u_1 X_{k_2} v_1 \beta_1 v_0 \beta_0 \Rightarrow \cdots \Rightarrow w''$$

with $w' \preccurlyeq w''$. Since, $w \preccurlyeq w'$ we have $w \in \nabla X_1$.

Proof of Theorem 4

Proof. The following steps achieve the desired result:

1. For every $x \in \Sigma \cup \{\varepsilon\}$ replace every occurrence of x in a production by A_x and finally add the production $A_x \rightarrow x$.
2. For every production $X \rightarrow \alpha Y \beta Z \gamma$ with $Y, Z \equiv X$ replace all productions with lhs Y such that $Y \equiv X$ (i.e. from the same SCC as X) by the productions $X \rightarrow A_x X$ for all $x \in \Sigma_X$ and add $X \rightarrow A_\varepsilon$.
3. Transform the grammar into 2NF, i.e. such that every production is of the form $X \rightarrow \alpha$ with $|\alpha| \leq 2$ (cf. [11]).
4. Contract every strongly connected component of the grammar into a univariate grammar via Lemma 3 [4].

It is easy to check that G' is indeed in simple QNF, moreover steps (1) and (3) do not change the language of the grammar. In step (2) we ensure that $\mathcal{L}(X) = \Sigma_X^*$ if $X \Rightarrow^* \alpha X \beta X \gamma$ (see Lemma 1). Step (4) also preserves the subword closure (by Lemma 3), thus altogether $\nabla \mathcal{L}(G) = \nabla \mathcal{L}(G')$. Step (2) reduces the size of G, steps (1) and (3) lead to a linear growth, and step (4) does not change the size so together there exists a constant c (independent of G) such that $|G'| \leq c \cdot |G|$.

Proof of Theorem 5

Before describing the proof, we state some useful definitions:

Definition 15. *Given a nonterminal X in a grammar in simple QNF with production set P, we define the following sets of nonterminals and terminals:*

- $Q(X) := \{YZ \in \mathcal{X} \cdot \mathcal{X} : (X \rightarrow YZ \in P)\}$ *("quadratic monomials")*

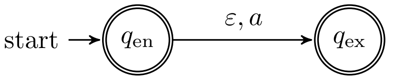
[4] Here we implicitly treat nonterminals from lower SCCs as terminals, since CFLs are closed under substitution this is fine.

- $L(X) := \{Y \in \mathcal{X} : X \to Y \in P\}$ *("linear monomials")*
- $C_l(X) := \{Y \in \mathcal{X} : X \to YX \in P\}$ *("left coefficients")*
- $C_r(X) := \{Y \in \mathcal{X} : X \to XY \in P\}$ *("right coefficients")*
- $\Sigma_l(X) := \Sigma \cap \bigcup\{\nabla L(Y) \mid Y \in C_l(X)\}$ *("left alphabet")*
- $\Sigma_r(X) := \Sigma \cap \bigcup\{\nabla L(Y) \mid Y \in C_r(X)\}$ *("right alphabet")*

Note that $\Sigma_l(X)$ (resp. $\Sigma_r(X)$) is simply the set of terminals reachable from any element of $C_l(X)$ (resp. $C_r(X)$), and therefore can easily be computed. Since G is in simple QNF we have $Y \not\trianglerighteq^* X$ for each Y with $X \triangleright Y$.

Proof. For every nonterminal X of G, let $n(X) = \{Y \mid X \trianglerighteq^* Y\}$ be the number of nodes reachable from X in the dependency graph. We proceed by induction on $n(X)$.

Pick any nonterminal X with $n(X) = 1$. Such an nonterminal has to exist as otherwise the dependency graph would contain a nontrivial cycle. By definition of simple QNF, G can only contain a single rule rewriting X which has to be of the form $X \to a$ for some $a \in \Sigma$. Then the following NFA A_X obviously satisfies $\nabla X = \mathcal{L}(\mathsf{A}_X)$ and $|\mathsf{A}_X| \leq 2 \cdot 3^{n(X)-1}$:



In the following every automaton constructed will have these special states q_{en} and q_{ex} to which we will simply refer to as entry and exit states, respectively.

Now, let X be any remaining nonterminal of G with $n(X) > 0$, i.e. there is at least one nonterminal $Y \neq X$ such that $X \triangleright Y$. By virtue of Lemma 1 and Lemma 3 we have

$$\nabla X = \Sigma_l(X)^* \left(\bigcup_{YZ \in Q(X)} \nabla Y \cdot \nabla Z \cup \bigcup_{Y \in L(X)} \nabla Y \right) \Sigma_r(X)^*.$$

where by definition of simple QNF we have $Y \not\trianglerighteq^* X$ and $Z \not\trianglerighteq^* X$ implying $n(X) > n(Y), n(Z)$. So by induction, we have already constructed for every Y with $X \triangleright Y$ an NFA A_Y such that $\nabla Y = \mathcal{L}(\mathsf{A}_Y)$ i.e.

$$\nabla X = \Sigma_l(X)^* \left(\bigcup_{YZ \in Q(X)} \mathcal{L}(\mathsf{A}_Y) \cdot \mathcal{L}(\mathsf{A}_Z) \cup \bigcup_{Y \in L(X)} \mathcal{L}(\mathsf{A}_Y) \right) \Sigma_r(X)^*.$$

It remains to construct A_X. To this end we use the last equality but only use at most two instances of every automaton A_Y: Initially, we let A_X be the disjoint union of all automata $\{\mathsf{A}_Y^{(i)} \mid i \in [2], X \triangleright Y\}$ where $\mathsf{A}_Y^{(1)}$ and $\mathsf{A}_Y^{(2)}$ denote two distinct copies of A_Y. Here we assume that these states are suitably renamed, in particular, the entry and exit states of all these automata are assumed to be distinct from q_{en} and q_{ex} so that we may add also q_{en} and q_{ex} to the states of A_X. Both q_{en} and q_{ex} are final with q_{en} also the unique initial state of A_X. Finally, we add additional ε-transitions to A_X to mimic the productions rewriting X (see also Subsection 3.1):

- For each $YZ \in Q(X)$: Add ε-transitions (1) from q_{en} to the entry state of $\mathsf{A}_Y^{(1)}$, (2) from the exit state of $\mathsf{A}_Y^{(1)}$ to the entry state of $\mathsf{A}_Z^{(2)}$, and (3) from the exit state of $\mathsf{A}_Z^{(2)}$ to q_{ex}.
- For each $Y \in L(X)$: Add ε-transitions (1) from q_{en} to the entry state of $\mathsf{A}_Y^{(2)}$, and (2) from the exit state of $\mathsf{A}_Y^{(2)}$ to q_{ex}.
- For each $a \in \Sigma_l(X)$: Add a self-loop $q_{\text{en}} \xrightarrow{a} q_{\text{en}}$.
- For each $a \in \Sigma_r(X)$: Add a loop $q_{\text{ex}} \xrightarrow{a} q_{\text{ex}}$.

By induction, we have $|\mathsf{A}_Y| \leq 2 \cdot 3^{n(Y)-1}$ for all Y with $X \triangleright Y$, so $|\mathsf{A}_X|$ is bounded by

$$|\mathsf{A}_X| = 2 + 2 \cdot \sum_{Y\colon X \triangleright Y} |\mathsf{A}_Y| \leq 2 + 4 \cdot \sum_{Y\colon X \triangleright Y} 3^{n(Y)-1}$$

Using breadth-first search, we can assign every nonterminal Z with $X \trianglerighteq^* Z$ a unique number $i(Y) \in [n(X)]$ such that $i(Y) \geq n(Y)$. We then may continue:

$$|\mathsf{A}_X| \leq 2+4 \cdot \sum_{Y\colon X \triangleright Y} 3^{i(Y)-1} \leq 2+4 \cdot \sum_{\substack{Z\colon X \trianglerighteq^* Z \\ Z \neq X}} 3^{i(Z)-1} \leq 2+4 \cdot \sum_{i=0}^{n(X)-2} 3^i = 2 \cdot 3^{n(X)-1}.$$

Proof of Theorem 7

Proof. For $k \in \mathbb{N}$ consider the language L_k of words $w \in \{0,1\}^{2k+1}$ such that $w_j = w_{j+k+1} = 0$ for some $j \in \{1, \dots, k\}$. We can write L_k as

$$L_k = \bigcup_{j=1}^{k} \{0,1\}^{j-1}\{0\}\{0,1\}^k\{0\}\{0,1\}^{k-j}.$$

We in particular interested in L_k for $k = 2^n$. The following CFG of size $\mathcal{O}(n)$ with $L(X_n') = L_{2^n}$ achieves an exponential compression:

$$\begin{array}{lll@{\qquad\qquad}lll}
X_n' & \to & X_{n-1}X_{n-1}' \mid X_{n-1}'X_{n-1} & & & \\
X_{n-1}' & \to & X_{n-2}X_{n-2}' \mid X_{n-2}'X_{n-2} & X_{n-1} & \to & X_{n-2}X_{n-2} \\
 & \vdots & & & \vdots & \\
X_1' & \to & X_0X_0' \mid X_0'X_0 & X_1 & \to & X_0X_0 \\
X_0' & \to & 0Y_n0 & X_0 & \to & 0 \mid 1 \\
Y_n & \to & Y_{n-1}Y_{n-1} & & & \\
Y_{n-1} & \to & Y_{n-2}Y_{n-2} & & & \\
 & \vdots & & & & \\
Y_1 & \to & Y_0Y_0 & & & \\
Y_0 & \to & 0 \mid 1 & & &
\end{array}$$

The grammar uses repeated squaring to achieve the required compression while the "primed" nonterminals X_i' nondeterministically choose where to insert a word from the set $\{0\}\{0,1\}^{2^n}\{0\}$ into a word of $\{0,1\}^{2^n}$.

We show that any two words $w_1, w_2 \in \{0,1\}^{2^n}$ with $w_1 \neq w_2$ are inequivalent w.r.t. the Myhill-Nerode relation of L_{2^n} which implies that the minimal DFA for L_{2^n} must have at least 2^{2^n} states: Consider the first position from the right where w_1 and w_2 differ, so w.l.o.g. we have $w_1 = \alpha 0 \beta$ and $w_2 = \alpha' 1 \beta$ for some $\alpha, \alpha', \beta \in \{0,1\}^*$. As a distinguishing word set $v := 1^{2^n - |\beta|} 0 1^{2^n - |\alpha| - 1}$. Note that

$$w_1 v = \alpha 0 \beta 1^{2^n - |\beta|} 0 1^{2^n - |\alpha| - 1} \in L_{2^n},$$

$$w_2 v = \alpha' 1 \beta 1^{2^n - |\beta|} 0 1^{2^n - |\alpha| - 1} \notin L_{2^n}.$$

The crucial observation is that from $|w_1 v| = |w_2 v| = 2 \cdot 2^n + 1$ it also follows that $w_1 v \in \nabla L_{2^n}$ and $w_2 v \notin \nabla L_{2^n}$ since the subword closure can only add new words of length at most $2 \cdot 2^n$. This shows that also the minimal DFA for ∇L_n must have at least 2^{2^n} states. The very same argument works for ΔL_{2^n}, showing that the minimal DFA for ΔL_{2^n} is of size at least double-exponential in the size of the CFG for L_{2^n} as well.

Proof of Lemma 8

Proof. We start with $\nabla \mathcal{L}(\mathsf{A}) = \mathcal{L}(\mathsf{A}^{\nabla})$: Pick any $w \in \nabla \mathcal{L}(\mathsf{A})$. Then there is some $w' \succcurlyeq w$ such that $w' \in \mathcal{L}(\mathsf{A})$, and thus by construction also $w' \in \mathcal{L}(\mathsf{A}^{\nabla})$. That is there is an accepting run $q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} \ldots \xrightarrow{x_l} q_{l+1}$ with $q_{l+1} \in F$ and $w' = x_0 x_1 \ldots x_l$ (with potentially $x_i = \varepsilon$ for some i). Using the additional ε-transitions of A^{∇} we therefore can turn this sequence into an accepting sequence for w by simply replacing those x_i by ε which do not occur in w. For the other direction, one can reverse this argument by recalling that for any ε-transition $q \xrightarrow{\varepsilon} q'$ added to A^{∇} there is some $a \in \Sigma$ such that $q \xrightarrow{a} q'$ is a transition of A.

Consider now the second claim $\Delta \mathcal{L}(\mathsf{A}) = \mathcal{L}(\mathsf{A}^{\Delta})$: Choose some $w \in \Delta \mathcal{L}(\mathsf{A})$. Then there is some $w' \preccurlyeq w$ such that $w' \in \mathcal{L}(\mathsf{A}) \subseteq \mathcal{L}(\mathsf{A}^{\Delta})$. Any accepting run $q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} \ldots \xrightarrow{x_l} q_{l+1}$ (with $q_{l+1} \in F$ and $w' = x_0 x_1 \ldots x_l$) of A^{Δ} can then be extended to an accepting run of A^{Δ} for w by using the additional loops of A^{Δ} to consume any letters occurring exclusively in w. In the other direction given an accepting run of A^{Δ} we simply strip it by any loops which is guaranteed to yield an accepting run (for a scattered subword) of A as the transition relations of A and A^{Δ} only differ in loops.

Proof of Lemma 9

Proof. Recall that the state (sets) of $\mathsf{D}^{\nabla}_{\mathsf{A}}$ are closed w.r.t. taking ε-successors in A^{∇}. As A^{∇} was obtained from A by introducing for every transition $q \xrightarrow{a} q'$ ($a \in \Sigma$) the ε-transition $q \xrightarrow{\varepsilon} q'$, this means that, if $q \in S$, then every state reachable from q in the directed graph underlying A has to be included in S, too. As for any transition $S \xrightarrow{a} T$ in $\mathsf{D}^{\nabla}_{\mathsf{A}}$, T is a subset of the states reachable from S, the claim follows.

In case of the superword closure, pick any transition $S \xrightarrow{a} T$ of $\mathsf{D}^{\Delta}_{\mathsf{A}}$ and any state $q \in S$. Then by construction of A^{Δ} there is the loop $q \xrightarrow{a} q$ in A^{Δ} which implies that also $q \in T$ by definition of the powerset construction.

Proof of Lemma 11

Proof. Assume $\mathsf{A} \not\equiv_{\nabla} \mathsf{B}$, and let w be a shortest separating word. Consider the unique run of the product DFA $\mathsf{D}_{\mathsf{A}}^{\nabla} \times \mathsf{D}_{\mathsf{B}}^{\nabla}$ on $w = w_0 w_1 \ldots w_l$:

$$(L_0, R_0) \xrightarrow{w_0} (L_1, R_1) \xrightarrow{w_1} \ldots \xrightarrow{w_l} (L_l, R_l).$$

By the preceding lemma we then have $L_i \supseteq L_{i+1}$ and $R_i \supseteq R_{i+1}$ along the run. As w is assumed to be a shortest separating word, it has to hold that $\neg(L_i = L_{i+1} \wedge R_i = R_{i+1})$ for all $i = 1, \ldots, l-1$. In other words, we have

$$n_{\mathsf{A}} + n_{\mathsf{B}} \geq |L_0| + |R_0| > |L_1| + |R_1| > \ldots > |L_l| + |R_l| \geq 2$$

from which the claim immediately follows.

In the case of the superword closure one deduces in the same way that the accepting run for a shortest separating word has to satisfy:

$$2 \leq |L_0| + |R_0| < |L_1| + |R_1| < \ldots < |L_l| + |R_l| \leq n_{\mathsf{A}} + n_{\mathsf{B}}$$

Proof of Thorem 13

Proof. Let φ be a formula of propositional calculus in disjunctive normal form. We construct a regular expression which encodes all satisfying assignments of φ:

Let $x_1, x_2, \ldots, x_n$ be the propositional variables occurring in φ, and assume that $\varphi = \bigvee_{i \in [k]} C_i$ with $C_i = \bigwedge_{j \in [l_i} L_{i,j}$ and $L_{i,j}$ literals. Further, we may assume that in every conjunction C_i is contradiction free. We associate with every C_i a simple regular expression ρ_i enumerating all satisfying assignments of D_i: Initially, set $\rho_i = \emptyset$. Going from $j = 1$ to $j = n$, if x_j occurs in C_i, then set $\rho_i := \rho_i 1$; if $\neg x_j$ occurs in C_i, set $\rho_i := \rho_i 0$; otherwise set $\rho_i := \rho_i(0+1)$. Finally, set $\rho := \rho_1 + \rho_2 + \ldots + \rho_k$. Obviously, the size of ρ is polynomial in the size of φ. Further, we can compute an NFA A from ρ in time polynomial in $|\rho|$, such that $\mathcal{L}(\rho) = \mathcal{L}(\mathsf{A})$. Note that $\mathcal{L}(\mathsf{A}) = \mathcal{L}(\rho) \subseteq \Sigma^n$ by construction. In particular, $\mathcal{L}(\mathsf{A}) = \mathcal{L}(\rho) = \Sigma^n$ if and only if φ is a tautology.

It therefore suffices to show that $\nabla\mathcal{L}(\mathsf{A}) = \Sigma^{\leq n}$ (resp. $\Delta\mathcal{L}(\mathsf{A}) = \Sigma^{\geq n}$) if and only if $\mathcal{L}(\mathsf{A}) = \Sigma^n$. But this is easy as the subword closure resp. superword closure can only add words of length less resp. greater than n.