Directed Regular and Context-Free Languages

Moses Ganardi
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Irmak Sağlam
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

We study the problem of deciding whether a given language is directed. A language L is *directed* if every pair of words in L have a common (scattered) superword in L. Deciding directedness is a fundamental problem in connection with ideal decompositions of downward closed sets. Another motivation is that deciding whether two *directed* context-free languages have the same downward closures can be decided in polynomial time, whereas for general context-free languages, this problem is known to be coNEXP-complete.

We show that the directedness problem for regular languages, given as NFAs, belongs to AC^1, and thus polynomial time. Moreover, it is NL-complete for fixed alphabet sizes. Furthermore, we show that for context-free languages, the directedness problem is PSPACE-complete.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Concurrency; Theory of computation → Formal languages and automata theory

Keywords and phrases Subword, ideal, language, regular, context-free, equivalence, downward closure, compression

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.36

Related Version *Extended Version*: `https://arxiv.org/abs/2307.13396` [27]

Funding Funded by the European Union (ERC, FINABIS, 101077902). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

We study the problem of deciding whether a given language is directed. A language L is called *(upward) directed* if for every $u, v \in L$, there exists a $w \in L$ with $u \preccurlyeq w$ and $v \preccurlyeq w$. Here, $\preccurlyeq$ denotes the (non-contiguous) *subword* relation: We have $u \preccurlyeq v$ if there are decompositions $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$ for some $u_1, \ldots, u_n \in \Sigma^*$ and $v_0, v_1, \ldots, v_n \in \Sigma^*$.

Downward closures and ideals. The *downward closure* of a language $L \subseteq \Sigma^*$ is the set $L\!\downarrow = \{u \in \Sigma^* \mid \exists v \in L\colon u \preccurlyeq v\}$. Over the last ca. 15 years, downward closures have been used in several approaches to verifying concurrent systems. This has two reasons: First, $L\!\downarrow$ is a regular language for every set $L \subseteq \Sigma^*$ [32] and an NFA can often be computed effectively [6, 21, 23, 30, 31, 47, 49, 50]. Second, many verification tasks are *downward closure invariant* w.r.t. subsystems: This means, a (potentially infinite-state) subsystem (e.g. a recursive program represented by a context-free language) can be replaced with another with the same downward closure, without affecting the verified property. This has been applied to parameterized systems with non-atomic reads and writes [46], concurrent programs with dynamic thread creation [5, 11, 13], asynchronous programs [10, 41], and thread pools [14].

In addition to finite automata, there is a second representation of downward closed languages: Every non-empty downward closed set can be written as a finite union of ideals. An *ideal* (in the terminology of well-quasi orderings) is a non-empty downward closed set that is directed. Moreover, ideals have a simple representation themselves: They are precisely the products of languages of the forms $\{a, \varepsilon\}$ and Δ^*, where a is a letter and Δ is an alphabet. Clearly, a language L is directed if and only if $L\downarrow$ is itself an ideal.

Ideal decompositions of downward closed sets have recently been the center of significant attention: They have been instrumental in computing downward closures [9, 28, 49] and deciding separability by piecewise testable languages [29,51]. Over other orderings, ideals play a crucial role in forward analysis of well-structured transition systems (WSTS) [16, 25, 26], infinitely branching WSTS [18], well-behaved transition systems [17] for clarifying reachability problems in vector addition systems [36–38], and for deciding regular separability [24].

Given the importance of ideals, it is a fundamental problem to decide whether a given language is directed, in other words, whether the ideal decomposition of its downward closure consists of a single ideal. It is a basic task for computing ideal decompositions, but also an algorithmic lens on the structure of ideals.

Efficient comparison. Aside from being a fundamental property, checking directedness is also useful for deciding equivalence. It is well-known that equivalence is PSPACE-complete for NFAs and undecidable for context-free languages. However, in some situations, it suffices to decide *downward closure equivalence*: Due to the aforementioned downward closure invariance in concurrent programs, if $L_1, L_2 \subseteq \Sigma^*$ describe the behaviors of sequential programs inside of a concurrent program, and we have $L_1\downarrow = L_2\downarrow$, then L_1 can be replaced with L_2 without affecting safety, boundedness, and termination properties in concurrent [11] and asynchronous programs [41]. Downward closure equivalence is known to be coNP-complete for NFAs [8, Thm. 12&13] and coNEXP-complete for context-free languages [50]. This is better than PSPACE and undecidable, but our results imply that if L_1 and L_2 are directed, then deciding $L_1\downarrow = L_2\downarrow$ is *in polynomial time*, both for NFAs and for context-free languages! Thus, directedness drastically reduces the complexity of downward closure equivalence.

Constraint satisfaction problems. Directedness has recently also been studied in the context of constraint satisfaction problems (CSPs) for infinite structures. If we view finite words in the usual way as finite relational structures (as in first-order logic), then a set of words is directed if and only if it has the *joint embedding property (JEP)*. More generally, a class $\mathcal{C}$ of finite structures has the JEP if for any two structures in $\mathcal{C}$, there is a third in which they both embed. The JEP is important for CSPs because if $\mathcal{C}$ is definable by a universal first-order formula and has the JEP, then it is the *age* of some (potentially infinite) structure, which then has a constraint satisfaction problem in NP [19, p. 1].

Motivated by this, it was recently shown that the JEP is undecidable for universal formulas by Braunfeld [20] (and even for universal Horn formulas by Bodirsky, Rydval, and Schrottenloher [19]). In the special case of finite words, the JEP (and thus directedness) was shown to be decidable in polynomial time by Atminas and Lozin [7] for regular languages of the form $\{w \in \Sigma^* \mid w_1, \ldots, w_n \not\preccurlyeq w\}$ for given $w_1, \ldots, w_n \in \Sigma^*$. However, to our knowledge, for general regular languages (or even context-free languages), the complexity is not known.

Contribution. Our first main result is that for NFAs, directedness is decidable in AC^1, a circuit complexity class within polynomial time, defined by Boolean circuits of polynomial size, logarithmic depth, and unbounded fan-in. If we fix the alphabet size, directedness becomes NL-complete. Our second main result is that for context-free languages, directedness is PSPACE-complete, and hardness already holds for input alphabets of size two.

The proof techniques for the main results also yield algorithms for downward closure equivalence. Given L_1 and L_2, we show that deciding $L_1\downarrow = L_2\downarrow$ is in AC^1 if L_1 and L_2 are directed and given by NFAs. As above, we obtain NL-completeness for fixed alphabets. If L_1 and L_2 are context-free languages, then deciding $L_1\downarrow = L_2\downarrow$ becomes P-complete.

Finally, we mention that counting the number of ideals in the ideal decomposition of $L\downarrow$ is #P-complete if L is given as an NFA. Here, hardness follows from #P-hardness of counting words in NFAs of a given length, and #P-membership is a consequence of our methods.

Key ingredients. The upper bounds as well as the lower bounds in our results rely on new techniques. With only slight extensions of existing techniques, one would obtain an NP upper bound for regular languages and an NEXP upper bound for context-free languages. This is because given a regular or context-free language, one can construct an acyclic graph where every path corresponds to an ideal of its downward closure. If the input is an NFA, this graph is polynomial-sized, and for CFGs, it is exponential-sized. One could then guess a path and verify that the entire language is included in this candidate ideal.

To obtain our upper bounds, we introduce a *weighting technique*, where each ideal is assigned a weight in the natural numbers. The weighting function has the property that if there is an ideal that contains the entire language, it must be one with maximal weight. Using either (i) matrix powering over the semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ for NFAs or (ii) dynamic programming for context-free grammars, this allows us to compute in NL/AC^1/polynomial time a unique candidate ideal (which is compressed in the context-free case), which is then verified in NL resp. in PSPACE.

For the PSPACE lower bound for context-free languages, we first observe that directedness is equivalent to deciding whether $L \subseteq I$, where L is a context-free language, and I is an SLP-compressed ideal. The problem is thus a slight generalization of the *compressed subword problem*, where we are given two SLP-compressed words u and v and are asked whether $u \preccurlyeq v$. This problem is known to be in PSPACE and PP-hard [39, Theorem 13], but its exact complexity is a long-standing open problem [40].

To exploit the increased generality of our problem, we proceed as follows. Our key insight is that for a given SLP-compressed word $w \in \Sigma^*$, one can construct an SLP-compressed infinite *complement ideal* I_w, meaning that $I_w \cap \Sigma^{|w|} = \Sigma^{|w|} \setminus \{w\}$. To this end, we apply a construction from definability of languages in the subword ordering [12, Lemma 3.1]. We use the complement ideal for PSPACE-hardness follows. We reduce from the PSPACE-complete problem of deciding, given two equal-length SLP-compressed words $v, w \in \{a, b\}^*$, whether their convolution $v \otimes w$ belongs to a fixed regular language [40, p. 269]. We reduce this to the problem of deciding $w \in L$, where w is SLP-compressed and L is a context-free of words of length $|w|$. Then $L \subseteq I_w$ if and only if $w \notin L$, hence $L \cup I_w$ is directed if and only if $w \notin L$.

2 Main results

Our first main result is that for a given NFA, one can decide directedness of its language in polynomial time, and even in $\mathsf{AC}^1 \subseteq \mathsf{NC}$.

▶ **Theorem 2.1.** *Given an NFA, one can decide in* AC^1 *whether its language is directed.*

Recall that AC^1 is the class of all languages that are accepted by a family of unbounded fan-in Boolean circuits of polynomial size and logarithmic depth, see [48] for more details. In particular, the directedness problem for regular languages can be efficiently parallelized.

The same techniques show that fixing the input alphabet leads to NL-completeness:

▶ **Theorem 2.2.** *For every fixed k, given an NFA over k letters, it is* NL*-complete to decide whether its language is directed.*

As mentioned before, slight extensions of known techniques would yield an NP upper bound for directedness of regular languages: Given an NFA $\mathcal{A}$, it is not difficult construct an acyclic graph whose paths correspond to ideals for which $L(\mathcal{A}) = I_1 \cup \cdots \cup I_n$. One can then guess such a path with ideal I_i and verify $L(\mathcal{A}) \subseteq I_i$ in NL. Clearly, $L(\mathcal{A})$ is directed iff such an I_i exists. The key challenge is to compute in AC^1 a single ideal I_i for which we check $L(\mathcal{A}) \subseteq I_i$.

Note that Theorems 2.1 and 2.2 also apply to one-counter languages. Given a one-counter language L, one can compute in logspace an NFA $\mathcal{A}$ with $L(\mathcal{A}) = L\!\downarrow$ [6, Theorem 7][1]. Then $L(\mathcal{A})$ is directed iff L is directed, and we can just decide directedness for $\mathcal{A}$.

Our second main result is that for context-free languages (given, e.g. by a grammar), directedness is PSPACE-complete:

▶ **Theorem 2.3.** *Given a context-free grammar, it is* PSPACE*-complete to decide whether its language is directed. Moreover,* PSPACE*-hardness holds already for binary input alphabets.*

Our methods also provide more efficient algorithms for downward closure equivalence in the case of directed input languages. The *downward closure equivalence (DCE)* problem is to decide, for given languages L_1 and L_2, whether $L_1\!\downarrow = L_2\!\downarrow$. While DCE is coNP-complete for general regular languages given as NFAs [8, Thm. 12&13], we show that for directed input languages, the complexity drops to AC^1, and NL for fixed alphabets.

▶ **Theorem 2.4.** *For directed languages given as NFAs, DCE belongs to AC^1, for fixed alphabets even to* NL*.*

For context-free languages, DCE is known to be coNEXP-complete [50]. For directed input languages, our methods yield a drastic drop in complexity down to polynomial time:

▶ **Theorem 2.5.** *For directed context-free languages, DCE is* P*-complete.*

Since our directedness algorithms decide whether the unique decomposition of $L\!\downarrow$ into maximal ideals consists of a single ideal, it is natural to ask about the complexity of counting all ideals of this decomposition. It follows easily using methods developed here that this problem is in #P. Moreover, the well-known #P-hardness of #NFA (i.e. counting words of a given length in an NFA) [4] provides a #P lower bound.

▶ **Theorem 2.6.** *Given an NFA $\mathcal{A}$, it is* #P*-complete to count the number of ideals in the decomposition of $L(\mathcal{A})\!\downarrow$ into maximal ideals.*

The proof is given in [27, App. E] #P is the class of functions f computable by some non-deterministic polynomial-time Turing machine (TM), in the sense that for a given input x, the computed value $f(x)$ is the number of accepting runs. #P-complete problems are very hard, as evidenced by Toda's well-known result that $\mathsf{P}^{\#\mathsf{P}}$ (i.e. polynomial-time algorithms with access to #P oracles) includes the entire polynomial time hierarchy [45]. This represents an interesting contrast between the complexity of directedness and counting all ideals.

[1] Theorem 7 in [6] only states a polynomial time computation, but it is clear that it can be performed in (deterministic) logspace.

3 Preliminaries

Ideals. We will use the notation $[m_1, m_2] := \{i \in \mathbb{Z} \mid m_1 \leq i \leq m_2\}$. Consider the context-free languages $K_1 = \{wcw \mid w \in \{ab\}^*\}$ and $K_2 = \{wcw \mid w \in \{a\}^* \cup \{b\}^*\}$. Note that K_1 is directed, whereas K_2 is not: The words aca and bcb in K_2 have no common superword in K_2. However, $K_1 \cup K_2$ is directed. Let Σ be a finite alphabet. A set $D \subseteq \Sigma^*$ is *downward closed* if $D{\downarrow} = D$. A subset $I \subseteq \Sigma^*$ is an *ideal* if it is non-empty, downward closed, and (upward) directed. Thus clearly, a non-empty $L \subseteq \Sigma^*$ is directed if and only if $L{\downarrow}$ is an ideal (note that taking the downward closure does not affect directedness). It is known that every ideal can be written as products of so called *atoms*: We identify two types of atoms over Σ:

Single atoms: $a^{\textcircled{?}}$ where $a \in \Sigma$,

Alphabet atoms: $\Delta^{\circledast}$ where $\emptyset \neq \Delta \subseteq \Sigma$.

Formally, each atom α is a formal symbol that describes an ideal $\mathtt{Idl}(\alpha)$. For a single atom $a^{\textcircled{?}}$, we define it as $\mathtt{Idl}(a^{\textcircled{?}}) = \{a, \varepsilon\}$, whereas for an alphabet atom $\Delta^{\circledast}$, $\mathtt{Idl}(\Delta^{\circledast}) = \Delta^*$. By $\mathtt{atoms}(\Sigma)$, we denote the set of atoms over Σ. Note that $|\mathtt{atoms}(\Sigma)| = 2^{|\Sigma|} - 1 + |\Sigma|$.

An *ideal representation* is a finite (possibly empty) sequence $r = \alpha_1 \cdots \alpha_n$ of atoms α_i. Its language is the concatenation $\mathtt{Idl}(\alpha_1 \cdots \alpha_n) = \mathtt{Idl}(\alpha_1) \cdots \mathtt{Idl}(\alpha_n)$ where the empty concatenation is interpreted as $\{\varepsilon\}$. It is a classical fact that every downward closed set can be decomposed into a finite union of ideals. For example, observe that $K_1{\downarrow} = (K_1 \cup K_2){\downarrow} = \{a,b\}^*\{c,\varepsilon\}\{a,b\}^* = \mathtt{Idl}(\{a,b\}^{\circledast}c^{\textcircled{?}}\{a,b\}^{\circledast})$ and $K_2{\downarrow} = \{a\}^*\{c,\varepsilon\}\{a\}^* \cup \{b\}^*\{c,\varepsilon\}\{b\}^* = \mathtt{Idl}(\{a\}^{\circledast}c^{\textcircled{?}}\{a\}^{\circledast}) \cup \mathtt{Idl}(\{b\}^{\circledast}c^{\textcircled{?}}\{b\}^{\circledast})$. This decomposition result was first shown by Jullien [34] and the equivalent fact that every downward closed set can be expressed as a *simple regular expression* was shown independently by Abdulla, Bouajjani, and Jonsson [1] (see [33] for a general treatment). If R is a set of ideal representations, we set $\mathtt{Idl}(R) := \bigcup_{r \in R} \mathtt{Idl}(r)$.

Reduced ideal representations. Note that one ideal can have multiple different representations. For instance, the representations $a^{\circledast} \cdot a^{\textcircled{?}} \cdot b^{\textcircled{?}} \cdot b^{\circledast} \cdot a^{\textcircled{?}}$ and $a^{\circledast} \cdot b^{\circledast} \cdot a^{\textcircled{?}}$ *represent* the same ideal, namely all words that start with a (possibly empty) sequence of a's, followed by a (possibly empty) sequence of b's, and possibly end with an a. This is because in the first representation, all the words $a^{\textcircled{?}}$ and $b^{\textcircled{?}}$ generate are produced by their neighboring alphabet atoms. Two representations are called *equivalent* if they represent the same ideal.

To achieve unique ideal representations, one can use *reduced representations*, which we define next. Two atoms α and β are *absorptive* if $\mathtt{Idl}(\alpha\beta) = \mathtt{Idl}(\alpha)$ or $\mathtt{Idl}(\alpha\beta) = \mathtt{Idl}(\beta)$. In the first case we say α *absorbs* β and in the second case, β *absorbs* α. Note that two single atoms are always non-absorptive since $\mathtt{Idl}(a^{\textcircled{?}} \cdot b^{\textcircled{?}}) \supsetneq \mathtt{Idl}(a^{\textcircled{?}})$. An atom α is said to *contain* an atom β if $\mathtt{Idl}(\alpha) \supseteq \mathtt{Idl}(\beta)$. α is said to *strictly* contain β if also $\mathtt{Idl}(\alpha) \neq \mathtt{Idl}(\beta)$. An ideal representation $\alpha_1 \cdots \alpha_n$ is said to be *reduced* if for all $i \in [1, n-1]$, α_i and α_{i+1} are non-absorptive. The following is obvious (and well-known [2, Lemma 5.4]), because we can just repeatedly merge neighboring absorptive atom pairs:

▶ **Lemma 3.1.** *For every ideal representation $\alpha_1 \cdots \alpha_n$, there exists a reduced ideal representation $\beta_1 \cdots \beta_m$ such that $\mathtt{Idl}(\alpha_1 \cdots \alpha_n) = \mathtt{Idl}(\beta_1 \cdots \beta_m)$ and $m \leq n$.*

Representing downward closed sets. We will use two classical facts about ideals. First, every downward closed set $D \subseteq \Sigma^*$ can be written as a finite union of ideals. Moreover, ideals are "prime" in the sense that if an ideal is included in a union $D_1 \cup D_2$ of downward closed sets, it is already included in one of them:

▶ **Lemma 3.2** ([26, 28, 35]). *For every downward closed set $D \subseteq \Sigma^*$, there exist $n \in \mathbb{N}$ and ideals $I_1, \ldots, I_n \subseteq \Sigma^*$ with $D = I_1 \cup \cdots \cup I_n$. Moreover, if I is an ideal with $I \subseteq D_1 \cup D_2$ for downward closed $D_1, D_2 \subseteq \Sigma^*$, then $I \subseteq D_1$ or $I \subseteq D_2$.*

The representation $D = I_1 \cup \cdots \cup I_n$ is also called an *ideal decomposition* of D. Observe that the second statement implies that this decomposition is unique (up to the order of ideals) if we require the ideals $I_1, \ldots, I_n$ to be pairwise incomparable. This is sometimes called the unique *decomposition into maximal ideals.*

Non-deterministic finite automata. We start by formally introducing NFAs. A *non-deterministic finite automaton (NFA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, $q_0 \in Q$ is the unique *initial state*, $F \subseteq Q$ is the set of *final states*, Σ is a *finite alphabet* and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the set of *transitions.* A transition $(p, a, q) \in \delta$ is usually displayed as $p \xrightarrow{a} q$, and we write $p_0 \xrightarrow{w} p_n$ if there exists a sequence of transitions $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n$ such that $w = a_1 \ldots a_n$. The language accepted by an NFA $\mathcal{A}$ is the set of all words $w \in \Sigma^*$ such that $q_0 \xrightarrow{w} q$ for some $q \in F$, and is denoted by $L(\mathcal{A})$.

4 Solution on Regular Languages

In this section, we prove Theorem 2.1 and Theorem 2.2. Let us quickly observe the NL lower bound of the directedness problem: We reduce from the emptiness problem for NFAs. Given an NFA $\mathcal{A}$, we may assume that there is exactly one final state that is different from the initial state, and that all edges are labeled with a. We construct an NFA $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A}) \cup \{b\}$. Then clearly, $L(\mathcal{A}) \neq \emptyset$ if and only if $L(\mathcal{A})$ contains some word in $\{a\}^+$. The latter is true if and only if $L(\mathcal{A}')$ is not directed.

Thus, the interesting part of Theorems 2.1 and 2.2 are the upper bounds. To explain the main steps, we need some terminology. We say that a function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *computable in* NL if there exists a non-deterministic logspace TM with a write-only output tape such that (i) on every input word $x \in \{0,1\}^*$ there exists an accepting computation, and (ii) every accepting computation on x produces the same output $f(x)$ on the output tape. We say that f is *computable in* AC^1 if the language $\{x01^i \mid \text{the } i\text{-th bit of } f(x) \text{ is } 1\}$ belongs to AC^1. The main difficulty in proving Theorem 2.1 is the following:

▶ **Lemma 4.1.** *Given a non-empty NFA $\mathcal{A}$, one can compute in AC^1 an ideal I such that (i) $I \subseteq L(\mathcal{A})\downarrow$ and (ii) $L(\mathcal{A})$ is directed if and only if $L(\mathcal{A})\downarrow \subseteq I$. Moreover, for every fixed alphabet size, this computation can be carried out in* NL.

Since the inclusion $L(\mathcal{A})\downarrow \subseteq I$ for a given NFA $\mathcal{A}$ and ideal I can be decided in NL [50], Lemma 4.1 immediately implies the upper bounds: Just compute I and check $L(\mathcal{A})\downarrow \subseteq I$.

Let us briefly outline the proof of Lemma 4.1. Given an NFA $\mathcal{A}$ for a regular language L, we first construct an NFA $\mathcal{R}^{\mathtt{idl}}$ accepting representations ideals in an ideal decomposition of $L\downarrow$. This is then transformed into an NFA $\mathcal{R}^{\mathtt{red}}$ that accepts *reduced* ideal representations. Reducedness of the ideal representations enables us to efficiently compute a maximal ideal in $L(\mathcal{R}^{\mathtt{red}})$, by solving a maximum weight path problem. This step can be carried out in AC^1 for arbitrary alphabets and in NL for fixed alphabets. Figure 1 depicts the mentioned transitionary automata and serves as a running example throughout the section.

4.1 Computing the ideal automaton $\mathcal{R}^{\mathtt{idl}}$

Our first step towards Lemma 4.1 is to transform the input NFA into one that is partially ordered. Here, an NFA is *partially ordered* if the state set Q is equipped with a partial order $(Q, \leq)$ such that for every transition from p to q, we have $p \leq q$. In particular, the automaton does not contain any cycles except for self-loops. The following is a standard fact:

▶ **Lemma 4.2.** *Given any NFA $\mathcal{A}$, one can compute in* NL *a partially ordered NFA $\mathcal{R}$ such that $L(\mathcal{R}) = L(\mathcal{A})\downarrow$.*

Essentially, one collapses each strongly connected component (SCC) C of $\mathcal{A}$ into a new state q_C of $\mathcal{R}$ and adds a self-loop to q_C for each letter that appears in C. Here, we require non-determinism in our logspace computation, because we need to determine whether a given letter appears in a strongly connected component. See [27, App. B] for details.

Next, we want to construct an NFA $\mathcal{R}^{\mathtt{idl}}$ over a finite alphabet of *atoms of* Σ, that will accept (as its *words*) the ideal representations given by the accepted paths of $\mathcal{R}$.

▶ **Lemma 4.3.** *Given any partially ordered NFA $\mathcal{R}$ on the finite alphabet Σ, one can compute in* NL *an acyclic NFA $\mathcal{R}^{\mathtt{idl}}$ over some polynomial-sized alphabet $\Gamma \subseteq \mathtt{atoms}(\Sigma)$ such that $\mathtt{Idl}(L(\mathcal{R}^{\mathtt{idl}})) = L(\mathcal{R})\downarrow$.*

Since the only cycles $\mathcal{R}$ contains are self loops, we can write $L(\mathcal{R})$ as the finite union,

$$L(\mathcal{R}) = \bigcup_{i\in[1,r]} a_{1,i}\Delta_{1,i}^* a_{2,i}\Delta_{2,i}^* \cdots a_{k_i,i}\Delta_{k_i,i}^* \text{ with } a_{n,i} \in \Sigma\cup\{\varepsilon\} \text{ and } \Delta_{n,i} \subseteq \Sigma \text{ for } n \in [1,k_i]$$

Since $L(\mathcal{R})$ is downward closed, it is equivalent to the following ideal decomposition of $L(\mathcal{A})\downarrow$:

$$L(\mathcal{R}) = \bigcup_{i\in[1,r]} \{\varepsilon, a_{1,i}\}\Delta_{1,i}^*\{\varepsilon, a_{2,i}\}\Delta_{2,i}^* \cdots \{\varepsilon, a_{k_i,i}\}\Delta_{k_i,i}^* \tag{1}$$

Proof sketch. To construct $\mathcal{R}^{\mathtt{idl}}$ from $\mathcal{R}$, for each state q in $\mathcal{R}$, we add two copies q and q' to $\mathcal{R}^{\mathtt{idl}}$. We keep the initial state the same, and make the final states of $\mathcal{R}^{\mathtt{idl}}$ the copies of final states of $\mathcal{R}$. Each state q with self-loops is turned into a transition reading an alphabet atom $q \xrightarrow{\Delta^{\circledast}} q'$ where Δ contains all letters read on self-loops on q. Furthermore, each transition $p \xrightarrow{a} q$ in $\mathcal{R}$ where $p \neq q$ is turned into a transition reading a single atom $p' \xrightarrow{a^{\textcircled{?}}} q$.

It is easily verified $L(\mathcal{R}^{\mathtt{idl}})$ is the ideal decomposition of $L(\mathcal{R})$ given in equation (1). ◀

4.2 Weighting functions for ideals

Lemma 4.3 tells us that for our given NFA $\mathcal{A}$, we can construct an NFA $\mathcal{R}^{\mathtt{idl}}$ over $\mathtt{atoms}(\Sigma)$ that is acyclic and whose paths correspond to the ideals of $L(\mathcal{A})\downarrow$. Observe that $L(\mathcal{A})$ is directed iff there exists a path π in $\mathcal{R}^{\mathtt{idl}}$ such that $L(\mathcal{A})$ is included in the ideal of π: Clearly, if there is such a path, then $L(\mathcal{A})\downarrow$ must equal this ideal and is thus directed. Conversely, if $L(\mathcal{A})$ is directed and $L(\mathcal{A})\downarrow = I_1 \cup \cdots \cup I_n$, where $I_1, \ldots, I_n$ are the ideals of the paths in $\mathcal{R}^{\mathtt{idl}}$, then directedness implies that $L(\mathcal{A})\downarrow$ is an ideal. By Lemma 3.2, we must have that $L(\mathcal{A})\downarrow$ coincides with some I_i for $i \in [1,n]$, which lies on some path π. Therefore, to show Lemma 4.1, it remains to pick a path π such that if there is a greatest ideal among the paths in $\mathcal{R}^{\mathtt{idl}}$, then it must lie on π. This is the main challenge in our decision procedures.

Our key insight is that this can be accomplished by a *weighting function*. Roughly speaking, we construct a function μ from the set of ideal representations to $\mathbb{N}$ such that μ is *strictly monotone*, meaning (i) if $\mathtt{Idl}(\alpha_1 \cdots \alpha_n) \subseteq \mathtt{Idl}(\beta_1 \cdots \beta_m)$, then $\mu(\alpha_1 \cdots \alpha_n) \leq \mu(\beta_1 \cdots \beta_m)$ and (ii) if in addition $\mathtt{Idl}(\alpha_1 \cdots \alpha_n) \neq \mathtt{Idl}(\beta_1 \cdots \beta_m)$, then $\mu(\alpha_1 \cdots \alpha_n) < \mu(\beta_1 \cdots \beta_m)$. Moreover, the function will be *additive*, meaning $\mu(\alpha_1 \cdots \alpha_n) = \mu(\alpha_1) + \cdots + \mu(\alpha_n)$. Given such a function, we can find the aforementioned path π by picking one with maximal weight.

The weight function. Strictly speaking, an additive strictly monotone function as above is impossible: It would imply $n \leq \mu(a^{\textcircled{?}} \cdots a^{\textcircled{?}}) < \mu(\{a\}^{\circledast})$ for every n (here, the product $a^{\textcircled{?}} \cdots a^{\textcircled{?}}$ has n factors). Therefore, we will only satisfy strict monotonicity on ideal representations of some maximal length k. Given $k \in \mathbb{N}$, the *(k-)weight* of an ideal representation $\alpha_1 \cdots \alpha_n$ is defined as $\mu_k(\alpha_1 \cdots \alpha_n) = \sum_{i=1}^{n} \mu_k(\alpha_i)$ where for each atom α:

$$\mu_k(\alpha) = \begin{cases} 1, & \text{if } \alpha \text{ is a single atom} \\ (k+1)^{|\Delta|}, & \text{if } \alpha = \Delta^{\circledast} \text{ for some } \Delta \subseteq \Sigma \text{ where } \Delta \neq \emptyset. \end{cases} \tag{2}$$

The function μ_k is clearly additive. However, it is not strictly monotone: For instance, the products $a^{\circledast} \cdot a^{\circledast}$ and $a^{\circledast} \cdot b^{\circledast}$ receive the same weight, but the former represents a strict subset of the latter. In the remainder of this subsection, we will show that μ_k is strictly monotone on reduced ideal representations.

Exponential weights are needed. Before we continue with our algorithm for directedness, let us quickly remark that μ_k cannot be chosen much smaller. If the alphabet Σ is not fixed, then μ_k can have exponential values, because $|\Delta|$ appears in the exponent. In fact, dealing with exponentially large numbers is the reason our upper bound in the general NFA case is AC^1 rather than NL. This raises the question of whether there is a weighting function that is strictly monotone on reduced ideal representations of length k with polynomial values. This is not the case. To see this, consider the exponential-length chain C_ℓ of ideals over the alphabet $\Sigma = \{a_0, a_1, \ldots, a_\ell\}$ constructed as follows. For $i \in [1, \ell]$, let $\Sigma_i = \{a_0, \ldots, a_i\}$. Set $C_0 := (a_0{}^{\textcircled{?}})$. For each $i \in [1, \ell]$, assuming $C_{i-1} = (I_1, \ldots, I_t)$, define C_i as

$$C_i := (I_1, \ldots, I_t, (\Sigma_{i-1})^{\circledast} \cdot a_i{}^{\textcircled{?}} \cdot I_1, \ldots, (\Sigma_{i-1})^{\circledast} \cdot a_i{}^{\textcircled{?}} \cdot I_t).$$

Clearly, the number of ideals in each chain C_ℓ is exponential in ℓ. Moreover, the maximal length k of ideals in C_ℓ is polynomial in ℓ. Since for each i, a_i does not appear in C_{i-1}, we know that (a) the ideal representations in C_ℓ are reduced and (b) the chain C_ℓ is strict. Therefore, any weight function that is strictly monotone on ideal representations of length k maps each ideal in C_ℓ to a distinct value, requiring exponentially high values.

Strict monotonicity. We now prove our strict monotonicity property for μ_k:

▶ **Proposition 4.4.** *Let $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_m$ be reduced ideal representations of ideals I and J, respectively, with $m, n \leq k$. If $I \subseteq J$ then $\mu_k(\alpha_1 \cdots \alpha_n) \leq \mu_k(\beta_1 \cdots \beta_m)$. Moreover, if $I \subsetneq J$, then $\mu_k(\alpha_1 \cdots \alpha_n) < \mu_k(\beta_1 \cdots \beta_m)$.*

To prove Proposition 4.4, we use Lemma 4.5, which roughly states that inclusion of ideals behaves similarly to the subword ordering: Inclusion is witnessed by some embedding map.

▶ **Lemma 4.5.** *Let $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_m$ be representations of ideals I and J on Σ, respectively. If $I \subseteq J$, then there exists a function $f\colon [1, n] \to [1, m]$ such that*
1. *$f(i) \leq f(i+1)$ for all $i \in [1, n-1]$,*
2. *α_i is contained in $\beta_{f(i)}$ for all $i \in [1, n]$,*
3. *if β_j is a single atom, then $|f^{-1}(j)| \leq 1$*

Proof sketch. For each atom α, we generate a unique word w_α. If $\alpha = a^{\textcircled{?}}$, then $w_\alpha = a$. If $\alpha = \Delta^{\circledast}$, then we fix an order on Σ and for each $\Delta \subseteq \Sigma$ let $\mathfrak{w}_\Delta$ be a word that contains each letter in Δ once, in the increasing order and set $w_\alpha = \mathfrak{w}_\Delta^{m+1}$. We define f so that it sends each i to the j for which $\beta_1 \cdots \beta_j$ is the shortest prefix for which $w_{\alpha_1} \cdots w_{\alpha_i} \in \mathtt{Idl}(\beta_1 \cdots \beta_j)$. Then f satisfies the premises of Item 1-3. Details can be found in [27, App. B]. ◀

Proof of Proposition 4.4. For the given ideal representations $\alpha_1 \cdots \alpha_n$ and $\beta_1 \cdots \beta_m$, let $f\colon [1,n] \to [1,m]$ be the embedding function introduced in Lemma 4.5.

▷ Claim. For all $j \in [1,m]$, $\mu_k(\beta_j) \geq \sum_{i \in f^{-1}(j)} \mu_k(\alpha_i)$

Proof of claim. If β_j is a single atom, then by Item 3, there exists at most one α_i embedded in β_j and by Item 2, β_j contains α_i; thus $\alpha_i = \beta_j$ is a single atom. In this case, $\mu_k(\beta_j) = \mu_k(\alpha_i) = 1$. Otherwise, $\beta_j = \Delta^{\circledast}$. By Item 1, the elements of $f^{-1}(j)$ have to be consecutive numbers, i.e. $f^{-1}(j) = [i_1, i_2]$. Then f embeds $\alpha_{i_1}, \ldots, \alpha_{i_2}$ in β_j. Since $\alpha_1 \cdots \alpha_n$ is reduced, each pair α_i, α_{i+1} is non-absorptive. Since they are all contained in β_j, either $|f^{-1}(j)| = 1$, or for all $i \in [i_1, i_2]$, $|\alpha_i| < |\beta_j|$ where $|\alpha_i| = 0$ if α_i is a single atom; otherwise it is the size of the alphabet of α_i. In the case $|f^{-1}(j)| = 1$, since the only atom in $f^{-1}(j)$ is contained in β_j, the claim trivially holds. In the latter case, the inequality (3)

$$\sum_{i \in f^{-1}(j)} \mu_k(\alpha_i) \leq n \cdot (k+1)^{|\Delta|-1} < (k+1)^{|\Delta|} = \mu_k(\beta_j) \tag{3}$$

follows from the fact that f can embed at most n many atoms into β_j and that $n \leq k$. ◁

$\mu_k(\beta_1 \cdots \beta_m) \geq \mu_k(\alpha_1 \cdots \alpha_n)$ follows from the claim due to the weight of an ideal representation being defined additively. This concludes the first part of the proof.

Assume $I \subsetneq J$. Then there exists an α_i strictly contained in $\beta_{f(i)}$, or f is not surjective. In the latter case, equation (4) follows from our previous argument.

$$\mu_k(\alpha_1 \cdots \alpha_n) < \mu_k(\beta_1 \cdots \beta_m) \tag{4}$$

In the former case, $\beta_{f(i)}$ is an alphabet atom, otherwise it cannot strictly contain α_i. If α_i is a single atom, (4) follows from $\mu_k(\alpha_i) = 1$. If it is an alphabet atom, (4) follows from (3). ◀

4.3 Reducing ideals

We will apply the weighting function with k being an upper bound on the path length in $\mathcal{R}^{\texttt{idl}}$ (e.g. the number of states). We have seen that the weighting function is strictly monotone on *reduced* ideal representations. Therefore, if all paths in $\mathcal{R}^{\texttt{idl}}$ had reduced ideals, we could prove Lemma 4.1 by picking the path with the largest weight. This is because, if $I_1, \ldots, I_n$ are the ideals on paths of $\mathcal{R}^{\texttt{idl}}$ and I_m has maximal μ_k among them, then Proposition 4.4 implies that $L(\mathcal{A})$ is directed if and only if $L(\mathcal{A}) \subseteq I_m$: Here, the "if" is trivial. Conversely, if $L(\mathcal{A})$ is directed, then $L(\mathcal{A})\!\downarrow = I_1 \cup \cdots \cup I_n$ is an ideal and hence $I_1 \cup \cdots \cup I_n = I_i$ for some i by Lemma 3.2. But then we must have $I_i = I_m$, because $I_m \subseteq I_i$ by the choice of I_i, and if I_i were a strict superset of I_m, $\mu_k(I_m)$ would not be maximal. Thus, $L(\mathcal{A}) \subseteq I_m$.

Thus, our next task is to transform $\mathcal{R}^{\texttt{idl}}$ so as to make all ideal representations reduced:

▶ **Lemma 4.6.** *Given any partially ordered NFA $\mathcal{R}$ on the finite alphabet Σ, one can compute in* NL *an acyclic NFA $\mathcal{R}^{\texttt{red}}$ over some polynomial-sized alphabet $\Gamma \subseteq$* $\texttt{atoms}(\Sigma)$ *such that* $\texttt{Idl}(L(\mathcal{R}^{\texttt{red}})) = L(\mathcal{R})\!\downarrow$ *and the ideal representations $\mathcal{R}^{\texttt{red}}$ accepts are reduced.*

Reducing an individual ideal representation is easy: repeatedly merge consecutive atoms, as briefly sketched in Lemma 3.1. Reducing *all* ideal representations accepted by an NFA at the same time is not obvious: For example, we cannot just merge two transitions $p \xrightarrow{\{a\}^{\circledast}} q \xrightarrow{a^{?}} r$, since each of them might be needed for other paths. We achieve this using transducers.

Transducers. A transducer is a tuple $\mathcal{T} = \langle Q, \Gamma^i, \Gamma^o, t^0, F, E\rangle$ where Q is a finite set of states, Γ^i and Γ^o are finite (input and output) alphabets, $t^0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $E \subseteq Q \times (\Gamma^i \cup \{\varepsilon\}) \times (\Gamma^o \cup \{\varepsilon\}) \times Q$ is the transition relation. Each transition, reads a letter (or ε) from the input alphabet, writes a letter (or ε) from the output alphabet and moves to a new state. A sequence $r = (q_1, a_1, b_1, q_2)(q_2, a_2, b_2, q_3) \cdots (q_m, a_m, b_m, q_{m+1})$ is called a run of $\mathcal{T}$ if each (q_i, a_i, b_i, q_{i+1}) is in E for $i \in [1, m]$ with $t^0 = q_1$ and $q_{m+1} \in F$. For such a run, let the projection of the transitions to the input (similarly, output) alphabet be denoted by $\mathrm{inp}(r)$ (similarly, $\mathrm{out}(r)$). That is, for the r defined above $\mathrm{inp}(r)$ is the subword of $a_1 a_2 \ldots a_m$ and $\mathrm{out}(r)$ is the subword of $b_1 b_2 \ldots b_m$. Then, the set of $(\mathrm{inp}(r), \mathrm{out}(r))$ over runs r of $\mathcal{T}$ is called the language of $\mathcal{T}$, is denoted by $L(\mathcal{T})$, and defines a rational relation on $\Gamma^i \times \Gamma^o$. For a set $Y \subseteq (\Gamma^i)^*$, $\mathcal{T}(Y)$ denotes the set of words $\mathcal{T}$ outputs upon a run from Y, i.e. $\mathcal{T}(Y) = \{b \mid a \in Y \text{ and } (a, b) \in L(\mathcal{T})\}$.

Composition of two transducers is again a transducer [15, 44].

Left- and right-reduced representations. We will later define two transducers $\mathcal{T}_{\mathcal{L}}$ and $\mathcal{T}_{\mathcal{R}}$ the composition of which will take an ideal representation $\alpha_1 \cdots \alpha_n$ produced by $\mathcal{R}^{\mathtt{idl}}$ and return an equivalent reduced ideal representation $\beta_1 \cdots \beta_m$ with $m \leq n$. In particular, $\mathcal{T}_{\mathcal{L}}$ will turn $\alpha_1 \cdots \alpha_n$ into an equivalent ideal representation that is *left-reduced*, and $\mathcal{T}_{\mathcal{R}}$ will turn it into an equivalent ideal representation that is *right-reduced. Left-* and *right-reducedness* are defined as follows. An ideal representation $\alpha_1 \cdots \alpha_n$ is called *left-reduced* if for all $i \in [1, n-1]$, α_i does not absorb α_{i+1}. Similarly, it is called *right-reduced* if α_{i+1} does not absorb α_i. Clearly, if a representation is both left- and right-reduced, then it is reduced.

Building the transducers. Intuitively, the transducer $\mathcal{T}_{\mathcal{L}}$ scans an ideal representation and after reading its first alphabet atom $\Delta^{\circledast}$, it outputs $\Delta^{\circledast}$, but then skips (i.e. reads without producing output) all atoms that are absorbed by $\Delta^{\circledast}$. Formally, we have $\mathcal{T}_{\mathcal{L}} = \langle Q_{\mathcal{L}}, \Gamma^i_{\mathcal{L}}, \Gamma^o_{\mathcal{L}}, t^0_{\mathcal{L}}, F_{\mathcal{L}}, E_{\mathcal{L}}\rangle$. $Q_{\mathcal{L}}$ contains a state corresponding to each alphabet atom in $\Gamma \subseteq \mathtt{atoms}(\Sigma)$ (as given in Lemma 4.3), with a new initial state $t^0_{\mathcal{L}}$ and a state t_1 for all of the single atoms. We set $\Gamma^i_{\mathcal{L}} = \Gamma^o_{\mathcal{L}} = \Gamma$. Let Φ be a mapping from Γ to $Q_{\mathcal{L}}$, which sends each alphabet atom to its corresponding state, and each single atom to t_1. $F_{\mathcal{L}} = Q_{\mathcal{L}} \setminus \{t^0_{\mathcal{L}}\}$ and $E_{\mathcal{L}}$ is defined as follows;

- For all $\alpha \in \Gamma$, $(t^0_{\mathcal{L}}, \alpha, \alpha, \Phi(\alpha))$ and $(t_1, \alpha, \alpha, \Phi(\alpha))$ are in $E_{\mathcal{L}}$,
- For each $t \in Q_{\mathcal{L}} \setminus \{t^0_{\mathcal{L}}, t_1\}$ and each atom $\alpha \in \Gamma$, if t (as an alphabet atom) does not absorb α (as an atom), then $(t, \alpha, \alpha, \Phi(\alpha))$ is in $E_{\mathcal{L}}$.
- For each $t \in Q_{\mathcal{L}} \setminus \{t^0_{\mathcal{L}}, t_1\}$ and each atom $\alpha \in \Gamma$, if t absorbs α, then $(t, \alpha, \varepsilon, t)$ is in $E_{\mathcal{L}}$.

It is easy to see that for an ideal representation $\alpha_1 \cdots \alpha_n$, $\mathcal{T}_{\mathcal{L}}(\alpha_1 \cdots \alpha_n)$ is left-reduced and represents the same ideal. Clearly the size of $Q_{\mathcal{L}}$, as well as the sizes of the alphabets is polynomial. Furthermore, for a word w and all $(\mathrm{inp}(w), \mathrm{out}(w))$, $|\mathrm{out}(w)| \leq |\mathrm{inp}(w)|$.

Reversing the edges and flipping the initial and final states of $\mathcal{T}_{\mathcal{L}}$ we obtain the reverse transducer $\mathcal{T}_{\mathcal{R}}$. It can be inductively shown that for any ideal representation $\alpha_1 \cdots \alpha_n$, $\mathcal{T}_{\mathcal{R}}(\alpha_1 \cdots \alpha_n)$ is right-reduced. To show Lemma 4.6, we will apply the composition $\mathcal{T}_{\mathcal{L}} \circ \mathcal{T}_{\mathcal{R}}$ to $L(\mathcal{R}^{\mathtt{idl}})$. Here, we need to show that applying $\mathcal{T}_{\mathcal{L}}$ after $\mathcal{T}_{\mathcal{R}}$ does not spoil right-reducedness:

▶ **Lemma 4.7.** *If $\alpha_1 \cdots \alpha_n$ is right-reduced, then $\mathcal{T}_{\mathcal{L}}(\alpha_1 \cdots \alpha_n)$ is also right-reduced.*

Proof. Since $\alpha_1 \cdots \alpha_n$ is right-reduced, for $i \in [1, n]$, α_{i+1} does not absorb α_i. By construction, $\mathcal{T}_{\mathcal{L}}(\alpha_1 \cdots \alpha_n)$ is a subword of $\alpha_1 \cdots \alpha_n$, say $\alpha_{i_1} \cdots \alpha_{i_k}$. We show that for all $j \in [1, k]$, $\alpha_{i_{j+1}}$ does not absorb α_{i_j}. Let $i_j = k$ and $i_{j+1} = k'$. If $k' = k+1$, then the claim follows from the right-reducedness of $\alpha_1 \ldots \alpha_n$. Otherwise, α_k absorbs all atoms between itself and $\alpha_{k'}$. In particular it absorbs $\alpha_{k'-1}$. Since $\alpha_{k'}$ does not absorb $\alpha_{k'-1}$, it cannot absorb α_k. ◀

Thus, by applying $\mathcal{T}_{\mathcal{L}} \circ \mathcal{T}_{\mathcal{R}}$ to $L(\mathcal{R}^{\mathtt{idl}})$, we obtain an NFA that reads ideal representations of the same set of ideals (i.e. $L(\mathcal{A})\downarrow$) and every ideal representation is reduced. The resulting NFA $\mathcal{R}^{\mathtt{red}}$ can be computed in NL. For details of the construction, see [27, Corollary B.5].

4.4 Deciding directedness

We now present our algorithms to decide directedness for a given NFA. We first complete the proof of Lemma 4.1. For this, in light of Lemma 4.6 and Proposition 4.4, it remains to compute a path of $\mathcal{R}^{\mathtt{red}}$ of maximal weight.

Computing the maximal weight. It is well known that the maximum weight path problem can be reduced to matrix multiplication over the max-plus semiring [3, Lemma 5.11]. This yields AC^1 (resp. NL) algorithms for binary (unary) encoded weights [22]. For completeness sake we provide a proof of this fact. Let $\mathcal{R}^{\mathtt{red}} = (Q^{\mathtt{red}}, \Gamma, \delta^{\mathtt{red}}, {q_0}^{\mathtt{red}}, F^{\mathtt{red}})$ be the NFA from Lemma 4.6. We may assume $\mathcal{R}^{\mathtt{red}}$ has a unique final state ${q_f}^{\mathtt{red}}$, is acyclic except for an ε self-loop in ${q_f}^{\mathtt{red}}$, and between each pair of states, there is at most one edge. The goal is to compute, for each state q, the maximal weight of any path starting in q.

Let $m = |Q^{\mathtt{red}}|$. Since $\mathcal{R}^{\mathtt{red}}$ is acyclic apart from the self-loop on ${q_f}^{\mathtt{red}}$, any ideal representation accepted by $\mathcal{R}^{\mathtt{red}}$ has length $\leq m$. Observe that due to the ε-loop on ${q_f}^{\mathtt{red}}$, every ideal $\alpha_1 \cdots \alpha_n \in L(\mathcal{R}^{\mathtt{red}})$ is read on some path of length *exactly* m. We want to find an accepted path with the maximum summation of m-weights of each transition (recall that $\mu_m(\varepsilon) = 0$). To do so, we fix an order $\{q_1, \ldots, q_m\}$ on the states of $Q^{\mathtt{red}}$ such that $q_1 := {q_0}^{\mathtt{red}}$ and $q_m := {q_f}^{\mathtt{red}}$ and construct a $m \times m$-matrix $\mathcal{M}$, the elements of which takes values from the *max-plus semiring* $(\mathbb{N} \cup \{-\infty\}, +, \max, 0, -\infty)$. For each $i, j \in [1, m]$, we set $\mathcal{M}(i, j)$ to (i) $\mu_m(x)$, if there is an edge $(q_i, x, q_j) \in \delta^{\mathtt{red}}$ with $x \in \mathtt{atoms}(\Sigma) \cup \{\varepsilon\}$, (ii) $-\infty$, otherwise. We can now apply the standard fact from weighted automata that for every $n \geq 0$, in the matrix power $\mathcal{M}^n$, the entry (i, j) is the maximum weight of all paths of length exactly n from q_i to q_j [3]. Therefore, the largest weight among all paths from q_s to q_m is the entry (s, m) in the matrix power $\mathcal{M}^m$. A single matrix product can be computed in AC^0 since binary addition and the maximum of multiple numbers can be computed in AC^0. Moreover, for n given in unary, a matrix power $\mathcal{M}^n$ can be computed in AC^1 by repeated squaring: One writes $n = \sum_{i=0}^{\ell} b_i 2^i$ with $b_0, \ldots, b_\ell \in \{0, 1\}$ and computes $\mathcal{M}'_0 = \mathcal{M}^{b_\ell}$, $\mathcal{M}'_i = (\mathcal{M}'_{i-1})^2 \cdot \mathcal{M}^{b_{\ell-i}}$, yielding $\mathcal{M}^n = \mathcal{M}'_\ell$. Thus, by applying the (constant depth) AC^0 circuit for matrix multiplication $\ell \in O(\log n)$ times, we obtain a circuit of logarithmic depth. In particular, we can compute $\mathcal{M}^m$, and hence the maximal path weights $\mathcal{M}^m(s, m)$, in AC^1.

In case the alphabet is fixed, we compute the maximal weights in NL. Observe that in this case, all weights $\mu_m(\alpha)$ for atoms $\alpha \in \mathtt{atoms}(\Sigma)$ have $\mu_m(\alpha) \leq (m+1)^{|\Sigma|}$, which is polynomial as $|\Sigma|$ is constant. In particular, all the maximal path weights from q_s to q_m, are bounded by $m \cdot (m+1)^{|\Sigma|}$ and can be stored in logarithmic space. Thus we can proceed as follows. Given $s \in [1, m]$, for every $\ell = m \cdot (m+1)^{|\Sigma|}, \ldots, 0$, we decide in NL whether there exists a path of weight ℓ from q_s to q_m. If so, then ℓ is the maximal weight. Since $\mathsf{NL} = \mathsf{coNL}$, we can also determine the non-existence of such a path and continue with $\ell - 1$.

Computing a maximal-weight ideal representation. We have now computed, for each s, the maximal weight M_s of any path from q_s to the final state q_m. For Lemma 4.1, we now need to compute in NL a path from q_1 to q_m of maximal weight. Here, it is important that this computation only depends on the input (even though our NL computation is non-deterministic). Starting from q_1, we successively compute the next transition in our path. If q_i is the current state, then we compute the next state q_j as follows. We compute

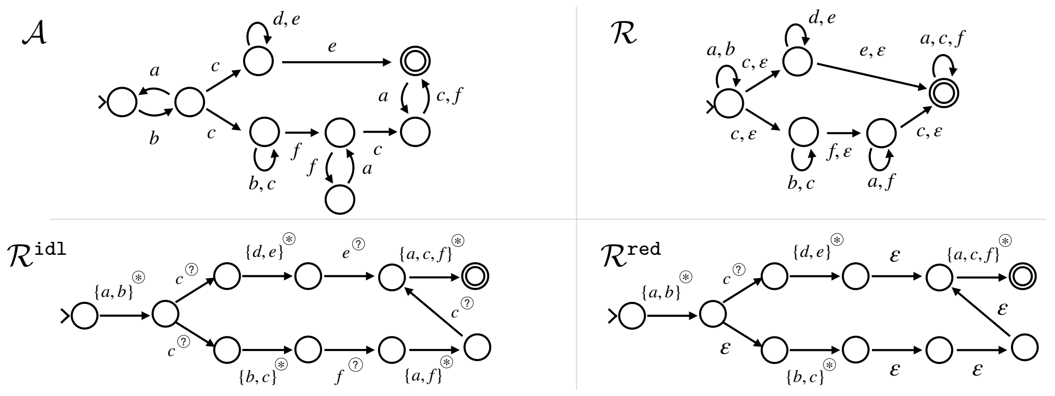

Figure 1 Initial automaton $\mathcal{A}$ on alphabet $\{a, b, c, d, e, f\}$ and the corresponding $\mathcal{R}$, $\mathcal{R}^{\mathtt{idl}}$ and $\mathcal{R}^{\mathtt{red}}$ are depicted. The initial states of the automata are marked with a half arrow sign and the final states are encircled. Since $\mathcal{R}^{\mathtt{red}}$ has 10 states all ideal representations in $L(\mathcal{R}^{\mathtt{red}})$ contain ≤ 10 atoms. Therefore, m is set to 10 and we calculate the maximal 10-weight ideal, which is $I = \mathtt{Idl}(\{a, b\}^{\circledast} \cdot c^{?} \cdot \{d, e\}^{\circledast} \cdot \{a, c, f\}^{\circledast})$ with the 10-weight $11^3 + 2 \cdot 11^2 + 1$. $L(\mathcal{A}) \downarrow \not\subseteq I$ witnessed by $cb \in L(\mathcal{A}) \downarrow \setminus I$; proving that $L(\mathcal{A})$ is not directed.

M as the maximal M_ℓ, where q_ℓ ranges over all states reachable in one step from q_i. Then, we pick the smallest j with $M_j = M$. This way, we successively output a path of maximal weight, such that the path only depends on the input. This completes Lemma 4.1.

Deciding directedness. The upper bounds in Theorems 2.1 and 2.2 follow by applying Lemma 4.1 to obtain an ideal $\alpha_1 \cdots \alpha_n$, either in AC^1 if Σ is part of the input, or in NL for fixed Σ. Finally, checking whether $L(\mathcal{A})\downarrow \subseteq \mathtt{Idl}(\alpha_1 \cdots \alpha_n)$ can be done in NL [50].

5 Solution on Context-free Languages

We now prove Theorem 2.3. As in Section 4, the upper bound uses the weighting function to compute a candidate ideal in $L(G)\downarrow$. However, the ideal representation may be exponentially long and will thus be compressed by a straight-line program. For the lower bound, the key idea is to employ a construction from [12] to compute a compressed ideal that contains all words of some length N (given in binary), except a particular word specified as an SLP.

5.1 Deciding directedness of context-free languages

We begin with the PSPACE upper bound, which requires some terminology. A *context-free grammar (CFG)* is a tuple $G = \langle N, \Sigma, P, S\rangle$ where N is the finite set of *nonterminals*, Σ is the finite set of *terminals*, or the finite *alphabet*, $S \in N$ is the *start nonterminal* and $P \subseteq N \times (N \cup \Sigma)^*$ is the finite set of *productions*. We use the arrow notation to denote productions. $A \to w$ denotes $(A, w) \in P$. We write $w \to^* w'$ for some $w, w' \in (N \cup \Sigma)^*$ to express that w' can be produced by w through a finite sequence of productions.

For $w \in (N \cup \Sigma)^*$, we denote by $L(w) = \{w' \in \Sigma^* \mid w \to^* w'\}$ all sequences of terminals w can produce and call it the *language of* w. We define the language of a grammar to be the language of its start nonterminal. That is, $L(G) := L(S)$. WLOG we assume that all nonterminals are reachable from S. A CFG is said to be in *Chomsky Normal Form (CNF)*, if all its productions are of the form $A \to BC$, $A \to a$ or $S \to \varepsilon$, where $A, B, C, S \in N$, $a \in \Sigma$ and $B, C \neq S$. It is well known that one can bring a given grammar into CNF in polynomial time. A CFG G is called *acyclic*, if non of its nonterminals produce itself.

A *straight line program (SLP)* is a CFG that produces a single word. Formally, an SLP is a CFG $G = \langle N, \Sigma, P, S\rangle$ where (i) for each $A \in N$, there is exactly one production $A \to w$ in P, (ii) G is acyclic. We denote the unique word $a_1 \cdots a_n$ an SLP $\mathbb{A}$ produces by $\mathrm{val}(\mathbb{A})$. If the letters of $\mathbb{A}$ belong to $\mathtt{atoms}(\Sigma)$, $\mathrm{val}(\mathbb{A})$ is an ideal representation over Σ. Thus $\mathbb{A}$ is a *compressed ideal representation for* $I = \mathtt{Idl}(\mathrm{val}(\mathbb{A}))$, or shortly *compressed ideal* I.

Our algorithm is analogous to the one for NFAs. First, an analogue of Lemma 4.1:

▶ **Lemma 5.1.** *There is a polynomial time algorithm that given a non-empty CFG G, computes a compressed ideal $I \subseteq L(G){\downarrow}$ such that $L(G){\downarrow}$ is directed if and only if $L(G){\downarrow} \subseteq I$.*

And given Lemma 5.1, it remains to decide whether $L(G) \subseteq I$:

▶ **Lemma 5.2.** *Given any CFG G and a compressed ideal I, one can decide in* PSPACE *whether $L(G) \subseteq I$.*

A grammar of ideals. The remainder of this subsection is devoted to Lemmas 5.1 and 5.2. Analogously to Lemma 4.3, we first transform G into an acyclic grammar $G^{\mathtt{idl}}$ that produces ideal representations of an ideal decomposition of $L(G){\downarrow}$.

▶ **Lemma 5.3.** *Given any CFG G in CNF over Σ, one can compute in polynomial time an acyclic CFG $G^{\mathtt{idl}}$ over a polynomial-sized alphabet $\Gamma \subseteq \mathtt{atoms}(\Sigma)$ with $\mathtt{Idl}(L(G^{\mathtt{idl}})) = L(G){\downarrow}$.*

The procedure is similar to Courcelle's construction [23] (see [27, App. C] for details).

Reducing ideals. The next step is analogous to Lemma 4.6: We want to transform $G^{\mathtt{idl}}$ so as to only produce reduced ideal representations. Luckily, we can directly apply the transducers $\mathcal{T}_{\mathcal{L}}$ and $\mathcal{T}_{\mathcal{R}}$ constructed for Lemma 4.6: Since for a given CFL K and a transducer $\mathcal{T}$, one can compute in polynomial time a grammar for $\mathcal{T}(K)$ [44], we obtain the following:

▶ **Lemma 5.4.** *Given any CFG G in CNF over alphabet Σ, one can compute in polynomial time an acyclic CFG $G^{\mathtt{red}}$ in CNF over some polynomial-sized alphabet $\Gamma \subseteq \mathtt{atoms}(\Sigma)$ such that (i) $\mathtt{Idl}(L(G^{\mathtt{red}})) = L(G){\downarrow}$ and (ii) all ideal representations in $L(G^{\mathtt{red}})$ are reduced.*

Similar to Lemma 4.6, we apply $\mathcal{T}_{\mathcal{L}} \circ \mathcal{T}_{\mathcal{R}}$ to $L(G^{\mathtt{idl}})$ (see [27, Lem. C.1]) and convert to CNF.

Calculating the maximum weight ideal. Similar to Section 4, the next step is to compute for each nonterminal A of $G^{\mathtt{red}}$ the maximal weight of any ideal representation produced by A. Let $G^{\mathtt{red}} = \langle N^{\mathtt{red}}, \Gamma, P^{\mathtt{red}}, S^{\mathtt{red}}\rangle$ denote the grammar from Lemma 5.4. With the same argument as in Section 4, an ideal of maximal weight will be as desired in Lemma 5.1. We use the weighting function μ_m, where $m = 3 \cdot 2^{2|N^{\mathtt{red}}|}$ is an upper bound on the length of words in $G^{\mathtt{red}}$. A notable difference to Section 4 is that here m is exponential.

For each nonterminal A of $G^{\mathtt{red}}$, we denote by $\mu_m(A)$ the maximal possible weight of any ideal representation generated by A. To calculate $\mu_m(A)$ for each A, we employ a simple dynamic programming approach. We maintain a table T that contains for each nonterminal A a number $T(A) \in \mathbb{N}$, which is the maximal weight of a derivable ideal representation observed so far. We initialize $T(A) = -\infty$ for every A. Then, we set $T(A)$ to the maximal value of $\mu_m(a)$, where a ranges over all $a \in \mathtt{atoms}(\Sigma)$ for which $A \to a$ is a production. Finally, we perform the following update step. For each nonterminal A, if there is a production $A \to BC$

such that currently $T(A)$ is smaller than $T(B)+T(C)$, then we update $T(A) := T(B)+T(C)$. It can be shown by induction that after i update steps, $T(A)$ contains the correct value $\mu_m(A)$ for each nonterminal A that has a depth $\leq i$ derivation tree that attains $\mu_m(A)$. When we apply the update step $|N^{\mathtt{red}}|$ times, we arrive at $T(A) = \mu_m(A)$ for every nonterminal A.

Computing the candidate ideal. Given the numbers $\mu_m(A)$, it is easy to prove Lemma 5.1. For each nonterminal A, there must exist a "max-weight" production $A \to BC$, resp. $A \to a$, such that $\mu_m(A) = \mu_m(B) + \mu_m(C)$, resp. $\mu_m(A) = \mu_m(a)$. We build a new grammar $\mathbb{S}$ by selecting for each nonterminal A of $G^{\mathtt{red}}$ this max-weight production. Then $\mathbb{S}$ contains at most one production for each nonterminal and is thus an SLP. Moreover, it clearly generates an ideal representation $\mathrm{val}(\mathbb{S})$ of maximal weight. We only need to argue that $L(G)$ is directed iff $L(G) \subseteq \mathtt{Idl}(\mathrm{val}(\mathbb{S}))$. As before, the "if" direction is obvious, because $L(G) \subseteq \mathtt{Idl}(\mathrm{val}(\mathbb{S}))$ implies that $L(G)$ is an ideal. Conversely, suppose $L(G)$ is directed and let $L(G)\!\downarrow = I_1 \cup \cdots \cup I_n$ be the ideal decomposition given by $L(G^{\mathtt{red}})$ with $\mathtt{Idl}(\mathrm{val}(\mathbb{S})) = I_i$. Since $L(G)$ is directed, $L(G)\!\downarrow$ is an ideal and thus $I_1 \cup \cdots \cup I_n = I_j$ for some j by Lemma 3.2. In particular, we have $I_i \subseteq I_j$. Moreover, if the inclusion were strict, I_i would not have maximal weight. Hence, $I_i = I_j$ and thus $L(G) \subseteq I_i = \mathtt{Idl}(\mathrm{val}(\mathbb{S}))$ as required.

Deciding directedness. With Lemma 5.1 in hand, it remains to prove Lemma 5.2. Suppose we are given a grammar G and an SLP $\mathbb{S}$ for $I = \mathtt{Idl}(\mathrm{val}(\mathbb{S}))$, and we want to check $L(G) \subseteq \mathrm{val}(\mathbb{S})$. Since this is equivalent to $L(G)\!\downarrow \subseteq \mathrm{val}(\mathbb{S})$, we first construct $G^{\mathtt{red}}$ given in Lemma 5.4. Recall that $L(G^{\mathtt{red}})$ generates representations of ideals of $L(G)\!\downarrow$. The algorithm guesses an ideal representation in $L(G^{\mathtt{red}})$ whose ideal *does not embed in I*.

We guess an ideal representation generated by $G^{\mathtt{red}}$, atom by atom, via its leftmost derivation. This word can be exponentially long, but we only store one (polynomial-length) path in the derivation tree, leading to the terminal atom that we are currently guessing (see [27, App.D] for an example). While guessing the representation, we simultaneously maintain a (binary encoded) pointer into $\mathrm{val}(\mathbb{S})$. Suppose $\alpha_1 \cdots \alpha_{j-1}$ is guessed so far. While α_j is being guessed, the pointer holds the length of the shortest prefix of $\mathrm{val}(\mathbb{S})$, $\alpha_1 \cdots \alpha_{j-1}$ embeds in. Let $\mathrm{val}(\mathbb{S})[i]$ denote the i^{th} index of $\mathrm{val}(\mathbb{S})$. If there is an atom $\mathrm{val}(\mathbb{S})[i']$ with $i' \geq i$ (if $\mathrm{val}(\mathbb{S})[i]$ is an alphabet atom) or $i' > i$ (if $\mathrm{val}(\mathbb{S})[i]$ is a single atom) that α_j embeds in, we update the pointer to the smallest such i'. If there is no such atom, the guessed ideal does not embed in I. On the other hand, if $j-1$ is the last atom guessed, then the guessed ideal embeds in I. Details are in [27]. This establishes Lemma 5.2 and thus Theorem 2.3.

5.2 PSPACE Lower Bound

Let us now come to the lower bound in Theorem 2.3. It remains to show:

▶ **Lemma 5.5.** *Given a CFG G over $\{0,1\}$, directedness of $L(G)$ is* PSPACE*-hard.*

To this end, we reduce from compressed membership in automatic relations. Given two words $u = a_1 \cdots a_n, v = b_1 \cdots b_n \in \{0,1\}^*$, their *convolution* is defined as $u \otimes v = (a_1, b_1) \cdots (a_n, b_n) \in (\{0,1\} \times \{0,1\})^*$. The following was shown in [39, Corollary 8]:

▶ **Lemma 5.6.** *There exists a regular language $R \subseteq (\{0,1\} \times \{0,1\})^*$ such that for given two SLPs $\mathbb{A}$ and $\mathbb{B}$ with $|\mathrm{val}(\mathbb{A})| = |\mathrm{val}(\mathbb{B})|$, deciding $\mathrm{val}(\mathbb{A}) \otimes \mathrm{val}(\mathbb{B}) \in R$ is* PSPACE*-hard.*

From Lemma 5.6, we deduce the following:

▶ **Lemma 5.7.** *Given an SLP $\mathbb{B}$ and a CFG G such that all words in $L(G)$ have length exactly $|\mathrm{val}(\mathbb{B})|$, both over the alphabet $\Sigma = \{0,1\}$, deciding $\mathrm{val}(\mathbb{B}) \in L(G)$ is* PSPACE*-hard.*

Proof. We reduce from the PSPACE-complete problem in Lemma 5.6. Let R be the regular language from Lemma 5.6, and let $\mathbb{A}$ and $\mathbb{B}$ be SLPs with $n = |\operatorname{val}(\mathbb{A})| = |\operatorname{val}(\mathbb{B})|$. Observe that $\operatorname{val}(\mathbb{A}) \otimes \operatorname{val}(\mathbb{B}) \in R$ if and only if $\operatorname{val}(\mathbb{B})$ belongs to the language $K = \{w \in \{0,1\}^n \mid \operatorname{val}(\mathbb{A}) \otimes w \in R\}$. We can construct a context-free grammar G for K in polynomial-time by viewing an automaton for R as a transducer and applying it to the SLP $\mathbb{A}$. ◀

Now in order to reduce the compressed membership problem $\operatorname{val}(\mathbb{B}) \in L(G)$ in Lemma 5.7 to an inclusion $L(G) \subseteq I$, the key trick is to construct an ideal I that acts like a complement of $\{\operatorname{val}(\mathbb{B})\}$. We expect that this will be of independent interest.

▶ **Lemma 5.8.** *Given an SLP $\mathbb{B}$ over Σ, one can construct in polynomial time an SLP $\mathbb{I}$ over $\mathtt{atoms}(\Sigma)$ so that $\mathtt{Idl}(\operatorname{val}(\mathbb{B}))$ is infinite and $\mathtt{Idl}(\operatorname{val}(\mathbb{I})) \cap \Sigma^{|\operatorname{val}(\mathbb{B})|} = \Sigma^{|\operatorname{val}(\mathbb{B})|} \setminus \{\operatorname{val}(\mathbb{B})\}$.*

The proof uses a construction from [12]. The authors of the latter were interested in defining languages in the existential fragment of first-order logic over the structure the set Σ^*, ordered by $\preccurlyeq$. In one step [12, Lemma 3.1], given a word $u \in \Sigma^*$, they construct a word $\bar{w} \in \Sigma^*$ such that $\{\bar{w}\}\!\downarrow \cap\, \Sigma^{|w|} = \Sigma^{|w|} \setminus \{w\}$. To this end, they write $w = a_1 \cdots a_n$ and define u_i to be a word that contains every letter from Σ, except for a_i. Then, they argue that $\bar{w} = u_1 a_1 \cdots u_{n-1} a_{n-1} u_n$ is as desired. Here, we cannot use $\bar{w}$ directly, because we want I to be infinite. However, we can use a similar construction.

Proof of Lemma 5.8. Suppose $\operatorname{val}(\mathbb{B}) = b_1 \cdots b_n$ and define $I = \mathtt{Idl}(w)$ with

$$w = (\Sigma \setminus \{b_1\})^{\circledast} \cdot b_1^{\textcircled{?}} \cdot (\Sigma \setminus \{b_2\})^{\circledast} \cdot b_2^{\textcircled{?}} \cdots (\Sigma \setminus \{b_{n-1}\})^{\circledast} \cdot b_{n-1}^{\textcircled{?}} \cdot (\Sigma \setminus \{b_n\})^{\circledast}.$$

We first show $I \cap \Sigma^n = \Sigma^n \setminus \{\operatorname{val}(\mathbb{B})\}$. Clearly, $b_1 \cdots b_n \notin I$: Let j_i be length of the shortest prefix of w whose ideal contains $b_1 \cdots b_i$. Clearly $j_1 = 2$, since the first atom $(\Sigma \setminus \{b_1\})^{\circledast}$ does not embed b_1. Inductively we get $j_2 = 4, \ldots, j_{n-1} = 2n-2$, which leaves only the last atom $(\Sigma \setminus \{b_n\})^{\circledast}$ to embed b_n, but this is not possible. We now prove $\Sigma^n \setminus \{b_1 \cdots b_n\} \subseteq I$. Let $c_1 \cdots c_n \neq b_1 \cdots b_n$ and choose d minimally with $c_d \neq b_d$. Let h_i to be the length of the of shortest prefix of w whose ideal contains $c_1 \cdots c_i$. Then $h_{d-1} = j_{d-1} = 2d-2$. Since c_d differs from b_d, it embeds in the $(2d-1)$-th atom $(\Sigma \setminus \{b_d\})^{\circledast}$, i.e. $h_d = 2d-1$. The remaining $(n-d)$-length suffix $c_{d+1} \cdots c_n$ embeds in the $2(n-i)$ length suffix $(\Sigma \setminus \{b_d\})^{\circledast} \cdot b_d^{\textcircled{?}} \cdots (\Sigma \setminus \{b_{n-1}\})^{\circledast} \cdot b_{n-1}^{\textcircled{?}}$: Indeed, since $(\Sigma \setminus \{b_k\})^{\circledast} \cdot b_k^{\textcircled{?}}$ embeds every letter, the aforementioned suffix even embeds every word from Σ^{n-d}.

It remains to be shown that we can compute an SLP for w. Note that w is *almost* a homomorphic image of $\operatorname{val}(\mathbb{B})$. Given SLP $\mathbb{B}$, we obtain an SLP $\mathbb{I}'$ for w by replacing each production $A \to b$ with $A \to (\Sigma \setminus \{b\})^{\circledast} b^{\textcircled{?}}$. Then, $\operatorname{val}(\mathbb{I}') = (\Sigma \setminus \{b_1\})^{\circledast} \cdot b_1^{\textcircled{?}} \cdots (\Sigma \setminus \{b_n\})^{\circledast} \cdot b_n^{\textcircled{?}}$. To get w exactly, we construct $\mathbb{I}$ so that $\operatorname{val}(\mathbb{I})$ is $\operatorname{val}(\mathbb{I}')$ without its last letter. It is easy to see that this can be done in polynomial time (it follows, e.g. from [43, Theorem 7.1]). ◀

We are now ready to prove Lemma 5.5. Given a grammar G and an SLP $\mathbb{B}$ as in Lemma 5.7, we use Lemma 5.8 to construct an SLP $\mathbb{I}$ with $\mathtt{Idl}(\mathbb{I}) \cap \Sigma^{|\operatorname{val}(\mathbb{I})|} = \Sigma^{|\operatorname{val}(\mathbb{I})|} \setminus \{\operatorname{val}(\mathbb{B})\}$. Observe that now $\operatorname{val}(\mathbb{B}) \notin L(G)$ if and only if $L(G) \subseteq \mathtt{Idl}(\mathbb{I})$. Moreover, observe that $L(G)$ is finite and $\mathtt{Idl}(\mathbb{I})$ is infinite. Therefore, the following lemma implies that $\operatorname{val}(\mathbb{B}) \notin L(G)$ if and only if the context-free language $L(G) \cup \mathtt{Idl}(\operatorname{val}(\mathbb{I}))$ is directed, yielding PSPACE-hardness.

▶ **Lemma 5.9.** *For finite $L \subseteq \Sigma^*$ and an infinite ideal I, we have $L \subseteq I$ iff $L \cup I$ is directed.*

Proof. Clearly, if $L \subseteq I$, then $L \cup I = I$ is an ideal and thus directed. Conversely, suppose $L \cup I$ is directed. Consider an ideal decomposition $L{\downarrow} = I_1 \cup \ldots \cup I_n$. Here, all I_i are finite since L is finite. Since $L \cup I$ is directed, the downward closure $(L \cup I){\downarrow}$ must coincide with one of the ideals in the ideal decomposition $(L \cup I){\downarrow} = I_1 \cup \ldots \cup I_n \cup I$. Since I is the only infinite ideal, this is only possible with $(L \cup I){\downarrow} = I$. In particular, $L \subseteq I$. ◀

6 Downward closure comparison

Regular languages. We now show how to obtain Theorem 2.4 as a byproduct of our results. For the upper bounds, we use Lemma 4.1 to compute, in AC^1 resp. NL, a candidate ideal I_i for each input language L_i. Since L_1 and L_2 are directed, we must have $L_i{\downarrow} = I_i$ and we can decide in deterministic logspace whether $I_1 = I_2$ [50]. This yields an NL upper bound for fixed alphabets and an AC^1 upper bound for arbitrary alphabets.

For the NL lower bound, we reduce from emptiness of NFAs: Given an NFA $\mathcal{A}$, we may assume that all transitions are labeled with the empty word ε. We take an NFA $\mathcal{A}'$ that just accepts $\{\varepsilon\}$. Then $L(\mathcal{A}) \neq \emptyset$ if and only if $L(\mathcal{A}'){\downarrow} = L(\mathcal{A}){\downarrow}$, proving NL-hardness.

Context-free languages. We now show Theorem 2.5. We first use Lemma 5.1 to compute an SLP $\mathbb{A}_i$ for a candidate ideal for each L_i. By directedness, $L_i{\downarrow} = \mathtt{Idl}(\mathrm{val}(\mathbb{A}_i))$. To decide $\mathtt{Idl}(\mathrm{val}(\mathbb{A}_1)) = \mathtt{Idl}(\mathrm{val}(\mathbb{A}_2))$, we use the fact that two reduced ideal representations yield the same ideal if and only if they are syntactically identical [33, Theorem 6.1.12]. To check $\mathrm{val}(\mathbb{A}_1) = \mathrm{val}(\mathbb{A}_2)$ we may apply the well-known result of Plandowski [42] that equality of SLPs can be decided in polynomial time (see also [40]).

For the P lower bound, we reduce from emptiness of CFL: Given a CFG G, we may assume that the only word G can produce is the empty word (otherwise, just replace all occurrences of terminal letters with the empty word). We also take a grammar G' with $L(G') = \{\varepsilon\}$. Then clearly, $L(G) \neq \emptyset$ if and only if $L(G){\downarrow} = L(G'){\downarrow}$, yielding P-hardness.

7 Conclusion

We have initiated the investigation of the directedness problem and determined the exact complexity for context-free languages and for NFAs over fixed alphabets. Over variable alphabets, we show an AC^1 upper bound for NFAs. Despite serious efforts, we leave the exact complexity open. Note that the complexity of directedness is the same for DFAs and NFAs [27, App.F]. Also, the complexity of the maximum weight path problem is not known [22].

The developed techniques could be of independent interest. The idea to analyze ideals by their weights might apply to other procedures for reachability involving ideals [16–18, 25, 26, 36–38]. Furthermore, our PSPACE lower bound can be viewed as progress towards resolving the complexity of the *compressed subword problem*: Our lower bound applies in particular to deciding $L \subseteq I$ for context-free L and a compressed ideal I. Compressed subword, on the other hand, is equivalent to deciding $I \subseteq J$ for compressed ideals I, J. As mentioned before, it is a long-standing open problem to close the gap between the PP lower bound and the PSPACE upper bound [39] (see [40] for a survey) for compressed subword.

The surprisingly low complexity of downward closure equivalence (DCE) for directed CFL calls for an investigation of further applications of directed CFL. As previously stated, safety properties of concurrent programs only depend on the downward closure of the participating threads [5, 11, 41]. It is conceivable that deciding safety [5, 13] or other notoriously difficult problems such as refinement [10] are more tractable for directed threads as well.

References

1 Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998. doi:10.1007/BFb0028754.

2 Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods Syst. Des.*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.

3 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

4 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993. doi:10.1016/0304-3975(93)90252-O.

5 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:4)2011.

6 Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetzsche. The complexity of regular abstractions of one-counter languages. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 207–216. ACM, 2016. doi:10.1145/2933575.2934561.

7 Aistis Atminas and Vadim V. Lozin. Deciding atomicity of subword-closed languages. In Volker Diekert and Mikhail V. Volkov, editors, *Developments in Language Theory - 26th International Conference, DLT 2022, Tampa, FL, USA, May 9-13, 2022, Proceedings*, volume 13257 of *Lecture Notes in Computer Science*, pages 69–77. Springer, 2022. doi:10.1007/978-3-031-05578-2_5.

8 Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 473–485. Springer, 2015. doi:10.1007/978-3-319-15579-1_37.

9 David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Pawel Parys. Cost automata, safe schemes, and downward closures. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 109:1–109:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.109.

10 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Checking refinement of asynchronous programs against context-free specifications. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 110:1–110:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.110.

11 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Context-bounded analysis of concurrent programs (invited talk). In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.3.

12 Pascal Baumann, Moses Ganardi, Ramanathan S. Thinniyam, and Georg Zetzsche. Existential definability over the subword ordering. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.7.

13 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. The complexity of bounded context switching with dynamic thread creation. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 111:1–111:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.111.

14 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Context-bounded verification of thread pools. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022. doi:10.1145/3498678.

15 Jean Berstel. *Transductions and Context-Free Languages.* Teubner, 1979.

16 Michael Blondin, Alain Finkel, and Jean Goubault-Larrecq. Forward analysis for WSTS, part III: Karp-Miller trees. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.16.

17 Michael Blondin, Alain Finkel, and Pierre McKenzie. Well behaved transition systems. *Log. Methods Comput. Sci.*, 13(3), 2017. doi:10.23638/LMCS-13(3:24)2017.

18 Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching well-structured transition systems. *Inf. Comput.*, 258:28–49, 2018. doi:10.1016/j.ic.2017.11.001.

19 Manuel Bodirsky, Jakub Rydval, and André Schrottenloher. Universal Horn sentences and the joint embedding property. *Discret. Math. Theor. Comput. Sci.*, 23(2), 2021. doi:10.46298/dmtcs.7435.

20 Samuel Braunfeld. The undecidability of joint embedding and joint homomorphism for hereditary graph classes. *Discret. Math. Theor. Comput. Sci.*, 21(2), 2019. doi:10.23638/DMTCS-21-2-9.

21 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.

22 Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Inf. Control.*, 64(1-3):2–21, 1985. doi:10.1016/S0019-9958(85)80041-3.

23 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, January 1991.

24 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 35:1–35:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.35.

25 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part II: complete WSTS. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2009. doi:10.1007/978-3-642-02930-1_16.

26 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: completions. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 433–444. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1844.

27 Moses Ganardi, Irmak Sağlam, and Georg Zetzsche. Directed regular and context-free languages, 2024. arXiv:2401.07106.

28 Jean Goubault-Larrecq, Simon Halfon, Prateek Karandikar, K. Narayan Kumar, and Philippe Schnoebelen. The ideal approach to computing closed subsets in well-quasi-ordering. *CoRR*, abs/1904.10703, 2019. arXiv:1904.10703.

29 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 97:1–97:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.97.

30 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.

31 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.

32 Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969. doi:10.1016/S0021-9800(69)80111-0.

33 Simon Halfon. *On Effective Representations of Well Quasi-Orderings*. PhD thesis, Université Paris Saclay, 2018. URL: https://theses.hal.science/tel-01945232.

34 P. Jullien. Sue un théorème d'extension dans la théorie des mots. *C. R. Acad. Sci. Paris, Ser. A*, 266:851–854, 1968.

35 Mustapha Kabil and Maurice Pouzet. Une extension d'un théorème de p. jullien sur les âges de mots. *RAIRO Theor. Informatics Appl.*, 26:449–482, 1992. doi:10.1051/ita/1992260504491.

36 Ranko Lazic and Sylvain Schmitz. The ideal view on Rackoff's coverability technique. *Inf. Comput.*, 277:104582, 2021. doi:10.1016/j.ic.2020.104582.

37 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

38 Jérôme Leroux and Sylvain Schmitz. Ideal decompositions for vector addition systems (invited talk). In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 1:1–1:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.1.

39 Markus Lohrey. Leaf languages and string compression. *Inf. Comput.*, 209(6):951–965, 2011. doi:10.1016/j.ic.2011.01.009.

40 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

41 Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. General decidability results for asynchronous shared-memory programs: Higher-order and beyond. *Log. Methods Comput. Sci.*, 18(4), 2022. doi:10.46298/lmcs-18(4:2)2022.

42 Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In Jan van Leeuwen, editor, *Algorithms - ESA '94, Second Annual European Symposium, Utrecht, The Netherlands, September 26-28, 1994, Proceedings*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470. Springer, 1994. doi:10.1007/BFb0049431.

43 Saul Schleimer. Polynomial-time word problems. *Commentarii mathematici helvetici*, 83(4):741–765, 2008.

44 Jeffrey O. Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2008. URL: `http://www.cambridge.org/gb/knowledge/isbn/item1173872/?site_locale=en_GB`.

45 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. `doi:10.1137/0220053`.

46 Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 72–84. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPIcs.CONCUR.2015.72`.

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

48 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. `doi:10.1007/978-3-662-03927-4`.

49 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2015. `doi:10.1007/978-3-662-47666-6_35`.

50 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.ICALP.2016.123`.

51 Georg Zetzsche. Separability by piecewise testable languages and downward closures beyond subwords. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 929–938. ACM, 2018. `doi:10.1145/3209108.3209201`.