The Complexity of Downward Closure Comparisons

Georg Zetzsche*

LSV, CNRS & ENS Cachan, Université Paris-Saclay, France
zetzsche@lsv.fr

Abstract

The downward closure of a language is the set of all (not necessarily contiguous) subwords of its members. It is well-known that the downward closure of every language is regular. Moreover, recent results show that downward closures are computable for quite powerful system models.

One advantage of abstracting a language by its downward closure is that then equivalence and inclusion become decidable. In this work, we study the complexity of these two problems. More precisely, we consider the following decision problems: Given languages K and L from classes $\mathcal{C}$ and $\mathcal{D}$, respectively, does the downward closure of K include (equal) that of L?

These problems are investigated for finite automata, one-counter automata, context-free grammars, and reversal-bounded counter automata. For each combination, we prove a completeness result either for fixed or for arbitrary alphabets. Moreover, for Petri net languages, we show that both problems are Ackermann-hard and for higher-order pushdown automata of order k, we prove hardness for complements of nondeterministic k-fold exponential time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Downward closures, Complexity, Inclusion, Equivalence

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.123

1 Introduction

The downward closure of a language is the set of (not necessarily contiguous) subwords of its members. It is a well-known result of Haines [17] that the downward closure of *every* language is regular. Of course, it is not always possible to compute the downward closure of a given language, but oftentimes it is. For example, it has been shown to be computable for such powerful models as *Petri net languages* by Habermehl, Meyer, and Wimmel [14] and *higher-order pushdown automata* by Hague, Kochems, and Ong [15]. A sufficient condition for computability can be found in [34].

Moreover, not only are downward closures often computable, they are also a meaningful abstraction of infinite-state systems. In a complex system, one can abstract a component by the downward closure of the messages it sends to its environment. This corresponds to the assumption that messages can be dropped on the way. Furthermore, recent work of La Torre, Muscholl, and Walukiewicz [32] shows that among other mild conditions, computing downward closures is sufficient for verifying safety conditions of parametrized asynchronous shared-memory systems.

The advantage of having an abstraction of an infinite-state systems as regular languages is that the latter offer an abundance of methods for analysis. An important example is deciding

* This work is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 123; pp. 123:1–123:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

behavioral equivalence or inclusion. This is notoriously hard to do and for nondeterministic infinite-state systems, language equivalence and inclusion are usually undecidable. Using downward closures, such behavioral comparisons can be made in an approximative manner.

Despite these facts, results about the complexity of deciding whether the downward closure of one language includes or equals that of another mainly considered regular languages. Bachmeier, Luttenberger, and Schlund [4] have shown that the equivalence problem for downward closures of two given NFAs is coNP-complete. Karandikar, Niewerth, and Schnoebelen [22] strengthened coNP-hardness to the case of DFAs over binary alphabets and proved coNP-completeness for the inclusion variant. They also obtained NL-completeness of inclusion in the case of NFAs over a unary alphabet. Together with exponential-time downward closure constructions [4, 7, 11, 33, 27], these results imply that equivalence and inclusion are in coNEXP for context-free grammars. Rampersad, Shallit, and Xu [31] proved that one can decide in linear time whether the downward closure of a given NFA contains all words. Subsequently, Karandikar, Niewerth, and Schnoebelen [22] showed that this problem is NL-complete. Similar questions have been studied for upward closures [4, 22].

Previous work on downward closures of infinite-state systems has mainly focused on mere computability [1, 2, 7, 14, 15, 33, 34, 35] or on descriptional complexity [3, 10, 11, 27, 22]. This work studies the complexity of the inclusion and the equivalence problem of downward closures between some prominent types of system models—finite automata, one-counter automata, reversal-bounded counter automata [19], and context-free grammars. More precisely, we are interested in the following questions: For two system models $\mathcal{M}$ and $\mathcal{N}$ and languages L and K generated by some device in $\mathcal{M}$ and $\mathcal{N}$, respectively, what is the complexity of (i) deciding whether $K{\downarrow} \subseteq L{\downarrow}$ (*downward closure inclusion problem*) or (ii) deciding whether $K{\downarrow} = L{\downarrow}$ (*downward closure equivalence problem*)?

Contribution. We determine the complexity of the downward closure inclusion problem and the downward closure equivalence problem among finite automata, one-counter automata, reversal-bounded counter automata (either with a fixed number of counters and reversals or without), and context-free grammars.

For the inclusion problem, we prove completeness results in all cases except for two. The complexities range from coNP over Π_2^{P} to coNEXP (see Table 1). The two cases for which we provide no completeness compare context-free grammars or general reversal-bounded counter automata on the one side with reversal-bounded counter automata with a fixed number of counters and reversals on the other side. However, we prove that both of these problems are coNP-complete for each fixed input alphabet. For the equivalence problem, the situation is similar. We prove completeness for each of the cases except for the combination above. Again, fixing the alphabet leads to coNP-completeness.

The tools developed to achieve these results fall into three categories. First, there are several generic results guaranteeing small witnesses to yield upper bounds. Second, we prove model-specific results about downward closures that yield the upper bounds in each case. Third, we have a general method to prove lower bounds for downward closure comparisons. In fact, it applies to more models than the above: We prove that for Petri net languages, the two comparison problems are Ackermann-hard. For higher-order pushdown automata of order k, we show co-k-NEXP-hardness.

Related work. Another abstraction of formal languages is the well-known Parikh image [28]. The Parikh image of a language $L \subseteq X^*$ contains for each word $w \in L$ a vector in $\mathbb{N}^{|X|}$ that counts the number of occurrences of each letter. For some language classes, it is known that

Table 1 Complexity of the inclusion problem. The entry in row $\mathcal{M}$ and column $\mathcal{N}$ is the complexity of $\mathcal{M} \subseteq_{\downarrow} \mathcal{N}$. Except in the case Ideal $\subseteq_{\downarrow}$ Ideal, all entries indicate completeness. A † means that the entry refers to the fixed alphabet case (for at least two letters).

	Ideal	NFA	OCA	$\mathsf{RBC}_{k,r}$	CFG	RBC
Ideal	$\in$ L	NL	NL	NL	P	NP
NFA	NL	coNP [4, 22]	coNP [3, 4, 22]	coNP	coNP	Π_2^{P}
OCA	NL	coNP [3, 4, 22]	coNP [3, 4, 22]	coNP	coNP	Π_2^{P}
$\mathsf{RBC}_{k,r}$	NL	coNP	coNP	coNP	coNP	Π_2^{P}
CFG	P	coNP	coNP	coNP†	coNEXP	coNEXP
RBC	coNP	coNP	coNP	coNP†	coNEXP	coNEXP

their Parikh image is effectively semilinear, which implies decidability of the inclusion and equivalence problem for Parikh images. The investigation of these problems' complexity has been initiated by Huynh [18] in 1985, who showed that this problem is Π_2^{P}-hard and in coNEXP for regular and context-free languages. Kopczyński and To [23, 24] have then shown that these problems are Π_2^{P}-complete for fixed alphabets. Only very recently, Haase and Hofman [13] have shown that the case of general alphabets is coNEXP-complete.

Due to space restrictions, most proofs can only be found in the full version of this work [36].

2 Concepts and Results

If X is an alphabet, X^* ($X^{\leq n}$) denotes the set of all words (of length $\leq n$) over X. The empty word is denoted by $\varepsilon \in X^*$. For words $u, v \in X^*$, we write $u \preceq v$ if $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$ for some $u_1, \ldots, u_n, v_0, \ldots, v_n \in X^*$. It is well-known that $\preceq$ is a well-quasi-order on X^* and that therefore the *downward closure* $L{\downarrow} = \{u \in X^* \mid \exists v \in L \colon u \preceq v\}$ is regular for every $L \subseteq X^*$ [17]. An *ideal* is a set of the form $Y_0^* \{x_1, \varepsilon\} Y_1^* \cdots \{x_n, \varepsilon\} Y_n^*$, where $Y_0, \ldots, Y_n$ are alphabets and $x_1, \ldots, x_n$ are letters. We will make heavy use of the fact that every downward closed language can be written as a finite union of ideals, which was first discovered by Jullien [21]. By $\mathbb{P}(S)$, we denote the powerset of the set S.

A *finite automaton* is a tuple $\mathcal{A} = (Q, X, \Delta, q_0, Q_f)$, where Q is a finite set of *states*, X is its input alphabet, $\Delta \subseteq Q \times X^* \times Q$ is a finite set of *edges*, $q_0 \in Q$ is its *initial state*, and $Q_f \subseteq Q$ is the set of its *final states*. The language accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$. Sometimes, we write $|\mathcal{A}|$ for the number of states of $\mathcal{A}$.

A *context-free grammar* is a tuple $\mathcal{G} = (N, T, P, S)$ where N and T are pairwise disjoint alphabets, whose members are called the *nonterminals* and *terminals*, respectively. $S \in N$ is the *start symbol* and P is the finite set of *productions* of the form $A \to w$ with $A \in N$ and $w \in T^*$. The language generated by $\mathcal{G}$ is defined as usual.

One-counter Automata. A *one-counter automaton (OCA)* is a nondeterministic finite automaton that has access to one counter that assumes natural numbers as values. The possible operations are *increment*, *decrement*, and *test for zero*. We will not require a formal definition, since in fact, all we need is the well-known fact that membership and emptiness are NL-complete and the recent result that given an OCA $\mathcal{A}$, one can compute in polynomial time an NFA $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A}){\downarrow}$ [3].

Reversal-bounded counter automata. Intuitively, an *r-reversal-bounded k-counter automaton* [19] (short (k,r)-RBCA) is a nondeterministic finite automaton with k counters that can store natural numbers. For each counter, it has operations *increment*, *decrement*, and *zero test*. Moreover, a computation is only valid if each counter *reverses* at most r times. Here, a computation *reverses* a counter c if on c, it first executes a sequence of increments and then a decrement command or vice versa. See [19] for details.

Instead of working directly with RBCA, we will work here with the model of *blind counter automata* [9]. It is not as well-known as RBCA, but simpler and directly amenable to linear algebraic methods. A *blind k-counter automaton* is a tuple $\mathcal{A} = (Q, X, q_0, \Delta, Q_f)$, where Q, X, q_0, and Q_f are defined as in NFAs, but Δ is a finite subset of $Q \times (X \cup \{\varepsilon\}) \times \{-1,0,1\}^k \times Q$. A *walk* is a word $\delta_1 \cdots \delta_m \in \Delta^*$ where $\delta_i = (p_i, x_i, d_i, p'_i)$ for $i \in [1,m]$ and $p'_j = p_{j+1}$ for $j \in [1, m-1]$. The *effect* of the walk is $d_1 + \cdots + d_m$. Its *input* is $x_1 \cdots x_m \in X^*$. If the walk has effect 0 and $p_0 = q_0$ and $p_m \in Q_f$, then the walk is *accepting*. The *language accepted by* $\mathcal{A}$ is the set of all inputs of accepting walks.

Using blind counter automata is justified because to each (k,r)-RBCA, one can construct in logarithmic space a language-equivalent $(kr,1)$-RBCA [5], which is essentially a blind kr-counter automaton. On the other hand, every blind k-counter automaton can be turned in logarithmic space into a $(k+1,1)$-RBCA [20]. Hence, decision problems about (k,r)-RBCA for fixed k and r correspond to problems about blind k-counter automata for fixed k.

In the following, by a *model*, we mean a way of specifying a language. In order to succinctly refer to the different decision problems, we use symbols for the models above. By Ideal, NFA, OCA, $\mathsf{RBC}_{k,r}$, RBC, CFG, we mean ideals, finite automata, OCA, RBCA with a fixed number of counters and reversals, general RBCA, and context-free grammars, respectively. Then, for $\mathcal{M}, \mathcal{N} \in \{\mathsf{Ideal}, \mathsf{NFA}, \mathsf{OCA}, \mathsf{RBC}_{k,r}, \mathsf{RBC}, \mathsf{CFG}\}$, we consider the following problems. In the *downward closure inclusion problem* $\mathcal{M} \subseteq_\downarrow \mathcal{N}$, we are given a language K in $\mathcal{M}$ and a language L in $\mathcal{N}$ and are asked whether $K{\downarrow} \subseteq L{\downarrow}$. For the *downward closure equivalence problem* $\mathcal{M} =_\downarrow \mathcal{N}$, the input is the same, but we are asked whether $K{\downarrow} = L{\downarrow}$.

Results. The complexity results for the inclusion problem are summarized in Table 1. For the equivalence problem, we will see that every hardness result for $\mathcal{M} \subseteq_\downarrow \mathcal{N}$ also holds for $\mathcal{M} =_\downarrow \mathcal{N}$. Since for non-ideal models, the appearing complexity classes are pairwise comparable, this implies that the complexity for $\mathcal{M} =_\downarrow \mathcal{N}$ is then the harder of the two classes for $\mathcal{M} \subseteq_\downarrow \mathcal{N}$ and $\mathcal{N} \subseteq_\downarrow \mathcal{M}$. For example, the problem $\mathsf{NFA} =_\downarrow \mathsf{RBC}$ is Π_2^P-complete and for fixed alphabets, $\mathsf{RBC}_{k,r} =_\downarrow \mathsf{CFG}$ is coNP-complete.

3 Ideals and Witnesses

Our algorithms for inclusion use three types of witnesses. The first type is a slight variation of a result of [4]. The latter authors were interested in equivalence problems, which caused their bound to depend on both input languages. The proof is essentially the same.

▶ **Proposition 1** (Short witness). *If $\mathcal{A}$ is an NFA and $K{\downarrow} \not\subseteq L(\mathcal{A}){\downarrow}$, then there exists a $w \in K{\downarrow} \setminus L(\mathcal{A}){\downarrow}$ with $|w| \leq |\mathcal{A}| + 1$.*

The other types of witnesses strongly rely on ideals, which requires some notation. An ideal is a product $I = Y_0^*\{x_1, \varepsilon\}Y_1^* \cdots \{x_n, \varepsilon\}Y_n^*$ where the Y_i are alphabets and the x_i are letters. Its *length* $|I|_\mathsf{I}$ is the smallest n such that I can be written in this form. Since every downward closed language can be written as a finite union of ideals, we can extend this definition to languages: $|L|_\mathsf{I}$ is the smallest n such that $L{\downarrow}$ is a union of ideals of length $\leq n$.

Sometimes, it will be convenient to work with a different length measure of ideals. An *ideal expression (of length n)* is a product $L_1 \cdots L_n$, where each L_i is of the form Y^* or $\{x, \varepsilon\}$, where Y is an alphabet and x is a letter. Note that $Y^* = Y^*\{x, \varepsilon\}$ if $x \in Y$ and $\{x, \varepsilon\} = \emptyset^*\{x, \varepsilon\}$. Therefore, an ideal expression of length n defines an ideal of length $\leq n$. In analogy to $|\cdot|_{\mathsf{I}}$, for a language L, we define its *expression length* $|L|_{\mathsf{E}}$ to be the smallest n such that $L{\downarrow}$ can be written as a finite union of ideal expressions of length $\leq n$. The expression length has the advantage of being subadditive: For languages K, L we have $|KL|_{\mathsf{E}} \leq |K|_{\mathsf{E}} + |L|_{\mathsf{E}}$. Moreover, we have $|L|_{\mathsf{I}} \leq |L|_{\mathsf{E}} \leq 2|L|_{\mathsf{I}} + 1$.

The measure $|\cdot|_{\mathsf{I}}$ turns out to be instrumental for the inclusion problem. Note that $K{\downarrow} \not\subseteq L{\downarrow}$ if and only if there is an ideal $I \subseteq K{\downarrow}$ of length $\leq |K|_{\mathsf{I}}$ with $I \not\subseteq L{\downarrow}$. We can therefore guess ideals and check inclusion for them. From now on, we assume alphabets to come linearly ordered. This means for every alphabet Y, there is a canonical word w_Y in which every letter from Y occurs exactly once.

▶ **Proposition 2** (Ideal witness). *Let $I = Y_0^*\{x_1, \varepsilon\}Y_1^* \cdots \{x_n, \varepsilon\}Y_n^*$. Then the following are equivalent:* (i) *$I \subseteq L{\downarrow}$.* (ii) *$w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_n w_{Y_n}^m \in L{\downarrow}$ for every $m \geq |L|_{\mathsf{I}} + 1$.* (iii) *$w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_n w_{Y_n}^m \in L{\downarrow}$ for some $m \geq |L|_{\mathsf{I}} + 1$.*

A word of the form $w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_n w_{Y_n}^m \in L{\downarrow}$ with $m \geq |L|_{\mathsf{I}} + 1$ is therefore called an *ideal witness for I and L*. The proof of Proposition 2 is a simple pumping argument based on the fact that an ideal of length $\leq m$ admits an NFA with $\leq m + 1$ states. Ideal witnesses are useful when we have a small bound on $|K|_{\mathsf{I}}$ and $|L|_{\mathsf{I}}$ but only a large bound on the NFA size of $L{\downarrow}$. Observe that putting a bound on $|L|_{\mathsf{I}}$ amounts to proving a pumping lemma: We have $|L|_{\mathsf{I}} \leq n$ if and only if for every $w \in L$, there is an ideal I with $|I|_{\mathsf{I}} \leq n$ and $x \in I \subseteq L{\downarrow}$.

However even if, say, $|K|_{\mathsf{I}}$ is polynomial and $|L|_{\mathsf{I}}$ is exponential, ideal witnesses can be stored succinctly in polynomial space, by keeping a binary representation of the power m. For instance, this will be used in the case $\mathsf{NFA} \subseteq_{\downarrow} \mathsf{RBC}$.

Sometimes, we have a small bound on $|L|_{\mathsf{I}}$, but $|K|_{\mathsf{I}}$ may be large. Then, ideal witnesses are too large to achieve an optimal algorithm. In these situations, we can guarantee smaller witnesses if we fix the alphabet.

▶ **Proposition 3** (Small alphabet witness). *Let $K, L \subseteq X^*$. If $K{\downarrow} \not\subseteq L{\downarrow}$, then there exists a $w \in K{\downarrow} \setminus L{\downarrow}$ with $|w| \leq |X| \cdot (|L|_{\mathsf{I}} + 1)^{|X|}$.*

The proof of Proposition 3 is more involved than Propositions 2 and 1. Note that a naive bound can be obtained by intersecting exponentially (in $|L|_{\mathsf{I}}$) many automata for the ideals of $L{\downarrow}$ and complementing the result. This would yield a doubly exponential (in $|L|_{\mathsf{I}}$) bound, even considering the fact that ideals have linear-size DFAs. We can, however, use the latter fact in a different way.

A DFA is *ordered* if its states can be partially ordered so that for every transition $p \xrightarrow{x} q$, we have $p \leq q$. In other words, the automaton is acyclic except for loop transitions. The following lemma is easy to see: In order to check membership in an ideal, one just has to keep a pointer into the expression that never moves left.

▶ **Lemma 4.** *Given an ideal representation of length n, one can construct in logarithmic space an equivalent ordered DFA with $n + 2$ states.*

An ordered DFA *cycles* at a position of an input word if that position is read using a loop. The following lemma is the key idea behind Proposition 3. Together with Lemma 4, it clearly implies Proposition 3. For unary alphabets, it is easy to see. We use induction on $|X|$ and show, roughly speaking, that without such a position, no strict subalphabet can be

used for too long. Then, all letters have to appear often, meaning a state has to repeat after seeing the whole alphabet. Hence, the automaton stays in this state until the end.

▶ **Lemma 5.** *If $w \in X^*$ with $|w| > |X| \cdot (n-1)^{|X|}$, then w has a position at which every ordered n-state DFA cycles.*

4 Insertion trees

In Section 5, we will show upper bounds for the size of downward closure NFAs and for ideal lengths for counter automata. These results employ certain decompositions of NFA runs into trees, which we discuss here. Let $\mathcal{A} = (Q, X, \Delta, q_0, Q_f)$ be a finite automaton. A *walk* is a word $w = \delta_1 \cdots \delta_m \in \Delta^*$ where $\delta_i = (p_i, x_i, p'_i)$ for $i \in [1, m]$ and $p'_j = p_{j+1}$ for $j \in [1, m-1]$. The walk is a *(p_1-)cycle* if $p_1 = p'_m$. In this case, we define $\sigma(w) := p_1$. A cycle is *prime* if $p_i = p_1$ implies $i = 1$. A cycle is *simple* if $p_i = p_j$ implies $i = j$. A state q *occurs* on the cycle if $p_i = q$ for some i. If $i \neq 1$, then q occurs *properly*.

A common operation in automata theory is to take a run and delete cycles until the run has length at most $|Q|$. The idea behind an insertion tree is to record where we deleted which cycles. This naturally leads to a tree.

For our purposes, trees are finite, unranked and ordered. An *insertion tree* is a tree $t = (V, E)$ together with a map $\gamma\colon V \to \Delta^*$ that assigns to each vertex $v \in V$ a simple cycle $\gamma(v)$ such that if u is the parent of v, then $\sigma(\gamma(v))$ properly occurs in $\gamma(u)$. Note that we allow multiple children for a state that occurs in $\gamma(u)$.

Since t is ordered and in every simple cycle there is at most one proper occurrence of each state, an insertion tree defines a unique (typically not simple) cycle $\alpha(t)$. Formally, if t is a single vertex v, then $\alpha(t) := \gamma(v)$. If t consists of a root r and subtrees $t_1, \ldots, t_s$, then $\alpha(t)$ is obtained by inserting each $\alpha(t_i)$ in $\gamma(r)$ at the (unique) occurrence of $\sigma(\alpha(t_i))$. The *height* of an insertion tree is the height of its tree.

▶ **Lemma 6.** *Every prime cycle of $\mathcal{A}$ admits an insertion tree of height at most $|Q|$.*

The idea is to pick a cycle c strictly contained in the prime cycle, but of maximal length. Then, after removing c, no state occurs both before and after the old position of c. This forces any insertion tree t of the remainder to place this position in the root. We then apply induction to the subtrees of t and to c. The resulting trees can then all be attached to the root, increasing the height by at most one.

One application of Lemma 6 is to construct short ideals in a pumping lemma for counter automata. Part of this construction is independent from counters, so we stay with NFAs for a moment. Suppose we have an insertion tree $t = (V, E)$ with map $\gamma\colon V \to \Delta^*$ and a subset $F \subseteq V$, whose members we call *fixed vertices* or *fixed cycles*. Those in $V \setminus F$ are called *pumpable vertices/cycles*.

We use fixed and pumpable vertices to guide a pumping process as follows. A sequence $s = t_1 \cdots t_m$ of insertion trees is called *compatible* if $\sigma(\alpha(t_1)) = \cdots = \sigma(\alpha(t_m))$. We assume that we have a global set F of vertices that designates the fixed vertices for all these trees. Suppose v is a pumpable vertex. We obtain new compatible sequences in two ways:

- Let $v_1, \ldots, v_\ell$ be the children of v. We choose $i \in [0, \ell]$ and split up v at i, meaning that we create a new vertex v' with $\gamma(v') = \gamma(v)$ to the right of v and move $v_{i+1}, \ldots, v_\ell$ (and, of course, their subtrees) to v'.
- If the whole subtree under v is pumpable (we call such subtrees *pumpable*), then we can duplicate this subtree and attach its root somewhere as a sibling of v.

If v is a root, these operations mean that we introduce a new tree in the sequence. If a compatible sequence s' is obtained from s by repeatedly performing these operations, we say that s' is obtained by *pumping* s. This allows us to define the following language:

$$P(t_1 \cdots t_m, F) = \{\iota(\alpha(t'_1) \cdots \alpha(t'_k)) \mid t'_1 \cdots t'_k \text{ results from pumping } t_1 \cdots t_m\}.$$

Here, for a walk w, $\iota(w)$ denotes the input word read by w. The following lemma will yield the desired short ideals.

▶ **Lemma 7.** *Let $s = t_1 \cdots t_m$ be a compatible sequence of insertion trees of height $\leq h$ and let F be a set of fixed vertices. Then, the language $P(s, F)\downarrow$ is an ideal that satisfies $|P(s, F)\downarrow|_{\mathsf{E}} \leq h|F|(2|Q| + |F|)^2$.*

Roughly speaking, the pumping process is designed so that pumpable subtrees only cause alphabets Y in factors Y^* of the ideal to grow and thus do not affect the ideal length. Hence, the only vertices that contribute to the length are those that are ancestors of vertices in F. Since the trees have height $\leq h$, there are at most $h|F|$ such ancestors.

5 Counter Automata

In this section, we construct downward closure NFAs for counter automata and prove upper bounds for ideal lengths. Mere computability of downward closures of blind counter automata can be deduced from computability for Petri net languages [14]. However, that necessarily results in non-primitive recursive automata (see Section 8). As a special case of stacked counter automata, blind counter automata were provided with a new construction method in [35]. That algorithm, however, yields automata of non-elementary size. Here, we prove an exponential bound.

▶ **Theorem 8.** *For each n-state blind k-counter automaton $\mathcal{A}$, there is an NFA $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A})\downarrow$ and $|\mathcal{B}| \leq (3n)^{5nk+7k^3}$. Moreover, $\mathcal{B}$ can be computed in exponential time.*

Linear Diophantine equations. In order to show correctness of our construction, we employ a result of Pottier [29], which bounds the norm of minimal non-negative solutions to a linear Diophantine equation. Let $A \in \mathbb{Z}^{k\times m}$ be an integer matrix. We write $\|A\|_{1,\infty}$ for $\sup_{i\in[1,k]}(\sum_{j\in[1,m]} |a_{ij}|)$, where a_{ij} is the entry of A at row i and column j. A solution $x \in \mathbb{N}^m$ to the equation $Ax = 0$ is *minimal* if there is no $y \in \mathbb{N}^m$ with $Ay = 0$ and $y \leq x$, $y \neq x$. The set of all solutions clearly forms a submonoid of $\mathbb{N}^m$, which is denoted M. The set of minimal solutions is denoted $\mathcal{H}(M)$ and called the *Hilbert basis* of M. Let r be the rank of A. Pottier showed the following.

▶ **Theorem 9** (Pottier [29])**.** *For each $x \in \mathcal{H}(M)$, $\|x\|_1 \leq (1 + \|A\|_{1,\infty})^r$.*

By applying Theorem 9 to the matrix $(A| - b)$, it is easy to deduce that for each $x \in \mathbb{N}^m$ with $Ax = b$, there is a $y \in \mathbb{N}^m$ with $Ay = b$, $y \leq x$, and $\|y\|_1 \leq (1 + \|(A| - b)\|_{1,\infty})^{r+1}$.

Automata for the downward closure. Let $\mathcal{A}$ be a blind k-counter automaton with n states. The idea of the construction of $\mathcal{B}$ is to traverse insertion trees of prime cycles of $\mathcal{A}$. Although insertion trees were introduced for finite automata, they also apply to blind counter automata if we regard the counter updates as input symbols. $\mathcal{B}$ keeps track of where it is in the tree using a stack of bounded height. The stack alphabet will be $\Gamma = Q \times [-n, n]^k$. We define $B = n + n \cdot (3n)^{(k+1)^2}$. The state set of our automaton $\mathcal{B}_1$ is the following:

$$Q_1 = Q \times \Gamma^{\leq n} \times [-B, B]^k \times \mathbb{P}([-n, n]^k) \times \mathbb{P}([-n, n]^k).$$

Here, the number of states is clearly doubly exponential, but we shall make the automaton smaller in two later steps. The idea behind $\mathcal{B}_1$ is that counter values in the interval $[-B, B]$ are simulated precisely (in the factor $[-B, B]^k$). Roughly speaking, whenever we encounter a cycle, we can decide whether to (i) add its effect to this precise counter or to (ii) remember the effect as "must be added at least once". We call the former *precise cycles*; the latter are dubbed *obligation cycles* and are stored in the first factor $\mathbb{P}([-n, n]^k)$. In either case, the effect of a cycle is kept as "repeatable" in the second factor $\mathbb{P}([-n, n]^k)$.

In order to be able to guess for each cycle whether it should be a precise cycle or an obligation cycle, we traverse an insertion tree of (the prime cycles on) a walk of $\mathcal{A}$. On the stack (the factor $\Gamma^{\leq n}$), we keep the cycles that we have started to traverse. Suppose we are executing a cycle in a vertex v and the path from the root to v consists of the vertices $v_1, \ldots, v_m$. Let $\gamma(v_i)$ be a q_i-cycle for $i \in [1, m]$. Then, the stack content is $(q_1, u_1) \cdots (q_m, u_m)$, where u_i is the effect of the part of $\gamma(v_i)$ that has already been traversed.

In the end, we verify that (i) the precise counter is zero and (ii) one can add up obligation cycles (each of them at least once) and repeatable cycles to zero. The latter condition is captured in the following notion. Let $S, T \subseteq \mathbb{Z}^k$ be finite sets with $S = \{u_1, \ldots, u_s\}$, $T = \{v_1, \ldots, v_t\}$. We call the pair (S, T) *cancellable* if there are $x_1, \ldots, x_s \in \mathbb{N} \setminus \{0\}$ and $y_1, \ldots, y_t \in \mathbb{N}$ with $\sum_{i=1}^{s} x_i u_i + \sum_{i=1}^{t} y_i v_i = 0$. In particular, $(\emptyset, T)$ is cancellable for any finite $T \subseteq \mathbb{Z}^k$. Together, (i) and (ii) guarantee that the accepted word is in the downward closure: They imply that we could have executed all of the obligation cycles and some others (again) to fulfill our obligation. Hence, there is a run of $\mathcal{A}$ accepting a superword.

The number of cycles we can use as precise cycles is limited by the capacity B of our precise counter. We shall apply Theorem 9 to show that there is always a choice of cycles to use as precise cycles so as to reach zero in the end and not exceed the capacity.

The first type of transition in $\mathcal{B}_1$ is the following. For each transition $(p, a, d, q) \in \Delta$ and state $(p, \varepsilon, v, S, T) \in Q_1$ such that $v + d \in [-B, B]^k$, we have a transition

$$(p, \varepsilon, v, S, T) \xrightarrow{a} (q, \varepsilon, v + d, S, T). \tag{1}$$

These allow us to simulate transitions in a walk of $\mathcal{A}$ that are not part of a cycle. We can guess that a cycle is starting. If we are in state p, then we push $(p, 0)$ onto the stack:

$$(p, w, v, S, T) \xrightarrow{\varepsilon} (p, w(p, 0), v, S, T). \tag{2}$$

While we are traversing a cycle, new counter effects are stored in the topmost stack entry. For each $(p, a, d, q) \in \Delta$ and $(p, w(r, u), v, S, T) \in Q_1$ with $u + d \in [-n, n]^k$, we have:

$$(p, w(r, u), v, S, T) \xrightarrow{a} (q, w(r, u + d), v, S, T). \tag{3}$$

When we are at the end of a cycle, we have to decide whether it should be a precise cycle or an obligation cycle. The following transition means it should be precise: The counter effect u of the cycle is added to the counter v, the stack is popped, and u is added to the set of repeatable effects T. For each $(p, w(p, u), v, S, T) \in Q_1$ with $v + u \in [-B, B]^k$, we have:

$$(p, w(p, u), v, S, T) \xrightarrow{\varepsilon} (p, w, v + u, S, T \cup \{u\}). \tag{4}$$

In order to designate the cycle as an obligation cycle, we have the following transition: The stack is popped and u is added to both S and T. For each state $(p, w(p, u), v, S, T) \in Q_1$, we include the transition

$$(p, w(p, u), v, S, T) \xrightarrow{\varepsilon} (p, w, v, S \cup \{u\}, T \cup \{u\}) \tag{5}$$

The initial state is $(q_0, \varepsilon, 0, \emptyset, \emptyset)$ and the final states are all those of the form $(q, \varepsilon, 0, S, T)$ where q is final in $\mathcal{A}$ and (S, T) is cancellable. Employing Lemma 6 and Theorem 9, one can now show that $L(\mathcal{A}) \subseteq L(\mathcal{B}_1) \subseteq L(\mathcal{A})\downarrow$.

State space reduction I. We have thus shown that $L(\mathcal{B}_1)\downarrow = L(\mathcal{A})\downarrow$. However, $\mathcal{B}_1$ has a doubly exponential number of states. Therefore, we now reduce the number of states in two steps. First, instead of remembering the set S of obligation effects, we only maintain a linearly independent set of vectors generating the same vector space. For a set $R \subseteq \mathbb{Q}^k$, let $\mathrm{span}(R)$ denote the $\mathbb{Q}$-vector space generated by R. Moreover, $\mathbb{I}(R)$ denotes the set of linearly independent subsets of R. Our new automaton $\mathcal{B}_2$ has states

$$Q_2 = Q \times \Gamma^{\leq n} \times [-B, B]^k \times \mathbb{I}([-n, n]^k) \times \mathbb{P}([-n, n]^k)$$

and a state in $\mathcal{B}_2$ is final if it is final in $\mathcal{B}_1$. $\mathcal{B}_2$ has the same transitions as $\mathcal{B}_1$, except that aside from those of type (5), it has

$$(p, w(p, u), v, S, T) \xrightarrow{\varepsilon} (p, w, v, S', T \cup \{u\}) \tag{6}$$

for each linearly independent subset $S' \subseteq S \cup \{u\}$ such that $\mathrm{span}(S') = \mathrm{span}(S \cup \{u\})$. Of course, such an S' exists for any S and u. This means, by induction on the length, for any walk of $\mathcal{B}_1$ from (p, w, v, S, T) to (q, w', v', S', T'), we can find a walk with the same input in $\mathcal{B}_2$ from (p, w, v, S, T) to (q, w', v', S'', T') with $S'' \subseteq S'$ and $\mathrm{span}(S'') = \mathrm{span}(S')$. Since (S', T') is cancellable and $S' \subseteq T'$, the pair (S'', T') is cancellable as well. This means, our walk in $\mathcal{B}_2$ is accepting and hence $L(\mathcal{B}_1) \subseteq L(\mathcal{B}_2)$. It remains to verify that $L(\mathcal{B}_2) \subseteq L(\mathcal{B}_1)$.

Observe that for any walk arriving in (q, w, v, S, T) in $\mathcal{B}_2$, there is a corresponding walk in $\mathcal{B}_1$ arriving in (q, w, v, S', T) for some $S' \supseteq S$ with $\mathrm{span}(S') = \mathrm{span}(S)$. The next lemma tells us that if (q, w, v, S, T) is a final state in $\mathcal{B}_2$, then (q, w, v, S', T) is final in $\mathcal{B}_1$. This implies that $L(\mathcal{B}_2) \subseteq L(\mathcal{B}_1)$ and hence $L(\mathcal{B}_2) = L(\mathcal{B}_1)$.

▶ **Lemma 10.** *Let $T \subseteq \mathbb{Z}^k$ and $S_1 \subseteq S_2 \subseteq \mathbb{Z}^k$ such that $\mathrm{span}(S_1) = \mathrm{span}(S_2)$. If (S_1, T) is cancellable, then so is (S_2, T).*

State space reduction II. We apply a similar transformation to the last factor of the state space. In $\mathcal{B}_3$, we have the state space

$$Q_3 = Q \times \Gamma^{\leq n} \times [-B, B]^k \times \mathbb{I}([-n, n]^k) \times \mathbb{I}([-n, n]^k).$$

and a state is final in $\mathcal{B}_3$ if and only if it is final in $\mathcal{B}_2$. Analogous to $\mathcal{B}_2$, we change the transitions so that instead of adding $u \in [-n, n]^k$ to T, we store an arbitrary $T' \in \mathbb{I}(T \cup \{u\})$.

This time, it is clear that $L(\mathcal{B}_3) \subseteq L(\mathcal{B}_2)$: For every walk in $\mathcal{B}_3$ arriving at (q, w, v, S, T), there is a corresponding walk in $\mathcal{B}_2$ arriving at (q, w, v, S, T') such that $T \subseteq T'$. Clearly, if (S, T) is cancellable, then (S, T') must be cancellable as well. The following lemma implies $L(\mathcal{B}_2) \subseteq L(\mathcal{B}_3)$: It says that for each walk in $\mathcal{B}_2$ arriving at (q, w, v, S, T), there is a corresponding walk in $\mathcal{B}_3$ arriving at (q, w, v, S, T') for some linearly independent $T' \subseteq T$ such that (S, T') is cancellable and hence (q, w, v, S, T') is final.

▶ **Lemma 11.** *Let $S, T \subseteq \mathbb{Z}^k$ such that (S, T) is cancellable. Then there is a linearly independent subset $T' \subseteq T$ such that (S, T') is cancellable.*

We have thus shown that $L(\mathcal{B}_3)\downarrow = L(\mathcal{A})\downarrow$. An estimation of the size of Q_3 now completes the proof of Theorem 8. We apply Theorem 8 to derive an algorithm for $\mathsf{Ideal} \subseteq_\downarrow \mathsf{RBC}$.

▶ **Corollary 12.** *The problem* Ideal $\subseteq_\downarrow$ RBC *is in* NP.

Since Theorem 8 provides an exponential bound on $|L(\mathcal{A})|_{\mathsf{I}}$, we can use an ideal witness $w = w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_\ell w_{Y_\ell}^m$ (Proposition 2) for which we have to check membership in $L(\mathcal{A})$. Since ℓ is polynomial and m exponential, we can compute a compressed representation of w in form of a *straight-line program*, a context-free grammar that generates one word [25]. It follows easily from work of Hague and Lin [16] that membership of such compressed words in languages of blind (or reversal-bounded) counter automata is decidable in NP.

Fixed number of counters. Unfortunately, the size bound for the NFAs provided by Theorem 8 has the number of states in the exponent, meaning that if we fix the number k of counters, we still have an exponential bound. In fact, we leave open whether one can construct polynomial-size NFAs for fixed k. However, in many cases it suffices to have a polynomial bound on the length of ideals.

▶ **Theorem 13.** *If $\mathcal{A}$ is an n-state blind k-counter automaton, then $|L(\mathcal{A})|_{\mathsf{I}} \leq (5n)^{7(k+1)^2}$.*

Recall that an upper bound on $|L|_{\mathsf{I}}$ is essentially a pumping lemma (see Section 3). Here, the idea is to take a walk of $\mathcal{A}$ and delete cycles until the remaining walk u is at most n steps. For the deleted cycles, we take an insertion tree of height at most n (Lemma 6). Then, using Theorem 9, we pick a subset F (whose size is polynomial when fixing k) of cycles that can balance out the effect of u. We then employ Lemma 7 to the insertion trees to construct an ideal whose length is polynomial in $|F|$.

6 Context-Free Grammars

We turn to context-free grammars. First, we mention that given a context-free grammar $\mathcal{G}$, one can construct in exponential time an (exponential-size) NFA accepting $L(\mathcal{A})\downarrow$ [4, 7, 11, 33, 27]. Second, we provide an algorithm for the problem Ideal $\subseteq_\downarrow$ CFG.

▶ **Theorem 14.** *The problem* Ideal $\subseteq_\downarrow$ CFG *is in* P.

In [34], this problem has been reduced to the *simultaneous unboundedness problem (SUP)* for context-free languages. The latter asks, given a language $L \subseteq a_1^* \cdots a_n^*$, whether we have $L\downarrow = a_1^* \cdots a_n^*$. Moreover, this reduction is clearly polynomial. Hence, we assume that $L(\mathcal{G}) \subseteq a_1^* \cdots a_n^*$ and that the grammar $\mathcal{G} = (N, T, P, S)$ is productive and in Chomsky normal form, meaning that productions are of the form $A \to BC$, $A \to a_i$, or $A \to \varepsilon$ for $A, B, C \in N$. First, we add productions $A \to \varepsilon$ for all $A \in N$, so that the resulting grammar $\mathcal{G}'$ satisfies $L(\mathcal{G}') = L(\mathcal{G})\downarrow$. For each $A \in N$, we can in polynomial time construct a CFG for $\{w \in (N \cup T)^* \mid A \Rightarrow_{\mathcal{G}'}^* w\}$, so we can compute the sets $L_i = \{A \in N \mid A \Rightarrow_{\mathcal{G}'}^* a_i A\}$ and $R_i = \{A \in N \mid A \Rightarrow_{\mathcal{G}'}^* A a_i\}$ using membership queries. We can thus compute the grammar $\mathcal{G}^\omega$, which results from $\mathcal{G}'$ by (i) removing all productions $A \to a_i$, (ii) adding $A \to a_i^\omega A$ for each $A \in L_i$ and (iii) adding $A \to A a_i^\omega$ for each $A \in R_i$. Clearly, an occurrence of a_i^ω certifies the ability to generate an unbounded number of a_i's. Thus, if $a_1^\omega \cdots a_n^\omega \in L(\mathcal{G}^\omega)$, then $a_1^* \cdots a_n^* \subseteq L(\mathcal{G}') = L(\mathcal{G})\downarrow$. It is not hard to see that the converse is true as well. We have thus reduced the SUP to the membership problem.

7 Algorithms

Algorithms for $\mathcal{M} \subseteq_\downarrow$ Ideal. Suppose $\mathcal{M} =$ Ideal and we want to decide whether $I \subseteq J$ for ideals $I, J \subseteq X^*$. In logspace, we construct an ideal witness w for I and J (Proposition 2)

and a DFA $\mathcal{A}$ for $X^* \setminus J$ (Proposition 4) and check whether $w \in L(\mathcal{A})$. In all other cases, to decide $L{\downarrow} \subseteq I$, we construct a DFA $\mathcal{A}$ for $X^* \setminus I$ and check whether $L{\downarrow} \cap L(\mathcal{A}) = \emptyset$.

Algorithms for $\mathcal{M} \subseteq_{\downarrow}$ NFA. Suppose $\mathcal{M} = \mathsf{Ideal}$ and we want to decide whether $I \subseteq L(\mathcal{A}){\downarrow}$ for an NFA $\mathcal{A}$. Since $|L(\mathcal{A})|_{\mathsf{I}} \leq |\mathcal{A}|$, we can construct in logspace an ideal witness w for I and $L(\mathcal{A}){\downarrow}$ and verify $w \in L(\mathcal{A}){\downarrow}$. In all other cases, we use a short witness for coNP-membership.

Algorithms for $\mathcal{M} \subseteq_{\downarrow}$ OCA. Suppose $\mathcal{M} = \mathsf{Ideal}$ and we want to decide whether $I \subseteq L(\mathcal{A}){\downarrow}$ for an OCA $\mathcal{A}$. We have a polynomial bound on $|L(\mathcal{A})|_{\mathsf{I}}$ (see Section 2). Hence, we construct in logspace an ideal witness w for I and $L(\mathcal{A}){\downarrow}$. We can also construct in logspace an OCA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}){\downarrow}$. Membership for OCA is in $\mathsf{NL} = \mathsf{coNL}$, so we can verify $w \in I$ and $w \notin L(\mathcal{A}') = L(\mathcal{A}){\downarrow}$. In all other cases, we convert the OCA to an NFA (see Section 2).

Algorithms for $\mathcal{M} \subseteq_{\downarrow} \mathsf{RBC}_{k,r}$. Let $\mathcal{A}$ be drawn from $\mathsf{RBC}_{k,r}$. First, suppose $\mathcal{M} = \mathsf{Ideal}$ and we want to decide whether $I \subseteq L(\mathcal{A})$. By Theorem 13, we have a polynomial bound on $|L(\mathcal{A})|_{\mathsf{I}}$ and can construct in logspace an ideal witness w for I and $L(\mathcal{A})$. We can also construct in logspace an RBCA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}){\downarrow}$. Since membership for $\mathsf{RBC}_{k,r}$ is in NL [12], we can check whether $w \in L(\mathcal{A}')$. Now let $\mathcal{M} \in \{\mathsf{NFA}, \mathsf{OCA}, \mathsf{RBC}_{k,r}\}$ and we are given L in $\mathcal{M}$ and an automaton $\mathcal{A}$ from $\mathsf{RBC}_{k,r}$. For NFA, OCA, and $\mathsf{RBC}_{k,r}$, we have a polynomial bound on $|L|_{\mathsf{I}}$ (see Section 2 and Theorem 13). Thus, we guess an ideal I of polynomial length and then verify that $I \subseteq L{\downarrow}$ but $I \not\subseteq L(\mathcal{A}){\downarrow}$. Since $\mathsf{Ideal} \subseteq_{\downarrow} \mathcal{M}$ and $\mathsf{Ideal} \subseteq_{\downarrow} \mathsf{RBC}_{k,r}$ are in NL, the verification is done in NL. Hence, non-inclusion is in NP. For $\mathcal{M} \in \{\mathsf{CFG}, \mathsf{RBC}\}$, we assume a fixed alphabet. Let L be in $\mathcal{M}$. Then Proposition 3 and Theorem 13 provide us with a witness of polynomial length. Since (non-)membership in $L{\downarrow}$ and in $L(\mathcal{A}){\downarrow}$ can be decided in NP, non-inclusion is in NP.

Algorithms for $\mathcal{M} \subseteq_{\downarrow}$ CFG. The case $\mathsf{Ideal} \subseteq_{\downarrow} \mathsf{CFG}$ is shown in Theorem 14. Suppose $\mathcal{M} \in \{\mathsf{NFA}, \mathsf{OCA}, \mathsf{RBC}_{k,r}\}$ and we are given L in $\mathcal{M}$ and a CFG $\mathcal{G}$. We have a polynomial bound on $|L|_{\mathsf{I}}$ (see Section 2 and Theorem 13), so that we can guess a polynomial-length ideal I. Since $\mathsf{Ideal} \subseteq_{\downarrow} \mathcal{M}$ is in NL in every case and $\mathsf{Ideal} \subseteq_{\downarrow} \mathsf{CFG}$ is in P, we can verify in polynomial time that $I \subseteq L{\downarrow}$ and $I \not\subseteq L(\mathcal{G}){\downarrow}$. Thus, non-inclusion is in NP. In the case $\mathcal{M} \in \{\mathsf{RBC}, \mathsf{CFG}\}$, we construct exponential-size downward closure NFAs and check inclusion for them (and the latter problem is in coNP). This yields a coNEXP algorithm.

Algorithms for $\mathcal{M} \subseteq_{\downarrow}$ RBC. Let $\mathcal{A}$ be from RBC. The ideal case is treated in Corollary 12. When given L in $\mathcal{M} \in \{\mathsf{NFA}, \mathsf{OCA}, \mathsf{RBC}_{k,r}\}$, we guess a polynomial length ideal I and verify that $I \subseteq L{\downarrow}$ in NL. Since $\mathsf{Ideal} \subseteq_{\downarrow} \mathsf{RBC}$ is in NP, we can also check in coNP that $I \not\subseteq L(\mathcal{A}){\downarrow}$. Hence, non-inclusion is in Σ_2^{P}. For $\mathcal{M} \in \{\mathsf{CFG}, \mathsf{RBC}\}$, we proceed as for $\mathcal{M} \subseteq_{\downarrow} \mathsf{CFG}$.

8 Hardness

In this section, we prove hardness results. Most of them are deduced from a generic hardness theorem that, under mild assumptions, derives hardness from the ability to generate finite sets with long words. We will work with bounds that exhibit the following useful property. A monotone function $f\colon \mathbb{N} \to \mathbb{N}$ will be called *amplifying* if $f(n) \geq n$ for $n \geq 0$ and there is a polynomial p such that $f(p(n)) \geq f(n)^2$ for large enough $n \in \mathbb{N}$. We say that a model *has property* $\Delta(f)$ (or short: *is* $\Delta(f)$) if for each given $n \in \mathbb{N}$, one can construct in polynomial time a description of a finite language whose longest word has length $f(n)$. For the sake of simplicity, we will abuse notation slightly and write $\Delta(f(n))$ instead of $\Delta(f)$. For a function $t\colon \mathbb{N} \to \mathbb{N}$, we use $\mathsf{coNTIME}(t)$ to denote the complements of languages accepted by nondeterministic Turing machines that are time bounded by $O(t(n^c))$ for some constant c.

We also need two mild language theoretic properties. A *transducer* is a finite automaton where every edge reads input and produces output. For a transducer $\mathcal{T}$ and a language L, the language $\mathcal{T}L$ consists of all words output by the transducer while reading a word from L. We call a model $\mathcal{M}$ a *full trio model* if given a transducer $\mathcal{T}$ and a language L described with $\mathcal{M}$, one can compute in polynomial time a description of $\mathcal{T}L$. A *substitution* is a map $\sigma\colon X \to \mathbb{P}(Y^*)$ that replaces each letter by a language. For languages L, we define $\sigma(L)$ in the obvious way. We call σ *simple* if $X \subseteq Y$ and there is some $x \in X$ such that for all $x' \in X \setminus \{x\}$, we have $\sigma(x') = \{x'\}$ and x occurs in each word from L at most once. We say that $\mathcal{M}$ has *closure under simple substitutions* if given a description of L and of $\sigma(x)$ in $\mathcal{M}$, we can compute in polynomial time a description of $\sigma(L)$.

▶ **Theorem 15.** *Let $t\colon \mathbb{N} \to \mathbb{N}$ be amplifying and let $\mathcal{M}$ and $\mathcal{N}$ be full trio models that are $\Delta(t)$ and have closure under simple substitutions. Then both $\mathcal{M} \subseteq_\downarrow \mathcal{N}$ and $\mathcal{M} =_\downarrow \mathcal{N}$ are hard for* coNTIME(t)*. Moreover, this hardness already holds for binary alphabets.*

Since NFAs are $\Delta(n)$, Theorem 15 yields coNP-hardness for inclusion and equivalence. In [4], hardness of equivalence was shown directly. RBCA and CFG clearly exhibit closure under simple substitutions and can generate exponentially long words. This yields:

▶ **Corollary 16.** *For $\mathcal{M}, \mathcal{N} \in \{CFG, RBC\}$, $\mathcal{M} \subseteq_\downarrow \mathcal{N}$ and $\mathcal{M} =_\downarrow \mathcal{N}$ are* coNEXP*-hard.*

From Theorem 15, we can also deduce hardness for other models. It was shown by Habermehl, Meyer, and Wimmel [14] that downward closures or Petri net languages are computable, which implies decidability of our problems. We use Theorem 15 to prove an Ackermann lower bound. Let $A_n\colon \mathbb{N} \to \mathbb{N}$ be defined as $A_0(x) = x + 1$, $A_{n+1}(0) = A_n(1)$, and $A_{n+1}(x+1) = A_n(A_{n+1}(x))$. Then, the function $A\colon \mathbb{N} \to \mathbb{N}$ with $A(n) = A_n(n)$ is the *Ackermann function*. Of course, for large enough n, we have $A_n(x) \geq x^2$. For such n, we have $A(n+1) = A_n(A_{n+1}(n)) \geq A_{n+1}(n)^2 \geq A(n)^2$, so A is amplifying. A result of Mayr and Meyer [26] (see also [30]) states that given $n \in \mathbb{N}$, one can construct in polynomial time a Petri net that, from its initial marking, can produce up to $A(n)$ tokens in an output place. Hence, Petri nets are $\Delta(A)$ and they clearly satisfy the language-theoretic conditions.

▶ **Corollary 17.** *For Petri net languages, inclusion and equivalence of downward closures is Ackermann-hard.*

Building on the sufficient condition of [34], Hague, Kochems, and Ong [15] have shown that downward closures are computable for higher-order pushdown automata. However, the method of [34] does not yield any information about the complexity of this computation. For $k \in \mathbb{N}$, we denote by $\exp_k$ the function with $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$. It is easy to see that order-k pushdown automata are $\Delta(\exp_k)$ (for instance, one can adapt Example 2.5 of [8]). By co-k-NEXP, we denote the complements of languages accepted by nondeterministic Turing machines in time $O(\exp_k(n^c))$ for some constant c.

▶ **Corollary 18.** *For higher-order pushdown automata of order k, inclusion and equivalence of downward closures is hard for* co-k-NEXP*.*

Our last hardness result could also be shown using the method of Theorem 15. However, it is simpler to reduce a variant of the subset sum problem [6].

▶ **Proposition 19.** NFA $\subseteq_\downarrow$ RBC *and* NFA $=_\downarrow$ RBC *are Π_2^{P}-hard, even for binary alphabets.*

We have thus shown hardness for all inclusion problems that do not involve ideals. The remaining cases inherit hardness from the emptiness problem (for $\mathcal{M} \subseteq_\downarrow$ Ideal) or the non-emptiness problem (Ideal $\subseteq_\downarrow \mathcal{M}$).

References

1. P. A. Abdulla, L. Boasson, and A. Bouajjani. Effective lossy queue languages. In *ICALP 2001*, 2001.
2. P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.
3. M. F. Atig, D. Chistikov, P. Hofman, K. N. Kumar, P. Saivasan, and G. Zetzsche. The complexity of regular abstractions of one-counter languages. To appear in LICS 2016.
4. G. Bachmeier, M. Luttenberger, and M. Schlund. Finite automata for the sub- and superword closure of cfls: Descriptional and computational complexity. In *LATA 2015*, 2015.
5. B. S. Baker and R. V. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Sciences*, 8(3):315–332, 1974.
6. P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. In *CPM 1997*, 1997.
7. B. Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.
8. W. Damm and A. Goerdt. An automata-theoretic characterization of the OI-hierarchy. In *ICALP 1982*, 1982.
9. S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.
10. H. Gruber, M. Holzer, and M. Kutrib. The size of Higman-Haines sets. *Theoretical Computer Science*, 387(2):167–176, 2007.
11. H. Gruber, M. Holzer, and M. Kutrib. More on the size of higman-haines sets: effective constructions. *Fundamenta Informaticae*, 91(1):105–121, 2009.
12. E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22(2):220–229, 1981.
13. C. Haase and P. Hofman. Tightening the complexity of equivalence problems for commutative grammars. In *STACS 2016*, 2016.
14. P. Habermehl, R. Meyer, and H. Wimmel. The downward-closure of Petri net languages. In *ICALP 2010*, 2010.
15. M. Hague, J. Kochems, and C.-H. L. Ong. Unboundedness and downward closures of higher-order pushdown automata. In *POPL 2016*, 2016.
16. M. Hague and A. W. Lin. Model checking recursive programs with numeric data types. In *CAV 2011*, 2011.
17. L. H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969.
18. D. T. Huynh. The complexity of equivalence problems for commutative grammars. *Information and Control*, 66(1):103–121, 1985.
19. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM (JACM)*, 25(1):116–133, 1978.
20. M. Jantzen and A. Kurganskyy. Refining the hierarchy of blind multicounter languages and twist-closed trios. *Information and Computation*, 185(2):159–181, 2003.
21. P. Jullien. *Contribution à létude des types d'ordres dispersés*. PhD thesis, Université de Marseille, 1969.
22. P. Karandikar, M. Niewerth, and Ph. Schnoebelen. On the state complexity of closures and interiors of regular languages with subwords and superwords. *Theoretical Computer Science*, 610, Part A:91–107, 2016.
23. E. Kopczyński. Complexity of problems of commutative grammars. *Logical Methods in Computer Science*, 11(1), 2015.

24 E. Kopczyński and A. W. To. Parikh images of grammars: Complexity and applications. In *LICS 2010*, 2010.

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

26 E. W. Mayr and A. R. Meyer. The complexity of the finite containment problem for petri nets. *Journal of the ACM*, 28(3):561–576, 1981.

27 A. Okhotin. On the state complexity of scattered substrings and superstrings. *Fundamenta Informaticae*, 99(3):325–338, 2010.

28 R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

29 L. Pottier. Minimal solutions of linear diophantine systems : bounds and algorithms. In *RTA 1991*, 1991.

30 L. Priese and H. Wimmel. *Petri-Netze*. Springer-Verlag, 2003.

31 N. Rampersad, J. Shallit, and Z. Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informaticae*, 116(1-4):223–236, 2012.

32 S. La Torre, A. Muscholl, and I. Walukiewicz. Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable. In *CONCUR 2015*, 2015.

33 J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978.

34 G. Zetzsche. An approach to computing downward closures. In *ICALP 2015*, 2015.

35 G. Zetzsche. Computing downward closures for stacked counter automata. In *STACS 2015*, 2015.

36 G. Zetzsche. The complexity of downward closure comparisons, 2016. URL: `http://arxiv.org/abs/1605.03149`.