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.

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 [33].

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. On the other hand, recent work of La Torre, Muscholl, and Walukiewicz [25] 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 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, little is known about the complexity of deciding whether the downward closure of one language includes or equals that of another. To the knowledge of the author, the only result in this direction is that (i) the equivalence problem for downward closures of two given NFAs is coNP-complete and (ii) this problem is in coNEXP for context-free grammars. The former was shown by Bachmeier, Luttenberger, and Schlund [4] and

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

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

the latter has not been mentioned explicitly, but follows from existing downward closure constructions [4, 7, 10, 26, 29] and from the coNP upper bound for NFAs. Previous work on downward closures of infinite-state systems has mainly focused on mere computability [1, 2, 7, 14, 15, 26, 33, 34] or on descriptional complexity [10, 11, 22, 29]. This work studies the complexity of the inclusion and the equivalence problem of downward closures for 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 and the downward closure equivalence problem among finite automata, one-counter automata, context-free grammars, and reversal-bounded counter automata (either with a fixed number of counters and reversals or without).

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 [30]. 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 their Parikh image is effectively semilinear, which implies decidability of the inclusion and equivalence problem for Parikh images. The equivalence problem has also been investigated for Parikh images. This line of research 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.

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-

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

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 $\mathsf{Ideal} \subseteq_\downarrow \mathsf{Ideal}$, all entries indicate completeness. A † means that the entry refers to the fixed alphabet case (for at least two letters).

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 [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, 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 *k-blind 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 $\Delta \subseteq Q \times (X \cup \{\varepsilon\}) \times \mathbb{Z}^k \times Q$. 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* $\mathcal{M} \subseteq_\downarrow \mathcal{N}$, where 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 3.1** (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 it can be written in this form. Since every doward 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 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 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 $K{\downarrow} \subseteq L{\downarrow}$ 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 3.2** (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 3.2 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 3.3** (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 3.3 is more involved than propositions 3.1 and 3.2. 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-sized DFAs. We can, however, use the latter fact in a different way.

A DFA is *ordered* if its states can be partially ordered such 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 3.4.** *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 insight behind proposition 3.3. Together with lemma 3.4, it clearly implies proposition 3.3. 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 3.5.** *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 Tree decompositions

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* in $\mathcal{A}$ is a word $\delta_1 \cdots \delta_m \in \Delta^*$ 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 a tree decomposition is to record where we deleted which cycles. This naturally leads to a tree.

For our purposes, trees are finite, unranked and ordered. A *tree decomposition* 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, a tree decomposition defines a unique (typically not simple) prime cycle $\alpha(t)$. Formally, if t is a single vertex v, then $\alpha(t) := \alpha(v)$. If t consists of a root r and subtrees $t_1, \dots, t_s$, 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 a tree decomposition is the height of its tree.

▶ **Lemma 4.1.** *Every prime cycle of $\mathcal{A}$ admits a tree decomposition of height at most $|Q|$.*

The idea is to pick a cycle c so large that after removing c, no state occurs both before and after the old position of c. This forces any tree decomposition 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 1.

One application of lemma 4.1 is to construct short ideals for a pumping lemma. Suppose we have a tree decomposition $t = (V, E)$ 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*. The ideals associated to tree decompositions do not depend on counter operations. Therefore, for now, we only consider tree decompositions for finite automata.

We use fixed and pumpable vertices to guide a pumping process as follows. A sequence $s = t_1 \cdots t_m$ of tree decompositions 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 $v_1, \dots, v_\ell$ 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 $v_{i+1}, \dots, v_\ell$ (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 4.2.** *Let $s = t_1 \cdots t_m$ be a compatible sequence of tree decompositions of height $\leq h$ and let F be a set of fixed vertices. Then, the language $P(s, F)\!\downarrow$ is an ideal with $|P(s, F)\!\downarrow|_{\mathsf{E}} \leq h|F|(2|Q| + |F|)^2$.*

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 the 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 7). As a special case of stacked counter automata, blind counter automata were provided with a new construction method in [34]. The algorithm, however, yields automata of non-elementary size. Here, we prove an exponential bound.

▶ **Theorem 5.1.** *For each blind k-counter automaton $\mathcal{A}$ with n states, there is a finite automaton $\mathcal{B}$ 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 [31], which bounds the norm of minimal non-negative solutions to a linear diophantine equation. Let $A \in \mathbb{Z}^{k\times\ell}$ be an integer matrix. We write $\|A\|_{1,\infty}$ for $\sup_{i\in[1,k]}(\sum_{j\in[1,\ell]} |a_{ij}|)$, where a_{ij} is the entry of A at row i and column j. A solution $x \in \mathbb{N}^k$ to the equation $Ax = 0$ is *minimal* if there is no $y \in \mathbb{N}^k$ with $Ay = 0$ and $y \leq x$, $y \neq x$. The set of all solutions clearly forms a submonoid of $\mathbb{N}^k$, 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 5.2** (Pottier [31])**.** *For each $x \in \mathcal{H}(M)$, $\|x\|_1 \leq (1 + \|A\|_{1,\infty})^r$.*

By applying theorem 5.2 to the matrix $(A|-b)$, it is easy to deduce that for each $x \in \mathbb{N}^k$ with $Ax = b$, there is a $y \in \mathbb{N}^k$ with $Ay = b$, $y \leq x$, and $\|y\|_1 \leq (1 + \|(A|-b)\|_{1,\infty})^{r+1}$.

Automata for the downward closure The idea of the construction of $\mathcal{B}$ is to traverse tree decompositions of prime cycles of $\mathcal{A}$. Although tree decompositions 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 $\Gamma = Q \times [-n, n]^k$. Let $n = |\mathcal{A}|$ and define $B = n \cdot (3n)^{(k+1)^2}$. 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 choose for each cycle whether it should be a precise cycle or an obligation cycle, we traverse a tree decomposition 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 5.2 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 transition $(p, a, d, q) \in \Delta$ and state $(p, w(r, u), v, S, T) \in Q_1$ such that $u + d \in [-n, n]^k$, we have a transition

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

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 state $(p, w(p, u), v, S, T) \in Q_1$ such thay $v + u \in [-B, B]^k$, we have a transition

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

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 4.1 and theorem 5.2, one can now show that $L(\mathcal{B}_1)$ has the same downward closure as $L(\mathcal{A})$.

▶ **Proposition 5.3.** $L(\mathcal{A}) \subseteq L(\mathcal{B}_1) \subseteq L(\mathcal{A})\downarrow$.

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 imlies that $L(\mathcal{B}_2) \subseteq L(\mathcal{B}_1)$ and hence $L(\mathcal{B}_2) = L(\mathcal{B}_1)$.

▶ **Lemma 5.4.** *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 $L(\mathcal{B}_3) = L(\mathcal{B}_2)$: It means given a 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 5.5.** *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 5.1. See the appendix for details. We apply theorem 5.1 to derive an algorithm for $\mathsf{Ideal} \subseteq_\downarrow \mathsf{RBC}$.

▶ **Corollary 5.6.** *The problem $\mathsf{Ideal} \subseteq_\downarrow \mathsf{RBC}$ is in NP.*

Since theorem 5.1 provides an exponential bound on $|L(\mathcal{A})|_{\mathsf{I}}$, there is an ideal witness $w = w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_\ell w_{Y_\ell}^m$ 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 [27]. 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 5.1 has the number of states in the exponent, meaning that if we fix the number k of counters, the 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 5.7.** *Suppose $\mathcal{A}$ is a blind k-counter automaton with n states. Then we have $|L(\mathcal{A})|_{\mathsf{I}} \leq (6n)^{4(k+1)^2}$.*

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 a tree decomposition of height at most n (lemma 4.1). Then, using theorem 5.2, 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 4.2 to the tree decompositions to construct an ideal whose length is polynomial in F.

6 Context-Free Grammars

In this section, we present results specific to context-free grammars. First, we mention that given a context-free grammar $\mathcal{G}$, one can construct in exponential time an NFA accepting $L(\mathcal{A})\downarrow$ [4, 7, 10, 26, 29]. Second, we provide an algorithm for the problem $\mathsf{Ideal} \subseteq_{\downarrow} \mathsf{CFG}$.

▶ **Theorem 6.1.** *The problem* $\mathsf{Ideal} \subseteq_{\downarrow} \mathsf{CFG}$ *is in* P.

As shown in [33], the problem can be reduced in polynomial time to the *simultaneous unboundedness problem (SUP)*. The latter asks, given a language $L \subseteq a_1^* \cdots a_n^*$, whether $L\downarrow = a_1^* \cdots a_n^*$. Hence, we assume that $L(\mathcal{G}) \subseteq a_1^* \cdots a_n^*$ and that $\mathcal{G} = (N, T, P, S)$ is in Chomsky normal form and productive, 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 $L(\mathcal{G}') = L(\mathcal{G})\downarrow$. Since for each $A \in N$, we can in polynomial time construct a CFG for $\{w \in (N \cup T)^* \mid A \Rightarrow_{\mathcal{G}'}^* w\}$, 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 regular intersection and testing for emptiness. We can therefore 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) $A \to A a_i^\omega$ for each $A \in R_i$. Clearly, the idea is that 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 $a_1^* \cdots a_n^* \subseteq L(\mathcal{G}') = L(\mathcal{G})\downarrow$. The following lemma tells us that the converse is true as well, so that we can decide the SUP by invoking the membership problem for CFG, which is in P.

▶ **Lemma 6.2.** *We have $a_1^\omega \cdots a_n^\omega \in L(G^\omega)$ if and only if $a_1^* \cdots a_n^* \subseteq L(G)\downarrow$.*

7 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 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 in $\mathcal{M}$, we can compute in polynomial time a description of $\sigma(L)$.

▶ **Theorem 7.1.** *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 $\mathsf{coNTIME}(t)$. Moreover, this hardness already holds for binary alphabets.*

Since NFAs are $\Delta(n)$, theorem 7.1 yields coNP-hardness for inclusion and equivalence. In [4], hardness of equivalence was shown directly. The next corollary follows because RBCA and CFG clearly exhibit closure under simple substitutions and it is easy to generate exponentially long words.

▶ **Corollary 7.2.** *For $\mathcal{M}, \mathcal{N} \in \{\mathsf{CFG}, \mathsf{RBC}\}$, $\mathcal{M} \subseteq_\downarrow \mathcal{N}$ and $\mathcal{M} =_\downarrow \mathcal{N}$ are coNEXP-hard.*

From theorem 7.1, 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 7.1 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 [28] (see also [32]) 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 7.3.** *For Petri net languages, inclusion and equivalence of downward closures is Ackermann-hard.*

Building on the sufficient condition of [33], Hague, Kochems, and Ong [15] have shown that downward closures are computable for higher-order pushdown automata. However, the method of [33] 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 7.4.** *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 7.1. However, it is simpler to reduce a variant of the subset sum problem [6].

▶ **Proposition 7.5.** $\mathsf{NFA} \subseteq_\downarrow \mathsf{RBC}$ *and* $\mathsf{NFA} =_\downarrow \mathsf{RBC}$ *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 $\mathcal{M} \subseteq_\downarrow \mathsf{Ideal}$) or the non-emptiness problem ($\mathsf{Ideal} \subseteq_\downarrow \mathcal{M}$).

8 Algorithms

Algorithms for $\mathcal{M} \subseteq_\downarrow \mathsf{Ideal}$. Suppose $\mathcal{M} = \mathsf{Ideal}$ and we want to decide whether $I \subseteq J$ for ideals I, J. In logspace, we can construct an ideal witness w for I and J (proposition 3.2) and a DFA $\mathcal{A}$ for the complement of J (lemma 3.4) and check whether $\mathcal{A}$ accepts w. In all other cases, to decide $L{\downarrow} \subseteq I$, we construct a DFA $\mathcal{A}$ for the complement of I and check whether $L{\downarrow} \cap L(\mathcal{A}) = \emptyset$.

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

Algorithms for $\mathcal{M} \subseteq_\downarrow \mathsf{OCA}$. Suppose $\mathcal{M} = \mathsf{Ideal}$ and we want to decide whether $I \subseteq L(\mathcal{A})$ 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 $L(\mathcal{A})$. We can also construct in logspace an OCA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}){\downarrow}$. Since membership (and hence the non-membership) for OCA is in NL, we can verify that $w \in I$ 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 5.7, 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 RBC $\mathcal{A}'$ 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 5.7). 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 3.3 provides 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 $\mathcal{M} \subseteq_\downarrow \mathsf{CFG}$. The case $\mathsf{Ideal} \subseteq_\downarrow \mathsf{CFG}$ is shown in theorem 6.1. 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 5.7), 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.

Algorithms for $\mathcal{M} \subseteq_\downarrow \mathsf{RBC}$. Let $\mathcal{A}$ be from RBC. The ideal case is treated in corollary 5.6. If we are given L in $\mathcal{M} \in \{\mathsf{NFA}, \mathsf{OCA}, \mathsf{RBC}_{k,r}\}$, we can guess a polynomial length ideal I and then 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}. Finally, for $\mathcal{M} \in \{\mathsf{CFG}, \mathsf{RBC}\}$, we construct downward closure NFAs and check inclusion for them. This yields a coNEXP algorithm.

References

[1] P. A. Abdulla, L. Boasson, and A. Bouajjani. "Effective Lossy Queue Languages." In: *ICALP 2001*.

[2] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. "Using Forward Reachability Analysis for Verification of Lossy Channel Systems." In: *Formal Methods in System Design* 25.1 (2004), pp. 39–65.

[3] M. F. Atig, D. Chistikov, P. Hofman, K. N. Kumar, P. Saivasan, and G. Zetzsche. *Complexity of regular abstractions of one-counter languages*. 2016. arXiv: 1602.03419.

[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*.

[5] B. S. Baker and R. V. Book. "Reversal-bounded multipushdown machines." In: *Journal of Computer and System Sciences* 8.3 (1974), pp. 315–332.

[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*.

[7] B. Courcelle. "On constructing obstruction sets of words." In: *Bulletin of the EATCS* 44 (1991), pp. 178–186.

[8] W. Damm and A. Goerdt. "An automata-theoretic characterization of the OI-hierarchy." In: *ICALP 1982*.

[9] S. A. Greibach. "Remarks on blind and partially blind one-way multicounter machines." In: *Theoretical Computer Science* 7.3 (1978), pp. 311–324.

[10] H. Gruber, M. Holzer, and M. Kutrib. "More on the size of Higman-Haines sets: effective constructions." In: *Fundamenta Informaticae* 91.1 (2009), pp. 105–121.

[11] H. Gruber, M. Holzer, and M. Kutrib. "The size of Higman-Haines sets." In: *Theoretical Computer Science* 387.2 (2007), pp. 167–176.

[12] E. M. Gurari and O. H. Ibarra. "The complexity of decision problems for finite-turn multicounter machines." In: *Journal of Computer and System Sciences* 22.2 (1981), pp. 220–229.

[13] C. Haase and P. Hofman. "Tightening the Complexity of Equivalence Problems for Commutative Grammars." In: *STACS 2016*.

[14] P. Habermehl, R. Meyer, and H. Wimmel. "The Downward-Closure of Petri Net Languages." In: *ICALP 2010*.

[15] M. Hague, J. Kochems, and C.-H. L. Ong. "Unboundedness and Downward Closures of Higher-order Pushdown Automata." In: *POPL 2016*.

[16] M. Hague and A. W. Lin. "Model Checking Recursive Programs with Numeric Data Types." In: *CAV 2011*.

[17] L. H. Haines. "On free monoids partially ordered by embedding." In: *Journal of Combinatorial Theory* 6.1 (1969), pp. 94–98.

[18] D. T. Huynh. "The complexity of equivalence problems for commutative grammars." In: *Information and Control* 66.1 (1985), pp. 103–121.

[19] O. H. Ibarra. "Reversal-bounded multicounter machines and their decision problems." In: *Journal of the ACM (JACM)* 25.1 (1978), pp. 116–133.

[20] M. Jantzen and A. Kurganskyy. "Refining the hierarchy of blind multicounter languages and twist-closed trios." In: *Information and Computation* 185.2 (2003), pp. 159–181.

[21] P. Jullien. "Contribution à létude des types d'ordres dispersés." PhD thesis. Université de Marseille, 1969.

[22] P. Karandikar, M. Niewerth, and P. Schnoebelen. "On the state complexity of closures and interiors of regular languages with subwords and superwords." In: *Theoretical Computer Science* 610, Part A (2016), pp. 91–107.
[23] E. Kopczyński. "Complexity of Problems of Commutative Grammars." In: *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*.
[25] S. La Torre, A. Muscholl, and I. Walukiewicz. "Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable." In: *CONCUR 2015*.
[26] J. van Leeuwen. "Effective constructions in well-partially-ordered free monoids." In: *Discrete Mathematics* 21.3 (1978), pp. 237–252.
[27] M. Lohrey. "Algorithmics on SLP-compressed strings: a survey." In: *Groups Complexity Cryptology* 4.2 (2012), pp. 241–299.
[28] E. W. Mayr and A. R. Meyer. "The complexity of the finite containment problem for Petri nets." In: *Journal of the ACM* 28.3 (1981), pp. 561–576.
[29] A. Okhotin. "On the state complexity of scattered substrings and superstrings." In: *Fundamenta Informaticae* 99.3 (2010), pp. 325–338.
[30] R. J. Parikh. "On Context-Free Languages." In: *Journal of the ACM* 13.4 (1966), pp. 570–581.
[31] L. Pottier. "Minimal solutions of linear diophantine systems : bounds and algorithms." In: *RTA 1991*.
[32] L. Priese and H. Wimmel. *Petri-Netze*. Springer-Verlag, 2003.
[33] G. Zetzsche. "An Approach to Computing Downward Closures." In: *ICALP 2015*.
[34] G. Zetzsche. "Computing Downward Closures for Stacked Counter Automata." In: *STACS 2015*.

A Ideals and Witnesses

Proof of proposition 3.1. Let $\mathcal{A} = (Q, X, \Delta, q_0, Q_f)$. Consider the DFA $\mathcal{B} = (\mathbb{P}(Q), X, \Delta', Q, Q'_f)$, where from a state $P \subseteq Q$ on input $x \in X$, we enter the state P', consisting of all $q' \in Q$ that are reachable from a state in P via a path on which x occurs. Moreover, Q'_f is the set of all $P \subseteq Q$ with $P \cap Q_f = \emptyset$. Then clearly $\mathcal{B}$ accepts $X^* \setminus L(\mathcal{A})\downarrow$.

Choose $w \in K\downarrow \setminus L(\mathcal{A})\downarrow$ of minimal length and write $w = w_1 \cdots w_m$ for letters $w_1, \ldots, w_m$. Suppose $m > |\mathcal{A}| + 1$ and consider the run of w in $\mathcal{B}$. For each $i \in [0, m]$, let $P_i \subseteq Q$ be the state entered after reading $w_1 \cdots w_i$. Then we have $P_0 \supseteq P_1 \supseteq \cdots$ and since $m > |Q| + 1$, there are $i < j$ with $P_i = P_j$. Yet this means that also $w' = w_1 \cdots w_i w_{j+1} \cdots w_m$ is a member of $L(\mathcal{B}) = X^* \setminus L(\mathcal{A})\downarrow$. Moreover, we have $w' \preceq w$ and thus $w' \in K\downarrow$. This contradicts our choice of w. ◀

Proof of proposition 3.2. The implications "(i)⇒(ii)" and "(ii)⇒(iii)" are trivial, so assume (iii). Write $L\downarrow = \bigcup_{i=1}^{m} I_i$ as a union of ideals of length $\leq |L|_{\mathsf{I}}$. Then we have $w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_n w_{Y_n}^m \in I_i$ for some i. Since I_i has length at most $|L|_{\mathsf{I}}$, there is an NFA $\mathcal{A}$ with at most $|L|_{\mathsf{I}} + 1$ states for I_i. However, we have $m \geq |L|_{\mathsf{I}} + 1$, so in the computation of the NFA for $w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_n w_{Y_n}^m$, for each $i \in [0, n]$, some power $w_{Y_i}^{k_i}$, $k_i > 0$, has to lie on a cycle of $\mathcal{A}$. We can therefore pump each of these cycles, which implies $I \subseteq L\downarrow$. ◀

Proof of lemma 3.4. Let $I = Y_0^*\{x_1, \varepsilon\}Y_1^* \cdots \{x_n, \varepsilon\}Y_n^*$ with. For $i \in [0, n]$ and $a \in X$, let

$$J_{i,a} = \{j \in [i, n] \mid a \in Y_i^*\{x_{i+1}, \varepsilon\}Y_{i+1}^* \cdots \{x_j, \varepsilon\}Y_j^*\}$$

Our DFA has states $Q = \{0, \dots, n+1\}$ and for $i \in Q$, we have $i \xrightarrow{a} j$ if and only if

$$j = \begin{cases} \min J_{i,a} & \text{if } J_{i,a} \neq \emptyset \\ m+1 & \text{if } J_{i,a} = \emptyset \end{cases}$$

Moreover, 0 is the initial state and the states $0, \dots, n$ are final. Clearly, the automaton is ordered, has $n+2$ states, and can be constructed in logarithmic space. In order to show the correctness, we define the ideal $I_k = Y_0^*\{x_1, \varepsilon\}Y_1^* \cdots \{x_k, \varepsilon\}Y_k^*$ for each $k \in [0, n]$. Observe that $I_0 \subseteq I_1 \subseteq \cdots \subseteq I_n = I$. By induction on the length of w, it is easy to see that if $0 \xrightarrow{w} j$, then

- if $j \in [0, n]$, then j is the smallest number with $w \in I_j$.
- if $j = n+1$, then $w \notin I$.

In particular, the automaton accepts I. ◀

Proof of lemma 3.5. To make our induction work, we define $f_n \colon \mathbb{N} \to \mathbb{N}$ by $f_n(1) = n-1$ and $f_n(k) = (f_n(k-1)+1) \cdot (n-1)$. We claim that if $w > f_n(|X|)$ for $w \in X^*$, then w has a position at which every ordered n-state DFA cycles.

We proceed by induction on $|X|$. If $X = \{a\}$, then it suffices to consider $w = a^n$. Consider an ordered n-state DFA $\mathcal{A}$ and let $q_0, q_1, \dots, q_n$ be the states occupied while reading w. Then there are $i < j$ with $q_i = q_j$ and since $\mathcal{A}$ is ordered, we have $q_i = q_{i+1}$. This means, q_i has an a-labeled loop and therefore $q_i = q_{i+1} = q_{i+2} = \cdots = q_n$. In particular, $\mathcal{A}$ cycles at the last position of w.

Now suppose $k = |X| > 1$ and $|w| > f_n(k) = (f_n(k-1)+1)(n-1)$. For every word $v \in X^*$, let $\alpha(v) \in X^*$ be the shortest prefix of v in which every letter from X occurs. If v does not contain every letter from X, then we define $\alpha(v) = v$. We factorize w as $p_1 \cdots p_m$ by applying α to w, then applying α to the rest of the word, and so on. Formally, we set $r_0 = w$, $p_i = \alpha(r_{i-1})$, and define r_i so that $r_{i-1} = p_i r_i$. For some smallest $m \geq 1$, we have $p_m = r_m$. Then clearly $w = p_1 \cdots p_m$ and every p_i is non-empty.

For each $i \in [1, m]$, let p_i' be obtained from p_i by removing its last position. By the choice of p_i, the word p_i' contains at most $k-1$ distinct letters. Hence, if $|p_i'| > f_n(k-1)$ for some $i \in [1, m]$, then p_i' contains a position at which every ordered n-state DFA cycles. In particular, w contains such a position (because every computation on w contains some computation on p_i'). Therefore, we may assume that $|p_i| = |p_i'| + 1 \leq f_n(k-1) + 1$ for every $i \in [1, m]$.

If we had $m \leq n-1$, this would imply $|w| = |p_1 \cdots p_m| \leq (f_n(k-1)+1)(n-1)$, which is not the case. Hence, we have $m \geq n$. Now consider an ordered n-state DFA $\mathcal{A}$ with its computation

$$q_0 \xrightarrow{p_1} q_1 \xrightarrow{p_2} \cdots \xrightarrow{p_m} q_m.$$

Since $m+1 > n$, there are $i < j$ with $q_i = q_j$ and since $\mathcal{A}$ is ordered, we have $q_i = q_{i+1} = \cdots = q_j$. We distinguish two cases.

- If $j = m$, then our computation cycles at every position in p_m.
- If $j < m$, then p_j contains every letter from X at least once. This means q_i has an a-loop for every $a \in X$. Therefore, $q_i = q_{i+1} = \cdots = q_m$. In particular, our computation cycles on every position in p_m.

Thus, we have shown that any ordered n-state DFA cycles on every position in p_m, which proves our claim.

From the definition of f_n, it follows easily by induction that $f_n(k) = \sum_{i=1}^{k} (n-1)^i$ and hence $f_n(k) \leq k \cdot (n-1)^k$. ◀

B Tree decompositions

Proof of lemma 4.1. Let $w \in \Delta^*$ be a prime q-cycle and let $P_w \subseteq Q$ be the set of states occurring properly in w. We show by induction on $|P_w|$ that every prime cycle w admits a tree decomposition of height at most $|P_w|$.

If no state from P_w repeats in w, then w is simple and the statement is trivial. For each $p \in P_w$ that does repeat in w, let $\lambda(p)$ be the length of the longest p-cycle that is a factor of w. Among all states from P_w that repeat in w, we choose p such that $\lambda(p)$ is maximal. Then $w = xyz$ where y is a p-cycle of length $\lambda(p)$. Observe that by the maximality of p, there is no state that occurs properly both in x and in z.

We write $y = y_1 \cdots y_r$ such that each y_i is a prime p-cycle. Then since p does not occur properly in y_i, each y_i admits a tree decomposition t_i of height $|P_w| - 1$.

Consider any tree decomposition t of xz. Observe that since there is no state that occurs properly both in x and in z, the only cycle in t where p can occur is t's root. Therefore, if $s_1, \ldots, s_k$ are the subtrees of t immediately below the root, then no $\alpha(s_i)$ contains p. We can therefore factorize each $\alpha(s_i)$ into prime cycles that each have a tree decomposition of height at most $|P_w| - 1$. Thus, by replacing in t each s_i by this sequence of trees, we obtain a tree decomposition t' of xz of height at most $|P_w|$.

Since p occurs in the root of t' and this is the only occurrence of p in t', we can attach the trees t_i directly below the root of t' to obtain a tree decomposition t'' of w. Moreover, since each t_i has height at at most $|P_w| - 1$, t'' has height at most $|P_w|$. ◀

Proof of lemma 4.2. If $F = \emptyset$, then we can duplicate every tree in the sequence, leading to $P(s, F)\downarrow = Y^*$, where Y is the set of letters occurring anywhere on a tree in s. Hence, $P(s, F)\downarrow$ is an ideal of length 1. Thus, we assume $F \neq \emptyset$.

As the first step, we consider the case where s consists of one tree t. Let A be the set of vertices in t that are ancestors of vertices in F. We show by induction on h that $P(t, F)\downarrow$ is an ideal and $|P(t, F)|_{\mathsf{E}} \leq |A| \cdot (2|Q| + |F|)$.

Let r be the root of t and $\gamma(r) = e_1 \cdots e_\ell$, where $e_1, \ldots, e_\ell \in \Delta$. Let C be the set of children of r that are in A. Moreover, let $e_i = (q_{i-1}, a_i, q_i)$ for $i \in [1, \ell - 1]$. Consider the subtrees 'inserted after e_i': In other words, those subtrees directly below r whose root node is assigned a q_i-cycle by γ. Some of them contain a fixed vertex; let $s_{i,1}, \ldots, s_{i,k_i}$ be those subtrees. The other subtrees inserted after e_i are pumpable; let Y_i be the set of input letters occuring them. Let $F_{i,j} \subseteq F$ be the set of fixed nodes in $s_{i,j}$ and let $A_{i,j}$ be the set of ancestors of fixed vertices in $s_{i,j}$. By induction, we have

$$|P(s_{i,j}, F_{i,j})\downarrow|_{\mathsf{E}} \leq |A_{i,j}| \cdot (2|Q| + |F_{i,j}|) \leq |A_{i,j}| \cdot (2|Q| + |F|). \tag{7}$$

- Suppose $r \in F$. Then we have $P(t, F)\downarrow = \{a_1, \varepsilon\} I_1 \{a_2, \varepsilon\} \cdots I_{\ell-1} \{a_\ell, \varepsilon\}$, where

  $$I_i = Y_i^* (P(s_{i,1}, F_{i,1})\downarrow) Y_i^* \cdots (P(s_{i,k_i}, F_{i,k_i})\downarrow) Y_i^*$$

  for $i \in [1, \ell - 1]$. Hence, $P(t, F)\downarrow$ is an ideal. Let us estimate the expression length. Note that (7) yields

  $$|I_i|_{\mathsf{E}} \leq k_i + 1 + \sum_{j=1}^{k_i} |P(s_{i,j}, F_{i,j})\downarrow|_{\mathsf{E}} \leq k_i + 1 + (2|Q| + |F|) \cdot \sum_{j=1}^{k_i} |A_{i,j}|$$

and therefore

$$\begin{aligned}|P(t,F)\downarrow|_{\mathsf{E}} &\le \ell + \sum_{i=1}^{\ell-1} |I_i| \le \ell + \sum_{i=1}^{\ell-1} (k_i+1) + (2|Q|+|F|) \sum_{i=1}^{\ell-1} |A_{i,j}| \\ &\le 2\ell + \underbrace{\sum_{i=1}^{\ell-1} k_i}_{\le |F|} \quad + \quad (2|Q|+|F|) \underbrace{\sum_{i=1}^{\ell-1} |A_{i,j}|}_{\le |A|-1} \\ &\le |A| \cdot (2|Q|+|F|).\end{aligned}$$

- Suppose $r \notin F$. Then we have $P(t,F)\downarrow = I_1 \cdots I_{\ell-1}$, where

  $$I_i = Z_i^* (P(s_{i,1}, F_{i,1})\downarrow) Z_i^* \cdots (P(s_{i,k_i}, F_{i,k_i})\downarrow) Z_i^*,$$

  for $i \in [1, \ell-1]$ with $Z_i = Y_i \cup \{a_1, \ldots, a_\ell\}$. Hence, $P(t,F)\downarrow$ is an ideal. Let us estimate the expression length. As before, (7) yields

  $$|I_i|_{\mathsf{E}} \le k_i + 1 + \sum_{i=1}^{k_i} |P(s_{i,j}, F_{i,j})\downarrow|_{\mathsf{E}} \le k_i + 1 + (2|Q|+|F|) \cdot \sum_{j=1}^{k_i} |A_{i,j}|$$

  and therefore

  $$\begin{aligned}|P(t,F)\downarrow|_{\mathsf{E}} &\le \sum_{i=1}^{\ell-1} (k_i+1) + (2|Q|+|F|) \sum_{i=1}^{\ell-1} |A_{i,j}| \\ &\le \ell + \sum_{i=1}^{\ell-1} k_i + (2|Q|+|F|) \sum_{i=1}^{\ell-1} |A_{i,j}| \le |A| \cdot (2|Q|+|F|).\end{aligned}$$

This concludes our first step. Note that since t has height $\le h$, every vertex in F has at most h ancestors, so that $|A| \le h \cdot |F|$. This means, our first step implies that in the case of a single tree t, we have $|P(t,F)\downarrow|_{\mathsf{E}} \le h|F| \cdot (2|Q|+|F|)$.

Let us now consider $P(s,F)\downarrow$ where $s = t_1 \cdots t_m$ is a compatible sequence. Of the trees $t_1, \ldots, t_m$, let $t'_1, \ldots, t'_\ell$ be those which contain a fixed vertex. The other trees in the sequence $t_1, \ldots, t_m$ are pumpable and we define Y to be the set of letters occurring in those pumpable trees. Note that $\ell \le |F|$.

According to our first step, we have $|P(t'_i, F)\downarrow|_{\mathsf{E}} \le h|F| \cdot (2|Q|+|F|)$ for each $i \in [1, \ell]$. Moreover, we have

$$P(s,F)\downarrow = Y^* P(t'_1, F)\downarrow Y^* \cdots P(t'_\ell, F)\downarrow Y^*,$$

which means $P(s,F)\downarrow$ is an ideal and we may estimate

$$\begin{aligned}|P(s,F)\downarrow|_{\mathsf{E}} &\le (\ell+1) + \sum_{i=1}^{\ell} |P(t'_i, F)\downarrow|_{\mathsf{E}} \le \ell + 1 + \ell \cdot h|F| \cdot (2|Q|+|F|) \\ &\le h \cdot |F| \cdot (2|Q|+|F|)^2,\end{aligned}$$

which proves the lemma. ◀

C Counter automata

Proof of theorem 5.1. We have seen that $L(\mathcal{B}_3)\downarrow = L(\mathcal{A})\downarrow$. To estimate the size of $\mathcal{B}_3$, notice that $|[-B,B]^k| = (2B+1)^k \leq (3n)^{((k+1)^2+1)k} \leq (3n)^{5k^3}$. Furthermore, our stack alphabet satisfies $|\Gamma| = n\cdot(2n+1)^k$, so that $|\Gamma^{\leq n}| \leq |\Gamma|^{n+1} = n^{n+1}\cdot(2n+1)^{(n+1)k} \leq (3n)^{4nk}$. Finally, we can estimate $|\mathbb{I}([-n,n]^k)| \leq \binom{(2n+1)^k}{k} \leq (2n+1)^{k^2} \leq (3n)^{k^2}$. This means in total $|Q_3| \leq n\cdot(3n)^{4nk+5k^3+2k^2} \leq (3n)^{5nk+7k^3}$. This completes the proof of theorem 5.1. ◀

Proof of proposition 5.3 We prove proposition 5.3 in the following two lemmas.

▶ **Lemma C.1.** $L(\mathcal{A}) \subseteq L(\mathcal{B}_1)$.

Proof. Let $w \in E^*$ be an accepting walk of $\mathcal{A}$. We can write $w = u_0v_1u_1\cdots v_\ell u_\ell$ such that $|u_0\cdots u_\ell| \leq n$ and every v_i is a prime cycle. For each v_i, lemma 4.1 yields a tree decomposition t_i of height at most n. In $\mathcal{B}_1$, we simulate $u_0\cdots u_\ell$ by transitions of type (1). When we arrive at a prime cycle v_i, we traverse the tree t_i: When at the current state a subtree is attached in t_i, we use a transition of type (2). When we arrive at the state where our current cycle has started, we use either (4) or (5) to use the cycle as a precise cycle or as an obligation cycle, respectively. During a cycle, we use transitions (3).

It remains to be shown that there exists a choice of cycles as 'precise' or 'oligation' to obtain an accepting run of $\mathcal{B}_1$, i.e. the capacity in the factor $[-B,B]^k$ is not exceeded and the sets S and T in the factors $\mathbb{P}([-n,n]^k)$ form a cancellable (S,T). To this end, we apply theorem 5.2. Let $e_1,\ldots,e_m \in \mathbb{Z}^k$ be the different effects (in any order) of the (simple) cycles in all the tree decompositions t_i, $i \in [1,\ell]$. Being effects of simple cycles, they are even contained in $[-n,n]^k$. For each $i \in [1,m]$, let x_i be the number of times e_i occurs as an effect of a cycle. Let e be the effect of the walk $u_0\cdots u_\ell$. Then we have $e \in [-n,n]^k$. Since the walk $w = u_0v_1u_1\cdots v_\ell u_\ell$ is accepting in $\mathcal{A}$, we have $e + \sum_{i=1}^m x_ie_i = 0$.

Consider the matrix A with columns $e_1,\ldots,e_m$. Then for the vector $x = (x_1,\ldots,x_m)$, we have $Ax = e$. Since the e_i are pairwise distinct and members of $[-n,n]^k$, we have $m \leq (2n+1)^k$. Therefore, we have $\|(A|e)\|_{1,\infty} \leq (n+1)m \leq (2n+1)^{k+1}$. Moreover, A has rank at most k. By theorem 5.2, there exists a $y \in \mathbb{N}^m$ with $Ay = -e$, $y \leq x$, and $\|y\|_1 \leq (1+(2n+1)^{k+1})^{k+1} \leq (3n)^{(k+1)^2}$. We can therefore choose for each $i \in [1,m]$, y_i of the x_i cycles with effect e_i and use them as precise cycles. Then, in the end, we arrive at a state (q,ε,v,S,T) with $v = 0$. Since we used at most $\|y\|_1$ precise cycles, the counter values encountered during the computation are bounded in absolute value by $n\cdot(3n)^{(k+1)^2} = B$.

Observe that we have $T = \{e_1,\ldots,e_m\}$. Consider $z \in \mathbb{N}^m$ with $z = x - y$. By our choice of precise cycles, $S = \{e_i \mid z_i > 0\}$. Therefore, since $Az = 0$, the pair (S,T) is cancellable. Hence, we have reached a final state of $\mathcal{B}_1$ and read the same word as w. ◀

▶ **Lemma C.2.** $L(\mathcal{B}_1) \subseteq L(\mathcal{A})\downarrow$.

Proof. Consider a walk w in $\mathcal{B}_1$ from $(p,\varepsilon,0,S,T)$ to (q,ε,v,S',T'). Then $S' \setminus S \subseteq T' \setminus T$, so let $S' \setminus S = \{e_1,\ldots,e_m\}$ and $T' \setminus T = \{e_1,\ldots,e_{m+\ell}\}$. Moreover, for each $i \in [1,m]$, let $x_i \in \mathbb{N}\setminus\{0\}$ be the number of times a cycle with effect e_i was used as an obligation cycle. Let $x \in \mathbb{N}^{m+\ell}$ be the vector $x = (x_1,\ldots,x_{m+\ell})$ where $x_{m+i} = 0$ for $i \in [1,\ell]$.

It is easy to show by induction on the height of the maximal stack height in w that for every $y \in \mathbb{N}^{m+\ell}$ with $y \geq x$, there exists a walk w' in $\mathcal{A}$ from $(p,0)$ to $(q,v+Ay)$ such that w' reads a superword of the input of w: We execute all the obligation cycles as normal cycles in $\mathcal{A}$, which means adding the effect Ax. Then, for each effect e_i, we execute some cycle with effect e_i an additional $y_i - x_i$ times. In total, we add $v + Ay$ to the counter in $\mathcal{A}$.

Now suppose w is an accepting walk. Then $S = T = \emptyset$, the pair (S', T') is cancellable, and $v = 0$. Since (S', T') is cancellable, there is a $z \in \mathbb{N}^{m+\ell}$ with $z_i \geq 1$ for $i \in [1, m]$ such that $Az = 0$. Since $x_{m+i} = 0$ for $i \in [1, \ell]$, we can find a number $M \in \mathbb{N} \setminus \{0\}$ such that $Mz \geq x$. We apply our observation above to $y = Mz$. Since $y \geq x$, this yields a walk w' in $\mathcal{A}$ from $(p, 0)$ to $(q, v + Ay) = (q, 0 + MAz) = (q, 0)$ such that w' reads a superword of the word read by w. This means, w' is accepting, so that the word read by w is contained in $L(\mathcal{A})\downarrow$. ◀

Proof of lemma 5.4. We may clearly assume that $|S_2 \setminus S_1| = 1$. Hence, let (S_1, T) be cancellable, $S_1 = \{u_1, \ldots, u_s\}$, and $S_2 = \{u_1, \ldots, u_{s+1}\}$. Since $\operatorname{span}(S_1) = \operatorname{span}(S_2)$, there are $z_1, \ldots, z_{s+1} \in \mathbb{Q}$ with $z_{s+1} \neq 0$ and $\sum_{i=1}^{s+1} z_i u_i = 0$. By multiplying with a common denominator and, if necessary, switching the sign of the $z_1, \ldots, z_{s+1}$, we may assume that $z_1, \ldots, z_s \in \mathbb{Z}$ and $z_{s+1} \in \mathbb{N} \setminus \{0\}$.

Let $T = \{v_1, \ldots, v_t\}$. Since (S_1, T) is cancellable, there are are $x_1, \ldots, x_s \in \mathbb{N} \setminus \{0\}$ and $y_1, \ldots, y_t \in \mathbb{N}$ with

$$x_1 u_1 + \cdots + x_s u_s \quad + \quad y_1 v_1 + \cdots + y_t v_t = 0.$$

Since $x_i \geq 1$ for $i \in [1, s]$, we can find $M \in \mathbb{N} \setminus \{0\}$ with $M \cdot x_i > -z_i$ for every $i \in [1, s]$. Then, since $\sum_{i=1}^{s+1} z_i u_i = 0$, we have

$$0 = M \left(\sum_{i=1}^{s} x_i u_i + \sum_{i=1}^{t} y_i v_i \right) + \sum_{i=1}^{s+1} z_i u_i = \sum_{i=1}^{s} (Mx_i + z_i) u_i + z_{s+1} u_{s+1} + \sum_{i=1}^{t} (My_i) v_i$$

Since $Mx_i + z_i \in \mathbb{N} \setminus \{0\}$ for $i \in [1, s]$ and $z_{s+1} \in \mathbb{N} \setminus \{0\}$, this proves that (S_2, T) is cancellable. ◀

Proof of lemma 5.5. Let $S = \{u_1, \ldots, u_s\}$ and choose $T' \subseteq T$ minimal with the property that (S, T') is cancellable. Let $T' = \{v_1, \ldots, v_t\}$. Then there are $x_1, \ldots, x_s \in \mathbb{N} \setminus \{0\}$ and $y_1, \ldots, y_t \in \mathbb{N}$ with $\sum_{i=1}^{s} x_i u_i + \sum_{i=1}^{t} y_i v_i = 0$. By minimality of T', we have $y_i > 0$ for every $i \in [1, t]$. Suppose T' is linearly dependent. Then there are $z_1, \ldots, z_t \in \mathbb{Z}$, not all zero, such that $\sum_{i=1}^{t} z_i v_i = 0$. We may assume that at least one z_i is positive, because otherwise they are all at most zero and we can negate them.

Choose $j \in [1, t]$ such that z_j / y_j is maximal, meaning $z_j / y_j \geq z_i / y_i$ for every $i \in [1, t]$. Note that then $z_j > 0$ because otherwise, $z_i \leq 0$ for every $i \in [1, t]$. Then we have $z_j y_i \geq y_j z_i$ for every $i \in [1, t]$ and hence

$$0 = z_j \left(\sum_{i=1}^{s} x_i u_i + \sum_{i=1}^{t} y_i v_i \right) - y_j \left(\sum_{i=1}^{t} z_i v_i \right) = \sum_{i=1}^{s} (z_j x_i) u_i + \sum_{i=1}^{t} \underbrace{(z_j y_i - y_j z_i)}_{\geq 0} v_i.$$

Since $z_j x_i > 0$ for $i \in [1, s]$ and we have the coefficient $z_j y_j - y_j z_j = 0$ in front of v_j, the last equation tells us that $(S, T' \setminus \{v_j\})$ is cancellable. This contradicts the choice of T'. Therefore T' is linearly independent. ◀

Proof of corollary 5.6. Suppose we are gven an ideal $I = Y_0^* \{x_1, \varepsilon\} Y_1^* \cdots \{x_\ell, \varepsilon\} Y_\ell^*$ and a blind k-counter automaton $\mathcal{A}$ with n states. By theorem 5.1, we have an exponential bound upper bound m on $|L(\mathcal{A})|_!$. According to proposition 3.2, we have $I \subseteq L(\mathcal{A})\downarrow$ if and only if $w := w_{Y_0}^m x_1 w_{Y_1}^m \cdots x_\ell w_{Y_\ell}^m \in L\downarrow$. Now the word w may be exponentially long, but since m is at most exponential, we can compute m in binary representation.

Given the polynomial-size ideal and the binary representation of m, we can construct a polynomial-size straight-line program $\mathcal{G}$ for w: A *straight-line program* (SLP) is a context-free

grammar that generates exactly one word (see [27] for details and a survey). $\mathcal{G}$ is obtained from an SLP for the polynomial-length word $z_0x_1z_1\cdots x_\ell z_\ell$ and SLPs for the words $w_{Y_i}^m$, which in turn result from an SLP for $\{a^m\}$. The latter is easily constructed from the binary representation of m.

Therefore, it remains to be shown that the *compressed membership problem* for blind counter automata is decidable in NP. The latter asks, given an SLP $\mathcal{G}$ and a blind counter automaton $\mathcal{A}$, whether the word generated by $\mathcal{G}$ is accepted by $\mathcal{A}$. This can be decided by constructing an automaton that has access to a pushdown and blind (or reversal-bounded) counters that accepts $L(\mathcal{G}) \cap L(\mathcal{A})$. For such automata, the emptiness problem is in NP, as shown by Hague and Lin [16]. ◀

Proof of theorem 5.7. We show that for every $u \in L(\mathcal{A})$, there exists an ideal I of length $\leq (6n)^{4(k+1)^2}$ with $u \in I \subseteq L{\downarrow}$.

So let $w \in \Delta^*$ be an accepting walk of $\mathcal{A}$. We can write $w = u_0v_1u_1\cdots v_\ell u_\ell$ such that $u_0, \ldots, u_\ell \in \Delta$, $\ell \leq n$, and every v_i is a cycle. We factorize each $v_i = v_{i,1}\cdots v_{i,k_i}$ into prime cycles $v_{i,1}, \ldots, v_{i,k_i}$ and let lemma 4.1 provide a tree decomposition $t_{i,j}$ of $v_{i,j}$ of height at most n.

Let $e_1, \ldots, e_m \in \mathbb{Z}^k$ be the effects of cycles occurring in any of these trees. Note that $\|e_i\|_1 \leq n$ for $i \in [1, m]$, so that $m \leq (3n)^k$. For $i \in [1, m]$, let $x = (x_1, \ldots, x_m) \in \mathbb{N}^m$ be the vector such that x_i be the number of times a cycle with effect e_i occurs. Moreover, let $e \in \mathbb{Z}^k$ be the effect of $u_0\cdots u_\ell$. Then we have $\|e\|_\infty \leq n+1$. Let $A \in \mathbb{Z}^{k\times m}$ be the matrix with columns $e_1, \ldots, e_m$. Since w is accepting, we have $Ax = -e$. Note that

$$\|(A|e)\|_{1,\infty} \leq \|e\|_\infty + \sum_{i=1}^{m} \|e_i\|_\infty \leq (m+1)(n+1) \leq ((3n)^k+1)(n+1) \leq (4n)^{k+1}$$

and that the rank of $(A|e)$ is at most k. According to theorem 5.2, there is a $y \in \mathbb{N}^m$, $y \leq x$, such that $Ay = -e$ and $\|y\|_1 \leq (1+(4n)^{k+1})^{k+1} \leq (5n)^{(k+1)^2}$.

From our tree decomposition, we now select for each $i \in [1, m]$, y_i-many vertices whose cycles have effect e_i. This is possible since $y \leq x$. Let F be the set of these vertices. Then we have $|F| \leq \|y\|_1 \leq (5n)^{(k+1)^2}$. We claim that the language

$$K = a_0P(t_{1,1}\cdots t_{1,k_i}, F)a_1\cdots P(t_{\ell,1}\cdots t_{\ell,k_\ell}, F)a_\ell$$

is contained in $L(\mathcal{A}){\downarrow}$. Let $z = (z_1, \ldots, z_m) \in \mathbb{N}^m$ be the vector with $z = x - y$. Then $Az = 0$ and every pumpable vertex has an effect e_i with $z_i \geq 1$.

Now suppose we obtain a run w' of $\mathcal{A}$ by performing some pumping to obtain a word $u' \in K$, either by duplicating a single vertex or by duplicating a whole pumpable subtree. Note that it might happen that w' does not leave the counters at zero in the end. But we will show that we can pump even more to get such a walk. For each $i \in [1, m]$, let $z_i' \in \mathbb{N}$ be the number of times we add an occurrence of a cycle with effect e_i. Let $z' = (z_1', \ldots, z_m')$. Since $z_i' \geq 1$ implies $z_i \geq 1$, we can find an $N \in \mathbb{N}$ with $N \cdot z \geq z'$. Now for every $i \in [1, m]$, we can find a pumpable vertex v_i whose cycle has effect e_i. We can pump v_i an additional $Nz_i - z_i'$ times. This results in a walk w'' of A with effect $e + Ax + ANz = e + Ax = 0$, meaning that it is accepting. Moreover, if u'' is the input word read by w'', then we have $u' \preceq u'' \in L(\mathcal{A})$. This proves $K \subseteq L(\mathcal{A}){\downarrow}$, which was our claim.

This means that the language

$$I = K{\downarrow} = \{a_0, \varepsilon\}P(t_{1,1}\cdots t_{1,k_i}, F){\downarrow}\{a_1, \varepsilon\}\cdots P(t_{\ell,1}\cdots t_{\ell,k_\ell}, F){\downarrow}\{a_\ell, \varepsilon\}$$

is contained in $L(\mathcal{A})\!\downarrow$. By lemma 4.2, it is an ideal and satisfies

$$|I|_{\mathsf{E}} \leq \ell + \sum_{i=1}^{\ell} |P(t_{i,1} \cdots t_{i,k_i}, F)\!\downarrow|_{\mathsf{E}} \leq 2n^2 \cdot |F| \cdot (2n + |F|)^2 \leq (6n)^{4(k+1)^2}.$$

◀

D Context-Free Grammars

Proof of lemma 6.2. Suppose $a_1^* \cdots a_n^* \subseteq L(G)\!\downarrow = L(G')$. Then there are derivation trees $t_1, t_2, \ldots$ of G' with $|\mathsf{yield}(t_j)|_{a_i} \geq j$ for every $i \in [1, n]$ and $j \geq 1$.

On the vertices t_j, we define partial orders $\ll_i$ as follows. We have $u \ll_i v$ if v is a descendent of u and $|\mathsf{yield}(u)|_{a_i} > |\mathsf{yield}(v)|_{a_i}$. By induction on ℓ, it is easy to check that if all $\ll_i$-chains in t_j have length $\leq \ell$, then $|\mathsf{yield}(t_j)|_{a_i} \leq 2^\ell$. Hence, if $m > 2^{|N|}$, then $|\mathsf{yield}(t_m)|_{a_i} > 2^{|N|}$, so that t_m must have a $\ll_i$-chain of length $> |N|$. On this chain, some $A_i \in N$ has to repeat, meaning $A_i \in L_i \cup R_i$. We can therefore expand t_m by applying for each $i \in [1, n]$ the production $A_i \to a_i^\omega A_i$ or $A_i \to A_i a_i^\omega$. Then, we replace every a_i-leaf by ε. By construction, the resulting tree t is a derivation tree of G^ω and every a_i^ω appears exactly once. Hence, $\mathsf{yield}(t)$ is a permutation of $a_1^\omega \cdots a_n^\omega$. It remains to be shown that $\mathsf{yield}(t) = a_1^\omega \cdots a_n^\omega$.

Consider the morphism $\alpha\colon \{a_1^\omega, \ldots, a_n^\omega\}^* \to \{a_1, \ldots, a_n\}^*$ such that for every $i \in [1, n]$, we have $\alpha(a_i^\omega) = a_i$. Observe that for every production $A \to a_i^\omega A$ or $A \to A a_i^\omega$, we have $A \Rightarrow_{G'}^* a_i A$ or $A \Rightarrow_{G'}^* A a_i$, respectively. This tells us that $\alpha(L(G^\omega)) \subseteq L(G') \subseteq a_1^* \cdots a_n^*$. Therefore, $\alpha(\mathsf{yield}(t)) = a_1 \cdots a_n$ and hence $\mathsf{yield}(t) = a_1^\omega \cdots a_n^\omega$. ◀

E Hardness

Proof of theorem 7.1. We actually prove a stronger statement, namely that the following problem is hard for $\mathsf{coNTIME}(t)$:

Given: A description in $\mathcal{M}$ of the language $X^{\leq m}$ and a description in $\mathcal{N}$ of a language $L \subseteq X^{\leq m}$, where X is an alphabet and $m \in \mathbb{N}$.
Question: Does $X^{\leq m}\!\downarrow \subseteq L\!\downarrow$ hold?

This is clearly an instance of both $\mathcal{M} \subseteq_\downarrow \mathcal{N}$ and of $\mathcal{M} =_\downarrow \mathcal{N}$. If we show that already this special case is hard, then so is the case of binary alphabets: Suppose $X = \{a_1, \ldots, a_k\}$ and let $\gamma\colon X^* \to \{a, b\}$ be the morphism with $\gamma(a_i) = a^i b^{k-i}$ for $i \in [1, k]$. Then clearly $\gamma(X^{\leq m})\!\downarrow \subseteq \gamma(L)\!\downarrow$ if and only if $X^{\leq m}\!\downarrow \subseteq L\!\downarrow$. Hence, we only show hardness for the problem above.

Let $K \subseteq Y^*$ belong to $\mathsf{coNTIME}(t)$. Then there is a $t(n^c)$-time-bounded ($c \geq 1$) Turing machine M with one tape, tape alphabet $Z \supseteq Y$ (which includes the blank symbol), and state set Q that accepts the complement of K.

Our goal is to construct the language $L \subseteq X^{\leq m}$ in such a way that the words in L of length m are precisely those words that do not encode an accepting computation of M. Here, m will be chosen so that if M has an accepting computation, it is encoded by a word of length m. Then, we will have $X^{\leq m}\!\downarrow \subseteq L\!\downarrow$ if and only if M does not accept the given input word. Our first task is to find a suitable m.

Observe that a function $h\colon \mathbb{N} \to \mathbb{N}$ is amplifying if and only if $h(n) \geq n$ for $n \geq 0$ and there is a $d \geq 1$ such that $h(n^d) \geq h(n)^2$ for large enough n. Let $g\colon \mathbb{N} \to \mathbb{N}$ be defined as $g(n) = t(n^c)$. Since t is amplifying, g is as well: for some constant d, we have

$g(n^d) = t(n^{cd}) \geq t(n^c)^2 = g(n)^2$ for large enough $n \in \mathbb{N}$. Since g is amplifying, there is a constant $e \geq 1$ such that $g(n^e) \geq g(n)^2$ for all $n \geq n_0$. We define $f(n) = g(n^e)$.

With these choices, we have: M is time bounded by g, the models $\mathcal{M}$ and $\mathcal{N}$ are $\Delta(g)$ and $\Delta(f)$, and $f(n) \geq g(n) \cdot (g(n) + 2)$ for $n \geq n_0$.

Now fix $w \in K$ and let $n = |w|$. For the reduction, it means no loss of generality to assume $n \geq n_0$. We choose $m = f(n) + g(n) + 3$. We encode a configuration of M by a word uqv, where $u, v \in Z^*$, $q \in Q$, and $|uv| = g(n)$ (recall that M is g-time-bounded and hence g-space-bounded). It means that M is in state Q and its head is at the first position of v. A computation is then encoded as a word $\#u_1\# \cdots u_k\#w_{k+1}$, where $u_1, \ldots, u_k$ encode the configurations of the computation (in this order) and w_{k+1} is any suffix in Z^*. Since $m = f(n) + g(n) + 3 \geq g(n) \cdot (g(n) + 3)$ and M is g-time-bounded, all computations have encodings where $|\#u_1 \cdots \#u_{k+1}| = m$.

Since $\mathcal{M}$ and $\mathcal{N}$ are $\Delta(g)$ and $\Delta(f)$, we can construct for each model finite languages whose longest word has length $g(n)$ or $f(n)$, respectively. By applying a homomorphism and taking the downward closure, we can thus construct descriptions of $\{a^{g(n)}\}\downarrow$ and of $\{a^{f(n)}\}\downarrow$ in each of the models, in polynomial time. Let $X = Z \cup Q \cup \{\#\}$. Using rational transductions and simple substitutions, we get $X^{\leq m} = X^{f(n)+g(n)+3}\downarrow$ in $\mathcal{M}$ and

$$L_1 = \{a^i\#a^{g(n)+1}\#a^{f(n)-i} \mid 0 \in [0, f(n)]\}\downarrow$$

in $\mathcal{N}$. Note that $L_1 \subseteq \{a, \#\}^{\leq m}$.

In the rest of the proof, we construct a (polynomial-size) rational transduction T such that $TL_1 \subseteq X^{\leq m}$ and $(TL_1) \cap X^m$ contains precisely those words that do not encode a computation of M that accepts w. Then, we have clearly shown that the problem described at the beginning of the proof is hard for $\mathsf{coNTIME}(t)$.

A word $u \in X^*$ of length m can fail to be an accepting computation for w for the following reasons. We decompose $u = u_0\#u_1\# \cdots \#u_{k+1}$.

1. It does not begin with #, i.e. $u_0 \neq \varepsilon$.
2. Two #'s are less than $g(n) + 1$ positions apart.
3. Two #'s are more than $g(n) + 1$ positions apart (without a # in between).
4. Some u_i is not contained in Z^*QZ^*.
5. The first configuration u_1 is not an initial configuration with input w.
6. The last configuration, i.e. u_k, is not accepting.
7. For some $i \in [1, k-1]$, the configuration u_i cannot reach u_{i+1} in one step.

For each of the cases 1 to 7, we shall explain how to obtain a transduction that generates those words from L_1. If we then have rational transductions $T_1, \ldots, T_7$, we take the rational transduction $T = T_1 \cup \cdots \cup T_7$, which is clearly as desired above.

Note that the cases 1 and 4 to 6 are trivial, so we consider cases 2, 3 and 7. For 2, notice that with a constant-size rational transduction $R_<$, one can obtain

$$P_< = \{a^i\#a^\ell\#a^{g(n)+1-\ell}a^{f(n)-i} \mid i \in [0, k],\ \ell \in [0, g(n) + 1]\}\downarrow.$$

as $R_<L_1$. Indeed, $R_<$ reads a word from L_1 and outputs every letter as read, up to the first #. Then, before it sees the second # in the input, it nondeterministically chooses a time to output # early. Then, it reads the rest of the input and output a for each input letter, be it a or #. Using a similar strategy, one can obtain

$$P_> = \{a^i\#a^{g(n)+1+\ell}\#a^{f(n)-i-\ell} \mid i \in [0, k],\ \ell \in [0, f(n) - i]\}\downarrow$$

using a constant-size rational transudction $R_>$. Now from $P_<$ and $P_>$, it is easy to obtain all words of case 2 and 3, respectively.

The case 7 is also not hard to realize with L_1 as input. We only have to make sure that either the immediate surrounding of the head is not updated properly or the rest of the tape is not copied correctly. For words of length m (and those are the only ones where we must produce an incorrect encoding), the input language L_1 gives us, with the two #'s, two pointers that are precisely $g(n)+1$ positions apart. We can therefore guarantee that at least one of these errors is present. The details are very straightforward. ◀

Proof of corollary 7.2. According to theorem 7.1, it suffices to show that each $\mathcal{M} \in \{\mathsf{CFG}, \mathsf{RBC}\}$ is $\Delta(2^n)$.

For CFG, we can take the well-known grammar with nonterminals $A_0, \dots, A_n$, start symbol A_{n-1}, and productions $A_i \to A_{i-1}A_{i-1}$ for $i \in [1,n]$, and $A_0 \to a$. It clearly generates $\{a^{2^n}\}$.

For RBC, we use a blind $(n+1)$-counter automaton. We increment the first counter once and then, for each $i = 1, \dots, n$, we count down counter i and simultaneously count up counter $i+1$ at twice the speed. After these n phases, counter $n+1$ contains the value 2^n. Then, we count down counter $n+1$ and each time read an a. Hence, we accept $\{a^{2^n}\}$. ◀

Proof of proposition 7.5. The *generalized subset sum problem* is the following:

Given: Two vectors $u, v \in \mathbb{N}^n$ and $t \in \mathbb{N}$, encoded in binary.
Question: Is it true that for every $x \in \{0,1\}^n$, there exists a $y \in \{0,1\}^n$ that satisfies $\langle u, x\rangle + \langle v, y\rangle = t$?

Here, $\langle w, z\rangle$ denotes the scalar product of $w, z \in \mathbb{Z}^n$. This problem is known to be Π_2^{P}-complete [6].

We identify vectors over $\{0,1\}$ of length n with words over $\{0,1\}$ of length n. Let $u, v \in \mathbb{N}^n$ and $t \in \mathbb{N}$ be an instance of the generalized subset sum problem and suppose each entry of u and v is encoded with k bits. Like in corollary 7.2, we can easily construct an RBC $\mathcal{A}$ with $3k$ counters that accepts $\{x \in \{0,1\}^n \mid \exists y \in \{0,1\}^n \colon \langle u, x\rangle + \langle v, y\rangle = t\}$: As it reads x, it uses counters $1, \dots, k$ to build up $\langle u, x\rangle$ in counter k. Then, it guesses y bit-by-bit while using counters $k+1, \dots, 2k$ to build up $\langle v, y\rangle$ in counter $2k$. Afterwards, it accumualates t in counter $3k$ using counters $2k+1, \dots, 3k$. Finally, it counts down counter $3k$ one-by-one and in each step, decrements counter k or $2k$. In the end, all counters are zero if and only if $\langle u, x\rangle + \langle v, y\rangle = t$.

Let $\mathcal{B}$ be the obvious $(n+1)$-state NFA that accepts $\{0,1\}^n$. Then we clearly have $L(\mathcal{B}){\downarrow} \subseteq L(\mathcal{A}){\downarrow}$ if and only if our instance of the generlized subset sum problem is positive. ◀