The complexity of downward closures of indexed languages

Anonymous Author(s)

ABSTRACT

Indexed languages are a classical notion in formal language theory, which has attracted attention in recent decades due to its role in higher-order model checking: They are precisely the languages accepted by order-2 pushdown automata.

The downward closure of an indexed language—the set of all (scattered) subwords of its members—is well-known to be a regular over-approximation. It was shown by Zetzsche (ICALP 2015) that the downward closure of a given indexed language is effectively computable. However, the algorithm comes with no complexity bounds, and it has remained open whether a primitive-recursive construction exists.

We settle this question and provide a triply (resp. quadruply) exponential construction of a non-deterministic (resp. deterministic) automaton. We also prove (asymptotically) matching lower bounds.

For the upper bounds, we rely on recent advances in semigroup theory, which let us compute bounded-size summaries of words with respect to a finite semigroup. By replacing stacks with their summaries, we are able to transform an indexed grammar into a context-free one with the same downward closure, and then apply existing bounds for context-free grammars.

CCS CONCEPTS

• **Theory of computation → Grammars and context-free languages**; **Verification by model checking**; *Algebraic language theory*.

KEYWORDS

Indexed languages, Higher-order pushdown automata, Downward closures, Semigroups, Verification

ACM Reference Format:

Anonymous Author(s). 2018. The complexity of downward closures of indexed languages. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Downward closures. For finite words u and v, we say that u is a *(scattered) subword* of v, written $u \preceq v$, if v can be obtained from u by inserting letters. The *downward closure* of a language $L \subseteq \Sigma^*$ is the set $L{\downarrow} = \{u \in \Sigma^* \mid u \preceq v \text{ for some } v \in L\}$ of all subwords of members of L. By the well-known Higman's lemma [33], the downward closure $L{\downarrow}$ is regular for *any* language L.

This makes the downward closure a fundamental abstraction—it is a regular overapproximation that preserves information about pattern occurrences and unboundedness behavior. Specifically, in the verification of complex systems, downward closures are often used to replace infinite-state components by finite-state ones, which then enables the algorithmic analysis of the entire system. Examples include asynchronous programs [15, 43], shared-memory systems with dynamic thread creation [4, 8, 16–18], parameterized asynchronous shared-memory systems [41], and systems communicating via lossy channels [1, 5, 9].

For these reasons, it is often useful to *compute downward closures*, which means constructing a finite automaton for $L{\downarrow}$ when given a description (i.e. a grammar or an infinite-state recognizer) for L. This is a notoriously difficult task, as it requires a deep understanding of how L is generated/recognized. Therefore, the problem of computing downward closures has attracted significant attention over recent decades, with work on context-free languages [23, 54], systems with counting and concurrency [3, 10, 11, 29, 57], models of higher-order recursion [20, 30, 55], lossy channel systems [1, 44], general algorithms for broad classes of infinite-state systems [6, 55], representation sizes [12, 18, 28, 43], related algorithmic tasks [26, 47, 48, 58], and even computability beyond subwords [6, 7, 13, 24, 59].

Indexed languages. A setting where downward closure computation is particularly challenging is that of *indexed languages* [2], a classical notion in formal language theory that generalizes context-free languages. Essentially, indexed grammars differ from context-free grammars in that each non-terminal carries a stack, which can be pushed and popped through special rules. These grammars have recently attracted interest because of their role in higher-order model-checking [40, 46]: Indexed grammars are equivalent to order-2 pushdown automata, from the hierarchy of *higher-order pushdown automata (HOPA)*, which model safe higher-order recursion [38, 39] (see also the survey [46]). Level k of this hierarchy consists of the *order-k pushdown automata (k-PDA)*, which have access to *stacks of (stacks of (...)) stacks*—with nesting depth k: In particular, a 1-PDA is an ordinary pushdown automaton, whereas a 2-PDA has a stack of stacks, where it can operate on the top-most stack as an ordinary pushdown automaton; but it can also copy the top-most stack.

There is a significant gap in our understanding of downward closures of indexed languages (and HOPA more generally). There is a general approach for computing downward closures [55]. Based on this approach, computability of downward closures has been shown for indexed languages (equivalently, 2-PDA) [55], then for general HOPA [30], and even higher-order recursion schemes [20].

However, while mere computability of downward closures of these models is settled, the *complexity has remained a long-standing open problem.* This is because the algorithm from [55] enumerates automata and then solves instances of the so-called *simultaneous unboundedness problem (SUP)* to decide whether the current automaton in fact recognizes $L{\downarrow}$. Thus, although the complexity of the SUP

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

itself is well-understood for HOPA [14, 47], there are no complexity upper bounds for the entire task of computing downward closures of indexed languages (nor for languages recognized by HOPA in general). In fact, even the existence of a primitive-recursive (or even hyper-Ackermannian) upper bound remained open[1].

Contribution. In this work, we settle the complexity of computing downward closures of indexed languages. We show that given an indexed language L, one can compute a non-deterministic finite automaton (NFA) for $L\downarrow$ in triply exponential time (and hence of triply exponential size). We also provide a triply exponential lower bound, improving on the doubly exponential lower bound in [58].

Furthermore, our constructions also provide a tight bound for the computation of a deterministic finite automaton (DFA) for $L\downarrow$: It follows that one can construct a quadruply-exponential-sized DFA for $L\downarrow$, and we provide a quadruply-exponential lower-bound.

Moreover, our results settle the complexity of decision problems involving downward closures of indexed languages. The *downward closure inclusion problem* asks, whether for given indexed languages L_1 and L_2, we have $L_1\downarrow \subseteq L_2\downarrow$. Similarly, the *downward closure equivalence problem* asks whether $L_1\downarrow = L_2\downarrow$. We show that both problems are co-3-NEXP-complete.

Finally, our construction for the downward closure lower bound also settles another question about indexed languages: It implies that the known triply exponential upper bound on the pumping threshold for indexed grammars [32, 53] (the maximal length of words in a finite indexed language) is in fact asymptotically tight. (In [37], a doubly exponential upper bound was claimed, but that seems to be a miscalculation, see Section 3 and Appendix B.)

Why are the results unexpected? The complexity results come as a considerable surprise. This is because (tight) complexity bounds for HOPA are usually towers of exponentials where the height grows linearly with the order. For example, for $k \geq 1$, the emptiness problem for k-PDA is $(k-1)$-EXPTIME-complete [25, comment before Thm. 7.12] (note that 0-EXPTIME = P). The same is true of the SUP [47, Thm. 3]. Since downward closure inclusion and equivalence are coNP-complete for NFAs (see [12, Sec. 5] and [36, Prop. 7.3]) and coNEXP-complete for 1-PDAs [58, Tab. 1], and hence co-k-NEXP-complete for k-PDAs with $k \leq 1$, one would expect co-2-NEXP rather than co-3-NEXP for 2-PDAs. In fact, the best (and only) known lower bound until now has been co-2-NEXP [58, Cor. 18]. Similarly, since downward closure NFAs are polynomial-sized for given NFAs (sometimes considered 0-PDAs) and exponential-sized for given 1-PDAs [12, Cor. 6], one would expect a doubly exponential bound for 2-PDAs. In fact, the best known lower bound on the NFA size for downward closures had been doubly exponential [58, remarks before Cor. 18].

Key ingredients. Indexed grammars extend context-free grammars by equipping the nodes of derivation trees with pushdown store (which we also call stack). This way, each branch of a derivation tree corresponds to a run of a pushdown automaton.

Our construction relies on recent advances in finite semigroup theory, which provide succinct "summaries" of arbitrarily long words relative to a finite semigroup [27]. We use this (for a suitable semigroup) to replace the stack inside each derivation tree node by such a summary, each of which takes up exponentially many bits. This replacement changes the overall language, but preserves the downward closure. Once the information in each node is bounded, we can transform the grammar into a doubly-exponential-sized context-free grammar. This yields the triply exponential upper bound overall, since for context-free grammars, existing algorithms yield exponential-sized downward closure NFAs [12, Corollary 6].

The aforementioned summaries are similar in spirit to Simon's factorization forests [52]. The latter annotate words by trees: Relative to a morphism $\varphi\colon \Sigma^* \to M$ into a finite monoid M, a factorization forest for $w \in \Sigma^*$ is a tree of height bounded by a function of $|M|$ that allows evaluating φ on infixes of w without processing the entire infix. Here, a crucial idea is that a sequence of infixes that all map under φ to the same idempotent e must evaluate to e.

Similar to factorization forests, our summaries also exploit repetitions of idempotents to collapse long infixes in a stack. However, in contrast to factorization forests, our summaries reduce the stack word to one of bounded length (hence losing information). Also crucially, taking summaries is compatible with pushing stack symbols: Given a summary of w, one can compute a summary of aw.

Structure of the paper. We recall necessary notations and basic results in Section 2, and state the main results in Section 3. Then in Section 4 we show how to make a given indexed grammar *productive*, meaning that every partial computation can be extended to a full one. In Section 5 we introduce an object at the core of our construction, the *production monoid*. In Section 6 we introduce new rules that extend and reduce parts of the stack without altering the downward closure of the language. We then proceed in Section 7 to define *summaries* of stack contents with respect to the production monoid. In Section 8 we leverage those summaries to compute from an indexed grammar a context-free one with the same downward closure, to which known constructions apply. Finally, in Section 9 we complement our upper bounds with matching lower bounds.

This paper contains internal links; every occurrence of a term is linked to its definition. The reader can click on terms (and some notations), or simply hover over them on some pdf readers, to get their definition.

2 PRELIMINARIES

Words, trees and languages. Given a finite alphabet Σ, we write Σ^* for the set of words over Σ, and Σ^k for the set of words of length k. Given $w \in \Sigma^*$, we denote $|w|$ its length. We assume familiarity with basic finite automata theory, see [49] for an introduction.

We write $u \preceq v$ if u is a (scattered) *subword* of v; that is, if u can be obtained from v by removing some of its letters. If a word w is equal to uv, then we call u a *prefix*, and v a *suffix* of w. The *downward closure* of a language $L \subseteq \Sigma^*$ is the set of subwords of words of L, and is denoted $L\downarrow$. An important reason for studying downward closures is that they are always regular. This was first shown by Haines [31, Theorem 3], but also follows from $\preceq$ being a well-quasi ordering, which was shown earlier by Higman [33, Theorem 4.3]:

[1]Given that computing downward closures involves a well-quasi-ordering (WQO) at its core, and WQO-based algorithms employing the subword ordering can often be furnished with hyper-Ackermannian/multiply-recursive upper bounds via length-function theorems [50, 51], one might hope for such a bound here. Unfortunately, it is not clear how to apply length-function theorems to downward closure computation.

Theorem 2.1. *The downward closure of any language is regular.*

A finite ordered Σ-labeled binary tree is a pair (τ, λ) with τ a finite prefix-closed subset of $\{0,1\}^*$ such that for all $v \in \tau$, $v1 \in \tau$ implies $v0 \in \tau$, and $\lambda : \tau \to \Sigma$ a function mapping each node in τ to a label. We use the usual terminology for trees, with *node, leaf, branch, subtree, parent, child, ancestor, descendant* etc. retaining their usual meaning. The *leaf word* of τ is the word obtained by concatenating $\lambda(v_1)\dots\lambda(v_k)$, where $v_1, \dots, v_k$ are the leaves of τ read from left to right, i.e., in lexicographic order.

Indexed grammars. In this work we use a syntax for indexed grammars that resembles the Chomsky normal form used for context-free grammars. This simplifies semantics as well as proofs.

An *indexed grammar* is a tuple $\mathcal{G} = (N, T, I, P, S)$ with

- N the set of non-terminals, and $S \in N$ the starting non-terminal
- T the set of terminal symbols
- I the set of index symbols, also called stack symbols
- P a set of productions, which are of the following types:
  - $A \to w$ with $w \in T^*$.
  - $A \to BC$ with $A, B, C \in N$
  - $A \to Bf$ with $A, B \in N$ and $f \in I$
  - $Af \to B$ with $A, B \in N$ and $f \in I$

We define the *size* of $\mathcal{G}$ as $|N| + |P| + \sum_{A \to w \in P} |w|$. A *context-free grammar* (or CFG) is an indexed grammar where all productions are of the two first forms.

A *sentential form* is a word over the (infinite) alphabet $NI^* \cup T$. The set of sentential forms is denoted $\mathbf{SF}_{\mathcal{G}}$, or simply $\mathbf{SF}$ when the grammar is clear from context. We write the elements of NI^* as $A[z]$, with $A \in N$ a non-terminal and $z \in I^*$ interpreted as the content of its stack; such an element is sometimes referred to as a *term*. If $u \in \mathbf{SF}$, we occasionally use the notation $u[z]$ to denote the sentential form obtained by pushing z on top of the stack of every non-terminal in u. The derivation relation $\Rightarrow_{\mathcal{G}}$ (or simply $\Rightarrow$) over $\mathbf{SF}$ is defined as:

$$\begin{aligned}
uA[z]v &\Rightarrow uB[z]C[z]v && \text{if } A \to BC \in P,\\
uA[z]v &\Rightarrow uB[fz]v && \text{if } A \to Bf \in P,\\
uA[fz]v &\Rightarrow uB[z]v && \text{if } Af \to B \in P,\\
uA[z]v &\Rightarrow uwv && \text{if } A \to w \in P.
\end{aligned}$$

for all $u, v \in \mathbf{SF}$, $A, B, C \in N$, $f \in I$, $z \in I^*$ and $w \in T^*$. We write $\xRightarrow{*}_{\mathcal{G}}$ (or simply $\xRightarrow{*}$) for the reflexive transitive closure of $\Rightarrow_{\mathcal{G}}$.

The subword relation $\preceq$ is extended to $\mathbf{SF}$ in the natural way, i.e. $u \preceq v$ if u can be obtained from v by deleting terms. Note that on $\mathbf{SF}$, the ordering $\preceq$ is not a well-quasi ordering, since any two terms $A[z]$ and $A[z']$ are incomparable for $z, z' \in I^*$, $z \neq z'$.

Given a sentential form u, we write $\mathsf{L}_{\mathbf{SF}}(u)$ for the set of sentential forms derivable from u, $\{v \in \mathbf{SF} \mid u \xRightarrow{*} v\}$. The *language* of $\mathcal{G}$ is denoted $\mathsf{L}(\mathcal{G})$ and defined as $\mathsf{L}_{\mathbf{SF}}(S) \cap T^*$. Furthermore, for all $X \subseteq N$ and $u \in \mathbf{SF}$, the language $\mathsf{L}_X(u)$ is defined as the set $\mathsf{L}_{\mathbf{SF}}(u) \cap (X \cup T)^*$ of sentential forms derivable from u with all stacks empty, and all non-terminals in X. In particular, $\mathsf{L}_\emptyset(u)$ is the set of terminal words which can be derived from u. If the language $\mathsf{L}_\emptyset(u)$ is non-empty then we say that u is *productive*.

A *derivation tree* is a finite ordered tree τ whose nodes are labeled by elements of $NI^* \cup T^*$, with the following constraints. For each node v with label $A[z]$ (where $A \in N$ and $z \in I^*$), exactly one of the following holds

- v is a leaf
- v has one child labeled w for some $A \to w \in P$
- v has two children labeled $B[z]$ and $C[z]$, for some production $A \to BC \in P$
- v has one child labeled $B[fz]$ for some $A \to Bf \in P$
- v has one child labeled $B[z']$ for some $Af \to B \in P$, with $z = fz'$.

A derivation tree whose root is labeled $A[z]$ and whose leaf word is equal to u is called a *derivation tree from $A[z]$ to u*. If $u \in T^*$ we say that the derivation tree is *complete*.

Remark 2.1. *An easy induction shows that for all $A[z] \in NI^*$, a sentential form u can be derived from $A[z]$ if and only if u is the leaf word of a derivation tree whose root is labeled $A[z]$. In the forthcoming proofs we will either use sentential forms or derivation trees depending on what is most convenient.*

Remark 2.2. *When describing examples of indexed grammars, we will use rules of the form $Af \to u$ and $A \to u$ with $u \in (N \cup T)^*$. This is just syntactic sugar, as we can replace them with rules from Definition 2 while adding a small number of non-terminals, in the spirit of the Chomsky normal form [19]. This transformation incurs a linear size increase the grammar. Details are left to Appendix A.*

Example 2.2. We can define the language $\{a^n b^{n^2} \mid n \in \mathbb{N}\}$ with an indexed grammar:

$S \to T\,g$	*# g is the stack bottom symbol;*
$T \to T\,f$	*# we push some number n of f;*
$T \to A$	
$A\,g \to \varepsilon$	*# if $n = 0$ then return ε;*
$A\,f \to C$	*# if $n > 0$ we pop an f;*
$C \to aAB$	*# repeat n times to get $a^n B[g] B[fg] \cdots B[f^{n-1}g]$;*
$B\,f \to bbB$	
$B\,g \to b$	*#the Bs output $b^{\sum_{i=0}^{n-1}(2i+1)} = b^{n^2}$.*

Remark 2.3. *We can assume without loss of generality that every symbol pushed on the stack carries the information of the production rule used to push it (this property can always be ensured by adding a quantity of new stack symbols and production rules that is at most quadratic in the size of the grammar). Formally, we assume that there are functions $\alpha, \beta : I \to N$ such that for every push rule $A \to Bf$ we have $A = \alpha(f)$ and $B = \beta(f)$. We also assume that for all $f \in I$ there is a rule of the form $A \to Bf$ in P. Clearly stack symbols not satisfying this condition can be removed.*

Complexity. We define the functions $\exp_k : \mathbb{N} \to \mathbb{N}$ inductively by setting $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$. A function $f : \mathbb{N} \to \mathbb{N}$ is *(at most) k-fold exponential* if there is a constant $c > 0$ such that $f(n) \leq \exp_k(n^c)$ for almost all n. We say that f is *at least k-fold exponential* if there is a constant $c > 0$ such that $f(n) \geq \exp_k(n^c)$ for infinitely many n. Instead of 2-fold, 3-fold, 4-fold exponential, resp., we also say doubly, triply, or quadruply exponential, resp. For $k \geq 1$, the class co-k-NEXP consists of the complements of sets accepted by an f-time-bounded non-deterministic Turing machine, for some $f: \mathbb{N} \to \mathbb{N}$ that is at most k-fold exponential.

3 MAIN RESULTS

In this section, we present the main results of this work.

Non-deterministic automata. Our first main contribution is an algorithm to compute an NFA of at most triply exponential size for the downward closure of an indexed language:

Theorem 3.1. *Given an indexed grammar $\mathcal{G}$, one can compute (in triply exponential time) a triply-exponential-sized NFA for $\mathsf{L}(\mathcal{G})\downarrow$.*

We also provide an asymptotically matching lower bound, which we infer from the following result:

Theorem 3.2. *Given $n \in \mathbb{N}$ (in unary encoding), we can compute in polynomial time an indexed grammar for the language $\{\mathsf{a}^{\exp_3(n)}\}$.*

An NFA for $\{\mathsf{a}^{\exp_3(n)}\}\downarrow$ clearly requires at least $\exp_3(n)$ states, implying Corollary 3.3.

Corollary 3.3. *There is a family $(\mathcal{G}_n)_{n\geq 1}$ of indexed grammars of size polynomial in n such that any NFA for $\mathsf{L}(\mathcal{G}_n)\downarrow$ requires at least $\exp_3(n)$ states.*

Deterministic automata. Of course, Theorem 3.1 implies a quadruply exponential upper bound for deterministic automata:

Corollary 3.4. *Given an indexed grammar $\mathcal{G}$, one can compute (in quadruply exponential time) a DFA of at most quadruply exponential size for $\mathsf{L}(\mathcal{G})\downarrow$.*

Here, we have an asymptotically matching lower bound as well:

Theorem 3.5. *There is a family $(\mathcal{G}_n)_{n\geq 1}$ of indexed grammars of size polynomial in n such that any DFA for $\mathsf{L}(\mathcal{G}_n)\downarrow$ requires at least $\exp_4(n)$ states.*

Downward closure comparisons. Theorem 3.1 and our construction for Theorem 3.2 also allow us to settle the complexity of decision problems related to downward closures. The *downward closure inclusion problem* (for indexed languages) asks whether two given indexed languages L_1, L_2 satisfy $L_1\downarrow \subseteq L_2\downarrow$. Similarly, the *downward closure equivalence problem* asks whether $L_1\downarrow = L_2\downarrow$.

In [58, Corollary 18], it was shown that downward closure inclusion and equivalence for indexed languages are co-2-NEXP-hard. Here, we settle their precise complexity:

Theorem 3.6. *Downward closure inclusion and downward closure equivalence for indexed languages are* co-3-NEXP*-complete.*

Here, the upper bounds follow from Theorem 3.1 and the fact that downward closure inclusion and equivalence are coNP-complete for NFAs (see [12, Section 5] and [36, Proposition 7.3]).

The pumping threshold for indexed languages. Our lower bound technique also settles the growth of the pumping threshold of indexed grammars. Consider the function

$$\mathfrak{P}(n) = \max\{|w| \mid \text{there is an indexed grammar } \mathcal{G} \text{ of size } n \text{ with } w \in \mathsf{L}(\mathcal{G}) \text{ such that } \mathsf{L}(\mathcal{G}) \text{ is finite}\}$$

We call $\mathfrak{P}(n)$ the *pumping threshold* (for size n) because placing an upper bound on $\mathfrak{P}(n)$ usually involves a pumping argument.

An analogous pumping threshold function for NFAs and CFGs is well-understood: It is linear for NFAs and exponential for CFGs. For indexed grammars, there are two proofs of a triply exponential upper bound: the pumping lemmas of Hayashi [32, Theorem 5.1] and Smith [53, Theorem 1] (Smith's proof mentions the bound explicitly; Hayashi's proof requires some analysis for this). We complement this by showing a triply exponential lower bound:

Corollary 3.7. *$\mathfrak{P}$ grows at least triply exponentially.*

This follows from Theorem 3.2, because $\{\mathsf{a}^{\exp_3(n)}\}$ is finite but has an indexed grammar of size polynomial in n. It should be noted that [37, Section 7] claims a doubly exponential upper bound for $\mathfrak{P}$. Unfortunately, this seems to result from a miscalculation, see Appendix B for a discussion.

Remark 3.1. *A triply exponential upper bound on $\mathfrak{P}(n)$ also follows from our Theorem 3.1: If an indexed grammar $\mathcal{G}$ generates a word that is longer than the number of states of an NFA for $\mathsf{L}(\mathcal{G})\downarrow$, then $\mathsf{L}(\mathcal{G})\downarrow$ must be infinite, and hence also $\mathsf{L}(\mathcal{G})$.*

Structure of the paper. In Sections 4 to 8, we will prove Theorem 3.1, from which all remaining upper bounds follow. In Section 9, we will then prove all lower bounds.

4 PRODUCTIVENESS

For the rest of the paper, we assume that our input grammar $\mathcal{G}$ generates a non-empty language. This is because emptiness of indexed languages can be decided in EXPTIME [25, Theorem 7.12], and if $\mathsf{L}(\mathcal{G})$ is empty, an NFA for $\mathsf{L}(\mathcal{G})\downarrow$ is immediate.

However, we will need to establish the stronger guarantee of *productiveness*, which expresses the absence of deadlocks. This means that if we can produce a sentential form u, then from u we can derive a terminal word in T^*.

Productive grammars. We say that $\mathcal{G}$ is *productive* if for all $u \in \mathsf{L}_{\mathsf{SF}}(S)$ we have $\mathsf{L}_{\emptyset}(u) \neq \emptyset$, that is, from every sentential form obtained from S we can derive a word in T^*.

This property is especially useful for downward closure computation. In a productive indexed grammar, we can observe: for all $u, v \in \mathsf{L}_{\mathsf{SF}}(S)$, if v is such that $v \preceq u$, then $\mathsf{L}_{\emptyset}(v) \subseteq \mathsf{L}_{\emptyset}(u)\downarrow$. In other words, we can interleave derivations and deletion of terms without any risk of obtaining "extra" terminal words. This property does not hold in general, because deleting terms that cannot produce terminal words may allow the derivation of a terminal word that is not in the downward closure.

Example 4.1. The following grammar $\mathcal{G}$ is not productive.

$$\begin{array}{lll} S \to S\bot & S \to Sf & S \to Sg \\ S \to AA & S \to AAC & \\ Af \to aA & Ag \to b & \\ Cf \to cC & Cg \to CAB & \\ A\bot \to \varepsilon & C\bot \to \varepsilon. & \end{array}$$

The language of $\mathcal{G}$ is $\{w^2 \mid w \in \{a,b\}^*\} \cup \{a^{2n}c^n \mid n > 0\}$. In particular, by setting $u = A[gf\bot]A[gf\bot]C[f\bot]A[f\bot]B[f\bot]$

and $v = C[f\bot]A[f\bot]$ we have $v \preceq u$ and $u \in \mathsf{L}_{\mathbf{SF}}(S)$, but $ca \in \mathsf{L}_\emptyset(v)$ while $\mathsf{L}_\emptyset(u)\!\downarrow = \emptyset$ (since B cannot produce a terminal word). Moreover, $ca \notin \mathsf{L}(\mathcal{G})\!\downarrow$. In this case, it is easy to modify $\mathcal{G}$ to obtain a productive grammar which generates the same language.

Tracking productivity of non-terminals. Before we describe our method for achieving productiveness, we observe that tracking the non-terminals which, with a given stack content, can derive a sentential form with terms confined to a certain subset, gives rise to a left semigroup action.

Definition 4.2. For $X \subseteq N$ and $z \in I^*$, we define

$$z \cdot X = \{A \in N \mid \mathsf{L}_X(A[z]) \neq \emptyset\},$$

i.e. $z \cdot X$ is the set of non-terminals A so that the term $A[z]$ can derive a sentential form consisting of terminals and empty-stack occurrences of non-terminals in X. Let $\mathbf{U}$ be defined as the set of non-terminals which can derive a word in T^*, i.e.

$$\mathbf{U} = \{A \in N \mid \mathsf{L}_\emptyset(A) \neq \emptyset\}.$$

Note that $\mathsf{L}(\mathcal{G})$ is empty if and only if $S \notin \mathbf{U}$.

We will often rely on the fact that I^* acts (on the left) as a semigroup[2] on the power set 2^N, as we state now:

Lemma 4.3. *For every $f \in I$, $z \in I^*$, and $X \subseteq N$, we have $fz \cdot X = f \cdot (z \cdot X)$. Moreover, $z \cdot \mathbf{U} = z \cdot \emptyset$.*

Here, only the inclusion $fz \cdot X \subseteq f \cdot (z \cdot X)$ is not trivial: It holds because a derivation that eliminates fz from the stack must in each branch first remove f, and then z. The proof is in Appendix C.1.

Achieving productiveness. We are now ready to present our construction of a productive equivalent of $\mathcal{G}$. Formally, the *annotated version* of $\mathcal{G}$, denoted $\overline{\mathcal{G}} = (\overline{N}, T, \overline{I}, \overline{P}, \overline{S})$, is defined as follows. Recall that we assumed $\mathsf{L}(\mathcal{G}) \neq \emptyset$, thus $S \in \mathbf{U}$. Otherwise, we have:

- $\overline{N} = \{(A, X) \in N \times 2^N \mid A \in X\}$, $\overline{I} = I \times 2^N$, and $\overline{S} = (S, \mathbf{U})$,
- $\overline{P}$ contains the following rules:
  - $(A, X) \to w$ for all $A \to w \in P$ with $w \in T^*$
  - $(A, X) \to (B, X)(C, X)$ for all $A \to BC \in P$ with $A, B, C \in X$
  - $(A, X) \to (B, Y)(f, X)$ for all $A \to Bf \in P$ with $Y = f \cdot X$, $A \in X$ and $B \in Y$
  - $(A, Y)(f, X) \to (B, X)$ for all $Af \to B \in P$ with $Y = f \cdot X$, $A \in Y$ and $B \in X$

Note that a stack word $\bar{z} = (f_n, X_n) \cdots (f_1, X_1) \in \overline{I}^*$ appearing as an infix of a stack content in a derivation of $\overline{\mathcal{G}}$ must be so that $X_i = f_{i-1} \cdot X_{i-1}$ for all $i > 1$. A stack word satisfying this property is determined by its projection onto I^* and its last element. Given $z = f_n \cdots f_1 \in I^*$ and $X \subseteq N$, define $\overline{z}^X$ as $(f_n, X_n) \cdots (f_1, X_1)$ with $X_1 = X$ and $X_{i+1} = f_i \cdot X_i$ for all $i < n$. Given $A[z] \in NI^+$ with $z = f_n \cdots f_1$ and $A \in z \cdot X$, the *X-based annotation* of $A[z]$ is the $\overline{\mathcal{G}}$ term $(A, Y)[\bar{z}] = (A, Y)[(f_n, X_n) \cdots (f_1, X_1)] \in \overline{N}\,\overline{I}^+$ such that $X_1 = X$, $Y = f_n \cdot X_n$ and $X_i = f_{i-1} \cdot X_{i-1}$ for all $i > 1$. In particular, by Lemma C.1 we have $X_i = f_{i-1} \cdots f_1 \cdot X$ for all i and $Y = z \cdot X$. In the case of an empty stack,the X-based annotation of A is (A, X).

Correctness of the construction. Having defined $\overline{\mathcal{G}}$, we can prove that it serves its purpose:

[2]However, it does not act as a monoid, because $\varepsilon \cdot X$ it not necessarily X, as a set $z \cdot X$ always includes $\mathbf{U}$.

Lemma 4.4. *$\mathcal{G}$ and $\overline{\mathcal{G}}$ have the same language, and $\overline{\mathcal{G}}$ is productive.*

The mostly straightforward proof can be found in Appendix C.2.

Remark 4.1. *Although we are able to turn any indexed grammar into a productive one with the same language, this transformation incurs an exponential blow-up in the size. Therefore, in order to obtain tight complexity bounds, we do not simply assume productiveness of the input grammar. Rather, we work with annotated versions explicitly when needed.*

5 THE STACK MONOID

Overall goal. Another key aspect of our construction is the stack monoid—a finite monoid in which we evaluate stack contents. Essentially, we map a stack $\bar{z} \in \overline{I}^*$ to a monoid element that encodes[3]

- the non-terminals from and to which this content was pushed (which are unique thanks to Remark 2.3), and
- for each pair A, B of non-terminals, whether we can derive a sentential form containing B from $A[\bar{z}]$ (i.e., whether we can obtain a B by popping $\bar{z}$ from A).

The purpose of this encoding is that based on this information, we will be able to simplify (or expand) stack contents during derivations, *while preserving the downward closure*. For example, if we have a derivation from $A[z]$ to a sentential form uBv, then we could just erase u and v, thus turning $A[z]$ into B. Even better, if we have a derivation from $A[z]$ to A then, roughly speaking, this means we can turn any term $A[zz']$ into $A[z']$.

Observations like this, based on the stack monoid, will be used in Section 6 to define more involved stack manipulations. In Section 7, these will allow us to reduce, roughly speaking, every stack to one of doubly exponentially many.

Semigroup terminology. We begin with some terminology. A semigroup $(\mathbf{S}, \cdot)$ is a set equipped with an associative product. We will often identify a semigroup with its set of elements and denote it simply as $\mathbf{S}$, when the product operation is clear. A monoid $(\mathbf{M}, \cdot, \mathbf{1_M})$ is a semigroup with a neutral element $\mathbf{1_M}$.

Given a monoid $(\mathbf{M}, \cdot)$, define $\psi_{\mathbf{M}} : \mathbf{M}^* \to \mathbf{M}$ to be its *evaluation morphism*, which maps a sequence of elements of $\mathbf{M}$ to its product, with the empty sequence being mapped to $\mathbf{1_M}$. An element $x \in \mathbf{M}$ is *idempotent* if $x \cdot x = x$. The set of idempotent elements of $\mathbf{M}$ is denoted $\mathbf{Idem(M)}$.

In all that follows we fix an indexed grammar $\mathcal{G} = (N, T, I, P, S)$. We will mainly work with its annotated version $\overline{\mathcal{G}}$. Note that we cannot use the annotation construction as a black box and simply assume that the input grammar is productive (see Remark 4.1).

Formal definition. Let us now start by describing formally the monoid at hand. It is a bit more involved than what is described above since we have to account for the productiveness issues explained in the previous section. We will use the boolean matrix monoid over non-terminals of $\mathcal{G}$: This monoid is defined as the set of matrices $\mathbb{B}^{N \times N}$, with $\mathbb{B} = \{\top, \bot\}$. The product is the usual product of matrices over the Boolean semiring $(\mathbb{B}, \vee, \wedge)$, and the neutral

[3]Notice that there is a dissymmetry between push and pop here (in fact, throughout the construction). This is because a stack symbol is only pushed once, but may be popped on multiple branches.

element is the matrix with $\top$ on the diagonal and $\bot$ everywhere else. Given matrices M_1, M_2, we write $M_1 M_2$ for their product.

We also define, for all $X \subseteq N$, the *reachability relation* $\mathcal{R}_X$ over N. It relates two non-terminals if from the first one we can produce a sentential form containing the second one, with empty stacks. Formally,

$$A \mathcal{R}_X B \text{ if and only if there is a derivation } (A, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(B, X)v \text{ with } u, v \in \mathbf{SF}$$

(note that due to the productiveness property, requiring stack emptiness is only important for (B, X)).

Let Q be the set of tuples (B, Y, M, A, X) with $A, B \in N$ non-terminals, $X, Y \subseteq N$ sets of non-terminals, and $M \in \mathbb{B}^{N \times N}$. Define the *stack monoid* as the monoid $(\mathbb{M}, \cdot, \mathbf{1}_{\mathbb{M}})$ whose elements are $Q \cup \{\mathbf{1}_{\mathbb{M}}\} \cup \{\mathbf{0}_{\mathbb{M}}\}$, where $\mathbf{0}_{\mathbb{M}}$ satisfies $\mathbf{0}_{\mathbb{M}} \cdot x = x \cdot \mathbf{0}_{\mathbb{M}} = \mathbf{0}_{\mathbb{M}}$ for all $x \in \mathbb{M}$, and whose product operation is defined as follows:

$$(B_2, Y_2, M_2, A_2, X_2) \cdot (B_1, Y_1, M_1, A_1, X_1) = \begin{cases} (B_2, Y_2, M_1 M_2, A_1, X_1) \text{ if } X_2 = Y_1 \text{ and } B_1 \mathcal{R}_{X_2} A_2 \\ \mathbf{0}_{\mathbb{M}} \text{ otherwise.} \end{cases}$$

Let α, β be the functions described in Remark 2.3 for $\mathcal{G}$. We define a morphism $\varphi : \overline{I}^+ \to \mathbb{M}$ as follows. For each letter $(f, X) \in I$, set

$$\varphi(f, X) = (\beta(f), f \cdot X, M_{f,X}, \alpha(f), X),$$

where for all $A, B \in N$, $M_{f,X}(A, B) = \top$ if and only if there exist $u, v \in (X \cup T)^*$ such that $A[f] \overset{*}{\Rightarrow}_{\mathcal{G}} uBv$. Note that computing this matrix easily reduces to an emptiness check for an indexed grammar, which can be done in exponential time.

Feasibility. Let us mention some basic properties of stacks that are encoded in their image in $\mathbb{M}$. The first concerns whether a given stack content can be pushed: We say that a non-empty stack content (in either I^* or $\overline{I}^*$) is *feasible* if, starting from some non-terminal, it can be pushed onto the stack of some non-terminal. In the case of an annotated stack content $\bar{z} = (f_n, X_n) \cdots (f_1, X_1) \in \overline{I}^*$, this is equivalent to the existence of a derivation

$$(\alpha(f_1), X_1) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(\beta(f_n), f_n \cdot X_n)[\bar{z}]v$$

with $u, v \in \mathbf{SF}$. The following lemma says that the stack monoid distinguishes the feasible stack contents in $\overline{I}^*$.

Lemma 5.1. *Let $\bar{z} = (f_n, X_n) \cdots (f_1, X_1) \in \overline{I}^*$ be a stack content. The following are equivalent:*

(1) $\bar{z}$ is feasible
(2) $\varphi(\bar{z}) \neq \mathbf{0}_{\mathbb{M}}$
(3) for all $i > 1$, $X_i = f_i \cdot X_{i-1}$ and $\beta(f_{i-1}) \mathcal{R}_{X_i} \alpha(f_i)$.

The proof is given in Appendix D.1. Note that Lemma 5.1 shows that all feasible stack contents in $\overline{I}^*$ can be written as $\overline{z}^X$ for some feasible $z \in I^*$ and $X \subseteq N$ (hence, we sometimes assume that a stack content is of this form).

Let $(A, X) \in \overline{N}$ and $\bar{z} = (f_n, X_n) \cdots (f_1, X_1) \in \overline{I}^*$, we say that the term $(A, X)[\bar{z}]$ is *feasible* if $\bar{z}$ is feasible, $X = f_n \cdot X_n$ and $\beta(f_n) \mathcal{R}_X A$.

Pushing between specific non-terminals. We will now see that $\mathbb{M}$ encodes which non-terminals allow a stack content to be pushed, and which non-terminals can result. More formally, $\varphi(\bar{z})$ encodes all pairs of non-terminals $(C, X), (D, Y) \in \overline{N}$ such that (C, X) can derive a sentential form $u(D, Y)[\bar{z}]v$.

Lemma 5.2. *Let $z \in I^+$ a non-empty stack content, let $X \subseteq N$ and let $(B, Y, M, A, X) = \varphi(\overline{z}^X)$. Then for all $C \in X$ and $D \in Y$, the following are equivalent:*

- *$C \mathcal{R}_X A$ and $B \mathcal{R}_Y D$*
- *there exist $u, v \in \mathbf{SF}$ such that $(C, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(D, Y)[\overline{z}^X]v$.*

This lemma is proven in Appendix D.2.

Popping between specific non-terminals. Finally, our monoid even encodes popping behavior. Specifically, the matrix M inside $\varphi(\overline{z}^X)$ tells us for which non-terminals D and C, it is possible to pop z from D to C:

Lemma 5.3. *Let $z \in I^+$ a non-empty stack content, let $X \subseteq N$ and let $(B, Y, M, A, X) = \varphi(\overline{z}^X)$. The following are equivalent:*

- *$M(D, C) = \top$*
- *$C \in X$, $D \in Y$ and there exist $u, v \in (X \cup T)^*$ such that*
$$D[z] \overset{*}{\Rightarrow}_{\mathcal{G}} uCv$$
- *$C \in X$, $D \in Y$ and there exist $u, v \in \mathbf{SF}_{\mathcal{G}}$ such that*
$$(D, Y)[\overline{z}^X] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(C, X)v.$$

This lemma is proven in Appendix D.3.

6 PUMPING AND SKIPPING

We introduce two new derivation rules on the terms of $\overline{\mathcal{G}}$, and show that they preserve the downward-closure of the resulting language. Essentially, we show that, under certain conditions, sequences of more than $2|N|$ contiguous infixes mapping to the same idempotent can be extended and reduced.

This will let us abstract stack contents by forgetting everything but the first and last N elements in such sequences of infixes mapping to the same idempotent. By adapting a recent construction by Gimbert, Mascle and Totzke [27], we will show that this allows us to obtain summaries of stack contents, of size bounded by an exponential function in the size of $\mathcal{G}$.

Pump and skip derivation steps. Let us formally define the two rules. The *pump rule* is defined as follows:

$$(B, X)[\bar{z}] \rightarrow_{\text{pump}} (B, X)[z_e \bar{z}]$$

for all $(A, X) \in \overline{N}$, $e = (B, X, M, A, X) \in \mathbf{Idem}(\mathbb{M}) \setminus \{\mathbf{0}_{\mathbb{M}}, \mathbf{1}_{\mathbb{M}}\}$, $z_e \in \overline{I}^+$ with $\varphi(z_e) = e$, and $\bar{z} \in \overline{I}^*$.

The *skip rule* is first defined on stack contents:

$$\bar{z}' u_1 \cdots u_N z_e u_1 \cdots u_N \bar{z} \rightarrow_{\text{skip}} \bar{z}' u_1 \cdots u_N \bar{z}$$

for all $\bar{z}', u_1, \ldots, u_N, z_e, \bar{z} \in \overline{I}^*$ and $e \in \mathbf{Idem}(\mathbb{M}) \setminus \{\mathbf{0}_{\mathbb{M}}\}$ such that $\varphi(u_1) = \cdots = \varphi(u_N) = \varphi(z_e) = e$ (where the u_i are in one-to-one correspondence with N; we use the subscript N instead of $|N|$ for convenience). We then extend it to terms naturally: $(A, X)[\bar{z}] \rightarrow_{\text{skip}} (A, X)[\bar{z}']$ whenever $\bar{z} \rightarrow_{\text{skip}} \bar{z}'$. Observe that the skip rule erases symbols *(potentially deep) inside the stack*, not just at the top. Nevertheless, we will see that allowing the skip rule does not extend the language beyond its downward closure.

Both the pump rule and skip rules are extended to sentential forms as expected: $u \rightarrow_{\text{pump}} u'$ if u' is obtained by applying $\rightarrow_{\text{pump}}$

to a term in u, and the same goes for $\rightarrow_{\text{skip}}$. We write $\Rightarrow_{\text{pump, skip}, \overline{\mathcal{G}}}$ for the union of the three relations $\Rightarrow_{\overline{\mathcal{G}}}$, $\rightarrow_{\text{pump}}$ and $\rightarrow_{\text{skip}}$, and $\overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}}$ for its reflexive transitive closure. We may thus define

$$\mathsf{L}_{\text{pump,skip}}(\overline{\mathcal{G}}) = \{w \in T^* \mid (S, \mathbf{U}) \overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}} w\}.$$

Pump and skip are harmless. We now show that these additional rules do not extend our language beyond its downward closure:

Proposition 6.1. $\mathsf{L}_{\text{pump,skip}}(\overline{\mathcal{G}}) \subseteq \mathsf{L}(\overline{\mathcal{G}})\downarrow$

The full proof is presented in Appendix E.1 and we sketch the ideas here. For the pump rule this is not hard. It follows from the definition of the product of $\mathbb{M}$ that if $e = (B, X, M, A, X)$ is an idempotent, then $B \mathrel{\mathcal{R}_X} A$, i.e., there must be a derivation from (B, X) to some $u(A, X)v$. By erasing u and v (thanks to downward closure and productiveness), from $(B, X)[\bar{z}]$ we can reach $(A, X)[\bar{z}]$, whence we can reach $(B, X)[z_e\bar{z}]$ (where $\varphi(z_e) = e$) thanks to Lemma 5.2. Hence, any terminal word derived from $(B, X)[z_e\bar{z}]$ is a subword of a terminal word derived from $(B, X)[\bar{z}]$ (note that the productiveness of $\overline{\mathcal{G}}$ is essential here).

Eliminating skip. Replacing applications of the skip rule requires more work. In an indexed grammar, a symbol is pushed once on the stack, but may be popped on multiple branches of a derivation. To tackle this issue, we make the following observations, illustrated by Figure 1. When a sequence $u \in \overline{I}^*$ with $\varphi(u) = (B, Y, M, A, X)$ is popped along a branch starting with a non-terminal (D, Y), the resulting non-terminal (C, X) (after popping u) must satisfy $M(D, C) = \top$ (see Lemma 5.3).

Now suppose we are popping a sequence of infixes $u_1 \cdots u_N$, with $\varphi(u_1) = \cdots = \varphi(u_N) = e = (B, X, M, A, X) \in \mathbf{Idem}(\mathbb{M})$, along a branch starting with the non-terminal (A_0, X). Consider, for $i = 1, \ldots, |N|$, the non-terminal (A_i, X) obtained along that branch right after popping $u_1 \cdots u_i$. By Lemma 5.3, it follows that $M(A_i, A_{i+1}) = \top$ for $i = 0, \ldots, |N|$. Since e is idempotent, $MM = M$, and we can then apply the following elementary lemma.

Lemma 6.2. *Let $M \in \mathbb{B}^{N \times N}$ a matrix. If $MM = M$ and we have terms $A_0, \ldots, A_N \in N$ such that $M(A_i, A_{i+1}) = \top$ for all $i = 1, \ldots, |N| - 1$, then there exists i such that $M(A_i, A_i) = \top$.*

Proof. By the pigeonhole principle, there exist $i < j$ such that $A_i = A_j$. Since M is idempotent, $M^{j-i} = M$. Hence, we have

$$M(A_i, A_i) = M^{j-i}(A_i, A_j) = \bigwedge_{l=i}^{j-1} M(A_l, A_{l+1}) = \top. \qquad \square$$

Thus, one of the A_i is such that $M(A_i, A_i) = \top$ (note that i may depend on the branch). This means that for all $\bar{z} \in \overline{I}^*$ with $\varphi(\bar{z}) = e$, we have a derivation from $(A_i, X)[\bar{z}]$ to $u(A_i, X)u'$ for some $u, u' \in \mathbf{SF}$ which can be erased since we are only interested in the downward closure (and because of the productiveness property). Hence, (A_i, X) can "erase" an arbitrary sequence of infixes mapping to e at the top of its stack. The skip rule is obtained, roughly speaking, by erasing the sequence $u_{i+1} \cdots u_N z_e u_1 \cdots u_i$ at the appropriate (A_i, X) in each branch of a derivation. Since we are sure to encounter, when popping a sequence of $|N|$ infixes mapping to e, such a non-terminal on every branch of the derivation (i.e. which can pop such infixes at will), we are able to eliminate all applications of the skip rule (see Appendix E.1 for the full proof).

7 SUMMARIZING STACK CONTENTS

We have shown that adding the pump rule and skip rule to our annotated indexed grammar does not change its downward closure. Those rules give us a lot of flexibility on the infixes of the stack mapping to idempotents: the pump rule lets us extend them, and the skip rule reduce them.

We wish to compress stack contents into bounded summaries, where such sequences of infixes are abstracted away, by remembering only the first and last $|N|$ of them. A natural candidate for this task is Simon's factorization forest theorem [52]. It gives us a way to evaluate a word in a finite monoid via a tree of bounded height, using binary products and arbitrary iterations of idempotent elements. If we only care about the first and last $|N|$ elements in a sequence of idempotent infixes, we can cut out from this tree all the nodes corresponding to the remaining idempotent infixes.

Two problems remain: First, Simon's theorem gives a linear bound for the height of the tree in the monoid size. In our case this would yield summaries of doubly exponential size, and a quadruply exponential upper bound on an NFA for the downward closure, while our lower bound is only triply exponential. To solve this, we dive deeper into the structure of the monoid, utilizing results by Jecker [34] which let us cut our monoid into polynomially many layers. Within each layer, we decompose the word by reading it from right to left and compressing sequences of idempotent infixes.

The other problem is that to simulate the indexed grammar with a CFG, we use a version of the push operation on the compressed words: from the compressed version of a word z and a letter x, we need to be able to compute the compressed version of xz. This is not a property of Simon's theorem, at least not in its original formulation. One way to circumvent this problem was through the introduction of *forward Ramsey splits* [21], a weaker version of factorizations which still detects sequences of idempotent infixes. As a matter of fact, Jecker's results have been applied to improve computational bounds on those splits [42]. It may be interesting to see if one can obtain a form of summaries from such splits, by losing information to obtain a bounded object. Here, however, we do not rely on these, but provide an elementary construction which is computed deterministically by reading the word right-to-left.

We rely on a recent construction of such summaries presented in [27]. Our exposition is self-contained, however, since we use different notation and our summaries need to be constructed in a slightly different manner. Specifically, theirs need to remember only the first and last infixes, while we need the last $|N|$, and theirs are constructed as trees of bounded height, bottom-up, while we need to build ours from right to left along the stack content.

Green's relations. We begin with some notions from semigroup theory. Let $(\mathbf{M}, \cdot, \mathbf{1_M})$ be a finite monoid. We define the usual Green relations $\mathcal{J}, \mathcal{L}, \mathcal{R}, \mathcal{H}$ on $\mathbf{M}$, starting with the following quasi-orders:

- $x \leq_{\mathcal{J}} y$ if there exist $a, b \in \mathbf{M}$ such that $x = a \cdot y \cdot b$
- $x \leq_{\mathcal{L}} y$ if there exist $a \in \mathbf{M}$ such that $x = a \cdot y$
- $x \leq_{\mathcal{R}} y$ if there exist $b \in \mathbf{M}$ such that $x = y \cdot b$
- $x \leq_{\mathcal{H}} y$ if $x \leq_{\mathcal{L}} y$ and $x \leq_{\mathcal{R}} y$

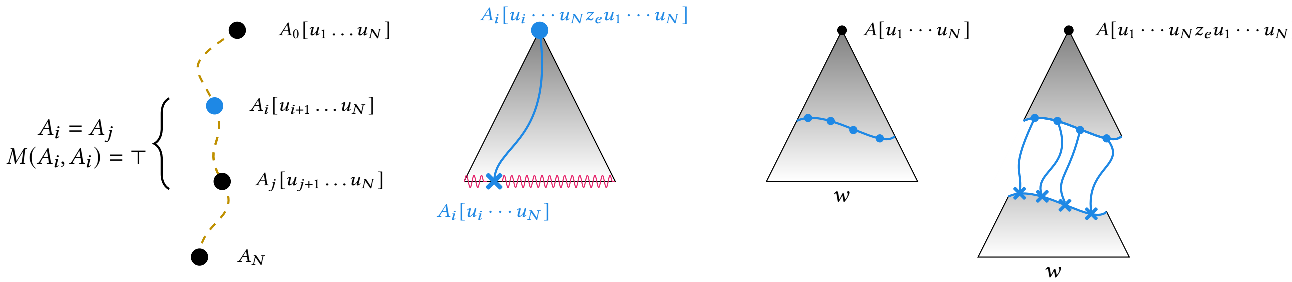

Figure 1: The idea behind the skip rule is that if we have a derivation from some non-terminal A in which we pop N consecutive infixes $u_1 \dots u_N$ mapping to the same idempotent $e = (B, X, M, A, X)$, then along every branch we must have a node of the form $A_i[u_{i+1} \cdots u_N]$ with $M(A_i, A_i) = \top$. This means that for any word mapping to e, there is a derivation from A_i popping this word, and with the resulting sentential form containing A_i. The rest of the sentential form can be erased since we are interested in the downward closure. This non-terminal can be used to "skip" the infix $u_{i+1} \cdots u_N z_e u_1 \cdots u_i$, which maps to e. Hence, we can turn a derivation from $A[u_1 \cdots u_N]$ into one from $A[u_1 \cdots u_N z_e u_1 \cdots u_N]$ by popping the right infix along every branch.

The relations $\mathcal{J}, \mathcal{L}, \mathcal{R}, \mathcal{H}$ are the equivalence relations induced by those quasi-orders: for each $\mathcal{X} \in \{\mathcal{J}, \mathcal{L}, \mathcal{R}, \mathcal{H}\}$ we define $\mathcal{X} = \leq_{\mathcal{X}} \cap \geq_{\mathcal{X}}$, as well as $<_{\mathcal{X}} = \leq_{\mathcal{X}} \setminus \geq_{\mathcal{X}}$. We do not include the monoid $\mathbf{M}$ in the notation since it will always be clear from the context.

$\mathcal{J}$-length and $\mathcal{J}$-depth. The *regular $\mathcal{J}$-length* [4] of $\mathbf{M}$, denoted $\mathcal{JL}(\mathbf{M})$, is defined as

$$\mathcal{JL}(\mathbf{M}) = \sup\{k \in \mathbb{N} \mid \exists e_1 >_{\mathcal{J}} \cdots >_{\mathcal{J}} e_k \in \mathbf{Idem}(\mathbf{M})\}.$$

Definition 7.1. The *regular $\mathcal{J}$-depth* (or just depth) of an element $x \in \mathbf{M}$ is the maximal number d such that there exist idempotents $e_1, \dots, e_d \in \mathbf{Idem}(\mathbf{M})$ such that $e_1 >_{\mathcal{J}} \cdots >_{\mathcal{J}} e_d >_{\mathcal{J}} x$.

We write $\mathbf{depth}(x)$ for the regular $\mathcal{J}$-depth of an element $x \in \mathbf{M}$. We extend this notation to sequences of elements: Given a sequence of elements $u \in \mathbf{M}^*$, we write $\mathbf{depth}(u)$ instead of $\mathbf{depth}(\psi_{\mathbf{M}}(u))$.

Remark 7.1. *Throughout this section, we will often use the observation that for all $x, y \in \mathbf{M}$, $\mathbf{depth}(x \cdot y) \geq \max(\mathbf{depth}(x), \mathbf{depth}(y))$. This is because of the definition of $\mathcal{J}$, since $x \cdot y \leq_{\mathcal{J}} x$ and $x \cdot y \leq_{\mathcal{J}} y$.*

Remark 7.2. *Observe that in the case of the stack monoid $\mathbb{M}$, we have $x <_{\mathcal{J}} \mathbf{1}_{\mathbb{M}}$ for every $x \in \mathbb{M} \setminus \{\mathbf{1}_{\mathbb{M}}\}$, because if $y, z \in \mathbb{M} \setminus \{\mathbf{1}_{\mathbb{M}}\}$, then $y \cdot z \neq \mathbf{1}_{\mathbb{M}}$. As a consequence, $\mathbf{depth}(\mathbf{1}_{\mathbb{M}}) = 0$, whereas $\mathbf{depth}(x)$ is in $[1, \mathcal{JL}(\mathbb{M})]$ for all $x \in \mathbb{M} \setminus \{\mathbf{1}_{\mathbb{M}}\}$. The upper bound follows from the definition of $\mathcal{JL}(\mathbb{M})$. Positivity of $\mathbf{depth}(x)$ is due to $x <_{\mathcal{J}} \mathbf{1}_{\mathbb{M}}$.*

Summaries. We now define the main object of this section, *summaries.* As mentioned above, they can be viewed as compressions of stack words (with some information loss). Syntactically, these are sequences of (sequences of (...)) sequences, where the nesting depth depends on the depth of $\varphi(w)$, where w is the stack word to be compressed. These sequences contain letters from $\bar{I}$, but also a special letter e^+ for each idempotent $e \in \mathbf{Idem}(\mathbb{M}) \setminus \{\mathbf{1}_{\mathbb{M}}\}$. Intuitively, e^+ represents a sequence of infixes that evaluate to e.

Summaries will have a well-defined image under φ, and the summary of $w \in \bar{I}^*$ will agree with w under φ. To this end, we set $\varphi(e^+) = e$ for all such e. We also extend φ to $(\bar{I}^*)^*$ by defining the image of a sequence of words $z_1 \dots z_n \in (\bar{I}^*)^*$ as the image of their concatenation, and so on for sequences of (sequences of (...)) sequences. To simplify notation, given a sequence of stack symbols $z \in \bar{I}^*$, we also write $\mathbf{depth}(z)$ instead of $\mathbf{depth}(\varphi(z))$, and we extend this notation to sequences of sequences of (sequences of (...)) sequences naturally.

Formally, a 0-summary is just the empty word, and a $(d+1)$-summary is a sequence of (i) elements of $\bar{I} \cup \{e^+ \mid e \in \mathbf{Idem}(\mathbb{M})\}$ and of (ii) d-summaries. As explained above, $\varphi(\sigma)$ and $\mathbf{depth}(\sigma)$ are well-defined for all summaries σ.

Definition 7.2. A *0-summary* is simply the empty word ε. Let $d \in [1, \mathcal{JL}(\mathbb{M})]$.

- A *d-atom* is a word $(f, X)\sigma$ with $(f, X) \in \bar{I}$ and σ a d'-summary for some $d' < d$, such that $\mathbf{depth}((f, X)\sigma) = d$.
- A *d-block* is a sequence of the form $u_1 \cdots u_N e^+ v_1 \cdots v_N w$ where $u_1, \dots, u_N, v_1, \dots, v_N$ and w are sequences of d-atoms and $e \in \mathbf{Idem}(\mathbb{M}) \setminus \{\mathbf{1}_{\mathbb{M}}\}$, such that $\varphi(u_i) = \varphi(v_i) = e$ for all i and $\mathbf{depth}(u_1 \cdots u_N e^+ v_1 \cdots v_N w) = d$.
- A *d-summary* is a sequence of the form $\sigma' u B_1 \dots B_k$ where σ' is a d'-summary for some $d' < d$, u is a word of d-atoms, and $B_1, \dots, B_k$ are d-blocks with $\mathbf{depth}(\sigma' u B_1 \dots B_k) = d$.

A *summary* is a d-summary for some d. Observe that by definition, a d-summary σ has $\mathbf{depth}(\sigma) = d$. The set of summaries is denoted **Summaries**.

Intuition. Let us give some intuition on the definition of summaries. Note that when processing a string from right to left, the depth of the growing suffix increases monotonically (Remark 7.1). We read the word from right to left since this is how our stacks are built. Here, a d-atom represents a suffix-minimal sequence of depth d: It has depth d, but when removing the left-most letter, the remainder has lower depth. Hence, that remainder is given as a d'-summary for some $d' < d$. A d-block can be thought of as a sequence of atoms where some of them (which mapped to e under φ) were (lossily) compressed into a single letter e^+: This compression is only allowed if the compressed part had been surrounded by $|N|$ sequences of d-atoms (on each side) that also map to e, which are still present in $u_1, \dots, u_N, v_1, \dots, v_N$. Finally, a d-summary consists

[4] Called regular $\mathcal{D}$-length in [34]. For finite monoids, $\mathcal{D} = \mathcal{J}$, and we use $\mathcal{J}$ here since it is more common. Furthermore, it is defined differently in [34], but a proof that both definitions are equivalent can be found in the long version [35][Appendix B].

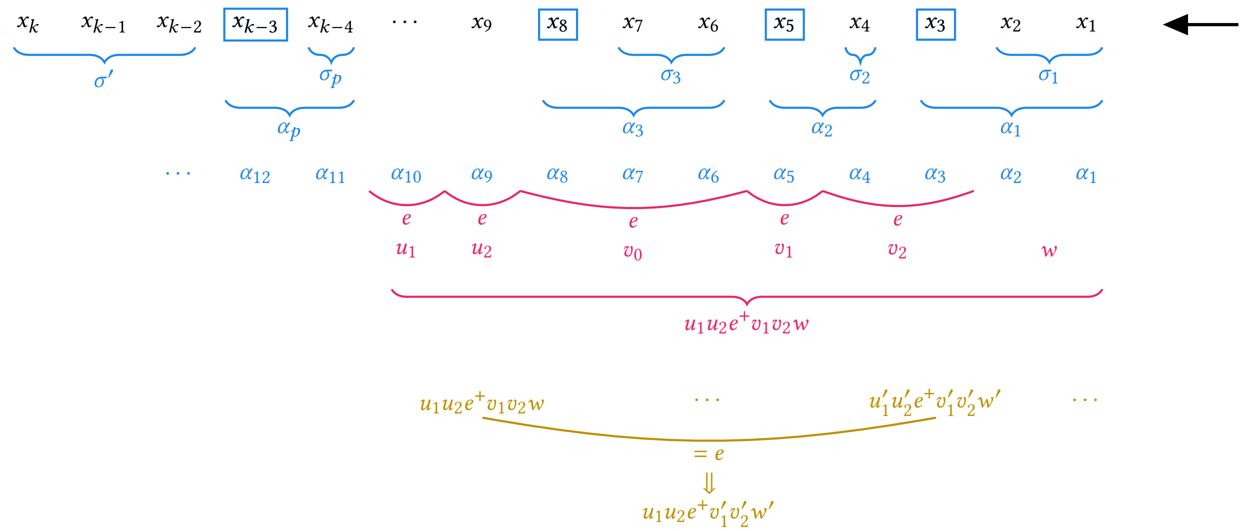

Figure 2: A visual of how summaries are constructed. Say we are given a word z of regular $\mathcal{J}$-depth d. We read it from right to left. We cut it into d-atoms by iteratively taking the smallest suffix of regular $\mathcal{J}$-depth d. Its summary consists of its leftmost letter and a d'-summary for its tail of depth $d' < d$. In parallel, we read the resulting sequence of atoms $\cdots\alpha_3\alpha_2\alpha_1$ from right to left. Every time we find an infix with a prefix made of 2N+1 infixes mapping to the same idempotent, we turn it into a d-block $u_1u_2e^+v_1v_2$. Finally, whenever we have two blocks corresponding to an idempotent e and such that the infix between their middle parts also evaluates to e, we merge them into one d-block.

of d-blocks $B_1, \ldots, B_k$, some remaining d-atoms (that did not permit compression into d-blocks), and a lower-depth part represented by a d'-summary.

Compressing stacks into summaries. Let us describe how a stack $\bar{z} \in \bar{I}^*$ is compressed into its (unique) summary. For a sequence (i.e. word over a finite alphabet or a summary) of length ≥ 1, its *tail* is obtained by removing its leftmost element. Given a word $\bar{z}$ with $\mathbf{depth}(\bar{z}) = d$, we proceed in three stages, illustrated in Fig. 2.

Stage I: Splitting into d-atoms. First we split the word into suffix-minimal infixes of depth d, from right to left, and a residual prefix of some depth $d' < d$. By suffix-minimality, the tail of each of these depth-d infixes has depth $< d$. We thus turn each of these depth-d infixes into a d-atom by computing recursively a summary for its lower-depth tail. Afterwards, the residual prefix of depth $d' < d$ is recursively turned into a summary.

Stage II: Compression into blocks. Then, we look at the sequence of d-atoms and their values in $\mathbb{M}$. Going from right to left, we look for sequences of $2|N| + 1$ infixes $u_1 \cdots u_N v_0 v_1 \cdots v_N$ all evaluating to the same idempotent. Whenever we find such a pattern we turn the current infix (i.e. the pattern with, possibly, a suffix w) into a d-block by replacing the middle part v_0 by e^+. There may be a remainder u at the left end of the d-atom sequence with no such pattern; we write it explicitly in the summary.

Stage III: Merging blocks. Finally, we look at the sequence of resulting d-blocks. We go again from right to left, this time looking for pairs of blocks corresponding to the same idempotent e, and so that the infix between their e^+ markers also evaluates to e. This lets us merge them into a single block, obtained by abstracting their e^+ markers and everything in between in a single e^+.

Updating a summary. A key feature of summaries is that given the summary σ of a stack $\bar{z} \in \bar{I}^*$ and an additional letter $\bar{x} \in \bar{I}$, we can compute the summary of $\bar{x}\bar{z}$. This is needed when simulating pushes. We now describe the relevant operation,

$$\mathsf{push}(\_ \blacktriangleright \_) : \bar{I}^* \times \mathbf{Summaries} \to \mathbf{Summaries}.$$

Given a depth $d \in [0, \mathcal{JL}(\mathbb{M})]$, an element $(f, X) \in \bar{I}$ and an d-summary σ, we define $\mathsf{push}((f, X) \blacktriangleright \sigma)$ as follows. If $d = 0$ then $\sigma = \varepsilon$ and $\mathsf{push}((f, X) \blacktriangleright \sigma) = (f, X)$. If $d > 0$ then σ is of the form $\sigma' u B_1 \ldots B_k$.

(1) If $\mathbf{depth}((f, X)\sigma) > d$ then let $d_+ = \mathbf{depth}((f, X)\sigma)$. We set $\mathsf{push}((f, X) \blacktriangleright \sigma)$ as the d_+-summary made of a single d_+-atom $(f, X)\sigma$
(2) Otherwise $\mathbf{depth}((f, X)\sigma) \leq d$. In fact, since $\mathbf{depth}((f, X)\sigma)$ is at least $\mathbf{depth}(\sigma) = d$, we even know $\mathbf{depth}((f, X)\sigma) = d$.
  (a) If $\mathbf{depth}((f, X)\sigma') < d$ then $\mathsf{push}((f, X) \blacktriangleright \sigma) = (\mathsf{push}((f, X) \blacktriangleright \sigma'))uB_1 \ldots B_k$
  (b) Otherwise, $\mathbf{depth}((f, X)\sigma') \geq d$. Then we even know $\mathbf{depth}((f, X)\sigma') = d$, because $\mathbf{depth}((f, X)\sigma')$ is at most $\mathbf{depth}((f, X)\sigma) \leq d$. Hence, $(f, X)\sigma'$ is a d-atom.
    (i) If the sequence of d-atoms $((f, X)\sigma')u$ is of the form $u_1 \ldots u_N v_0 v_1 \ldots v_N w$ with $\varphi(v_0) = \varphi(u_i) = \varphi(v_i) = e$ for all $i \geq 1$, for some $e \in \mathbf{Idem}(\mathbb{M})$, then define the d-block $B = u_1 \ldots u_N e^+ v_1 \ldots v_N w$ (note that e^+ replaces the middle infix v_0).
      (A) Suppose there is j such that B_j is of the form

$$u'_1 \ldots u'_N e^+ v'_1 \ldots v'_N w'$$

and the infix $v_1 \ldots v_N w B_1 \ldots B_{j-1} u'_1 \ldots u'_N$ also evaluates to e under φ. In this case, we merge

everything up to B_j into a single block: We define $\mathsf{push}((f,X) \blacktriangleright \sigma)$ to be $B'B_jB_{j+1}\dots B_k$, with $B' = u_1 \dots u_N e^+ v'_1 \dots v'_N w'$. When there are multiple such j, we pick the maximal one.
(B) Otherwise $\mathsf{push}((f,X) \blacktriangleright \sigma) = BB_1 \dots B_k$
(ii) Otherwise $\mathsf{push}((f,X) \blacktriangleright \sigma) = u'B_1 \dots B_k$ with $u' = ((f,X)\sigma')u$

The definition is extended inductively to $\overline{I}^*$ by

$$\mathsf{push}((f,X)\bar{z} \blacktriangleright \sigma) = \mathsf{push}((f,X) \blacktriangleright \mathsf{push}(\bar{z} \blacktriangleright \sigma)).$$

Intuitively, the $\mathsf{push}(\_ \blacktriangleright \_)$ operation simultaneously executes stages I, II and III described above. Upon adding a new letter (f,X), it checks whether it yields a new atom (Stage I). Case (1) happens when a new atom is generated because the depth of the summary is increased by adding (f,X), case (a) when no new atom is generated at depth d. In case (b), we have a new atom $(f,X)\sigma'$. We thus have to check whether this new atom yields a new block (Stage II). If it does, we are in case (i), and we then have to apply Stage III, i.e., see if we can merge this new block with another one. In case (A) we can; in case (B) we cannot.

Popping from summaries. We also define the inverse relation: for $\bar{z} \in \overline{I}^*$ and $\sigma, \sigma' \in \mathbf{Summaries}$, we have $\sigma \in \mathsf{pop}(\bar{z} \blacktriangleleft \sigma')$ if and only if $\sigma' = \mathsf{push}(\bar{z} \blacktriangleright \sigma)$. Note that $\mathsf{push}(\_ \blacktriangleright \_)$ is a function while $\mathsf{pop}(\_ \blacktriangleleft \_)$ is a relation. Intuitively, $\mathsf{push}(\_ \blacktriangleright \_)$ makes compressions, by creating and merging blocks, thereby losing information. Meanwhile, $\mathsf{pop}(\_ \blacktriangleleft \_)$ may revert those compressions, and thus requires non-determinism.

Bounding summaries. The *size* of a d-summary (resp. d-atom, d-block) is defined recursively as the sum of the sizes of its elements, the size of a single letter being 1. Although summaries can have a priori unbounded sizes, the ones we will use in our context-free grammar will be of the form $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ for some $\bar{z} \in \overline{I}^*$. For such summaries, we can prove a size bound:

Lemma 7.3. *For every $\bar{z} \in \overline{I}^*$, the summary $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ is of size at most exponential in $|N|$.*

To prove this, we rely on two results of Jecker [34], recalled in Appendix F.1:

- One guarantees that in a word of elements of exponential length in $\mathcal{JL}(\mathbb{M})$ we can find large sequences of consecutive infixes that all map to the same idempotent (Theorem F.3).
- Another bounds the regular $\mathcal{J}$-length of a Boolean matrix monoid by a polynomial in its dimension (Theorem F.4).

For Lemma 7.3, we show that when viewing when viewing d-atoms and d'-summaries, for $d' < d$, as individual letters, the size of a d-summary is bounded by an exponential in $|N|$. The overall bound on summaries then results from the product of those (polynomially many) exponential functions.

8 BUILDING THE CONTEXT-FREE GRAMMAR

We now construct a context-free grammar $C_{\mathcal{G}}$ that over-approximates $\mathsf{L}(\mathcal{G})$, but remains within its downward closure. It will thus satisfy $\mathsf{L}(C_{\mathcal{G}})\!\downarrow = \mathsf{L}(\mathcal{G})\!\downarrow$ and enable us to compute an NFA for the latter.

Definition of the grammar. Essentially, $C_{\mathcal{G}}$ is obtained from $\overline{\mathcal{G}}$ by replacing stack contents with their summaries.

We first restrict the set of summaries to those that can actually result from a derivation of the indexed grammar. A summary σ is *feasible* if $\sigma = \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ for some feasible $\bar{z} \in \overline{I}^*$. Note that this implies $\varphi(\sigma) \neq \mathbf{0}_{\mathbb{M}}$ by Lemma 5.1. The non-terminals will be triples (A, X, σ) where $(A,X) \in \overline{N}$, and σ is a feasible summary. We call such triples *feasible*. The set of feasible triples is denoted $\mathbf{FT}$.

Define the following context-free grammar $C_{\mathcal{G}}$: Its set of non-terminals is $\mathbf{FT}$, with $(S, \mathbf{U}, \varepsilon)$ the initial one. The set of terminal symbols is T. The productions directly mimic the productions in $\overline{\mathcal{G}}$, except that push and pop productions are simulated by the $\mathsf{push}(\_ \blacktriangleright \_)$ and $\mathsf{pop}(\_ \blacktriangleleft \_)$ relations on summaries:

- If $(A,X) \to w \in \overline{P}$ then $(A,X,\sigma) \to w$, for all feasible σ.
- If $(A,X) \to (B,X)(C,X) \in \overline{P}$ then $(A,X,\sigma) \to (B,X,\sigma)(C,X,\sigma)$, for all feasible σ.
- If $(A,X) \to (B,Y)(f,X) \in \overline{P}$ then $(A,X,\sigma) \to (B,Y,\mathsf{push}((f,X) \blacktriangleright \sigma))$, for all summary σ so that σ and $\mathsf{push}((f,X) \blacktriangleright \sigma)$ are feasible.
- If $(A,Y)(f,X) \to (B,X) \in \overline{P}$ then $(A,Y,\sigma) \to (B,X,\sigma')$, whenever σ, σ' are feasible and $\sigma' \in \mathsf{pop}((f,X) \blacktriangleleft \sigma)$.

Abusing terminology slightly, we call production rules of the third and fourth type *pushes* and *pops*, respectively (even though they are standard context-free productions).

Correctness of the construction. The key property of $C_{\mathcal{G}}$ is that it has the same downward closure as $\mathsf{L}(\mathcal{G})$.

Theorem 8.1. $\mathsf{L}(C_{\mathcal{G}})\!\downarrow = \mathsf{L}(\mathcal{G})\!\downarrow$.

One of the directions is quite easy: we can simply show that the language of $C_{\mathcal{G}}$ contains that of $\overline{\mathcal{G}}$. This is natural as $C_{\mathcal{G}}$ is built as an over-approximation of $\overline{\mathcal{G}}$. We can turn a derivation of $\overline{\mathcal{G}}$ into one of $C_{\mathcal{G}}$ by replacing every stack content with its summary. The formal proof is presented in Appendix G.1.

Proposition 8.2. $\mathsf{L}(\overline{\mathcal{G}}) \subseteq \mathsf{L}(C_{\mathcal{G}})$.

Simulating $C_{\mathcal{G}}$ with pumps and skips. For the inclusion $\mathsf{L}(C_{\mathcal{G}})\!\downarrow \subseteq \mathsf{L}(\mathcal{G})\!\downarrow$, we will show that every derivation in $C_{\mathcal{G}}$ can be simulated by pumps and skips:

Proposition 8.3. $\mathsf{L}(C_{\mathcal{G}}) \subseteq \mathsf{L}_{\mathsf{pump,skip}}(\overline{\mathcal{G}})$.

Indeed, by Proposition 6.1, this implies that $\mathsf{L}(C_{\mathcal{G}}) \subseteq \mathsf{L}(\overline{\mathcal{G}})\!\downarrow$ and thus $\mathsf{L}(C_{\mathcal{G}})\!\downarrow \subseteq \mathsf{L}(\overline{\mathcal{G}})\!\downarrow = \mathsf{L}(\mathcal{G})\!\downarrow$, establishing Theorem 8.1.

We shall prove Proposition 8.3 by simulating summaries by actual stacks. Recall that in the summary $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ that compresses the stack $\bar{z}$, the letters e^+ represent a sequence of e-words, where we call $\overline{w} \in \overline{I}^*$ an *e-word* if $\varphi(\overline{w}) = e$. The other letters in σ are taken directly from $\bar{z}$. Therefore, the unfolding reverses this: It replaces e^+ by e-words and leaves the other letters unchanged.

Intuitively, our strategy for replacing e^+ is as follows. We replace e^+ by a sequence of all e-words that might be needed to sustain the remainder of the derivation (specifically, all its pop operations). In a particular branch of the derivation, those e-words that are not needed can always be cleared using the skip rule. On the other hand, the pump rule allows us to justify introducing all these e-words.

Unfoldings. Instead of tailoring the stack $\bar{z}$ simulating σ to a specific derivation, we will construct a "canonical" stack word $\bar{z}$ for σ that only depends on the number of pops in that derivation. We will call this canonical stack word the "p-unfolding" of σ, where p is the number of pops it is designed to sustain. This means, intuitively, each e^+ is replaced by a large enough concatenation of e-words so that any sequence of p pops can be executed.

Just to choose the order in which those e-words appear, we impose some arbitrary total order (such as a length-lexicographical ordering) on the set of summaries, which we denote $\trianglelefteq$.

Definition 8.4. Let $p \in \mathbb{N}$ and σ be a d-summary. The *p-unfolding* of σ, denoted $\mathsf{unf}_p(\sigma)$ is defined inductively w.r.t. d and p as follows. For $d = 0$, the p-unfolding of a 0-summary (i.e., ε) is ε.

- The p-unfolding of a d-atom $(f, X)\sigma'$ is $(f, X)\mathsf{unf}_p(\sigma')$.
- The p-unfolding of a sequence of d-atoms $\alpha_1 \cdots \alpha_m$ is defined as $\mathsf{unf}_p(\alpha_1) \cdots \mathsf{unf}_p(\alpha_m)$.
- The p-unfolding of a d-block $B = u_1 \dots u_N e^+ v_1 \dots v_N w$ is

$$z^u z^v ((f, X)\mathsf{unf}_{p-1}(\sigma_1)) \cdots ((f, X)\mathsf{unf}_{p-1}(\sigma_r)) z^v \mathsf{unf}_p(w),$$

  where:
  - (f, X) is the first symbol of B, that is, the stack symbol such that the first d-atom of u_1 is of the form $(f, X)\sigma'$.
  - $z^u = \mathsf{unf}_p(u_1 \dots u_N)$ and $z^v = \mathsf{unf}_p(v_1 \dots v_N)$
  - if $p > 0$, then $(\sigma_i)_{1 \le i \le r}$ is the family of all feasible summaries σ' for which $\mathsf{push}((f, X) \blacktriangleright \sigma')$ equals $u_1 \dots u_N e^+ v_1 \dots v_N$, ordered according to $\trianglelefteq$.
  - if $p = 0$, then $r = 0$, i.e., the p-unfolding of B is simply $z^u z^v \mathsf{unf}_p(w)$.
- The p-unfolding of a d-summary $\sigma' u B_1 \dots B_m$ is defined as $\mathsf{unf}_p(\sigma')\mathsf{unf}_p(u)\mathsf{unf}_p(B_1) \cdots \mathsf{unf}_p(B_m)$.

Since the unfolding is obtained by replacing each e^+ in a summary by a concatenation of words with image e (and all other letters are unchanged), the unfolding has the same image under φ as σ:

Remark 8.1. *For every summary σ and every $p \in \mathbb{N}$, we have $\varphi(\mathsf{unf}_p(\sigma)) = \varphi(\sigma)$.*

Removing excess e-words. When choosing a stack word to simulate a given summary σ, we pick the p-unfolding, where p is the total number of pops in the entire derivation. However, some branches will apply less than p pops. The following lemma is therefore crucial: It allows us to get rid of excess e-words that are not needed on less pop-heavy branches, while maintaining the invariant that we have unfoldings on the stack:

Lemma 8.5. *For each σ and $p \ge 1$: $\mathsf{unf}_p(\sigma) \xrightarrow{*}_{\mathsf{skip}} \mathsf{unf}_{p-1}(\sigma)$.*

Note that it is not possible to simply skip e-words at will, since that requires equality of some infixes in the stack. Nevertheless, unfoldings are carefully constructed to allow Lemma 8.5. For example, the fact that we always follow a uniform order $\trianglelefteq$ on summaries is key. A full proof can be found in Appendix G.3.

Simulating pushes using unfoldings. Let us now show how unfoldings are used to mimic derivations of $C_{\mathcal{G}}$ in $\overline{\mathcal{G}}$, with pumps and skips. First, note that all productions of $C_{\mathcal{G}}$ that are not pushes and pops have direct counterparts in $\overline{\mathcal{G}}$. Suppose we want to simulate a push, say a rule $(A, X, \sigma_1) \to (B, Y, \sigma_2)$ of $C_{\mathcal{G}}$, induced by a push rule $(A, X) \to (B, Y)(f, X)$ in $\overline{\mathcal{G}}$. If (A, X, σ_1) is simulated by $(A, X)[\mathsf{unf}_p(\sigma_1)]$, then we can use $(A, X) \to (B, Y)(f, X)$ first in $\overline{\mathcal{G}}$. But then the stack is $(f, X)\mathsf{unf}_p(\sigma_1)$, rather than an unfolding of σ_2. The following lemma tells us that, using pump and skip, we can replace $(f, X)\mathsf{unf}_p(\sigma_1)$ by $\mathsf{unf}_p(\sigma_2)$, which will then enable us to continue the simulation.

Lemma 8.6. *Let $p \in \mathbb{N}$, let $(A, X, \sigma_1) \to (B, Y, \sigma_2)$ be a rule of $C_{\mathcal{G}}$ with $\sigma_2 = \mathsf{push}((f, X) \blacktriangleright \sigma_1)$. We have*

$$(B, Y)[(f, X)\mathsf{unf}_p(\sigma_1)] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip}} (B, Y)[\mathsf{unf}_p(\sigma_2)].$$

This lemma is proved in Appendix G.4; in fact, only $\to_{\mathsf{pump}}$ and $\to_{\mathsf{skip}}$ are needed.

Simulating pops using unfoldings. Pop steps are more difficult. In a production $(A, X, \sigma_1) \to (B, Y, \sigma_2)$ in $C_{\mathcal{G}}$ induced by a pop rule $(A, X)(f, Y) \to (B, Y)$, σ_2 is the summary of a word obtained by removing the first letter of a word compressed by σ_1. This removal might break a block centered around some $e \in \mathbf{Idem}(\mathbb{M})$, in σ_1. This means, the symbol e^+ is replaced by a concatenation of summaries. However, p-unfoldings are designed so that $\mathsf{unf}_p(\sigma_1)$ contains enough e-words for each e^+ so that using skip rules, we can remove a subset of them so that the resulting stack is precisely $(f, Y)\mathsf{unf}_{p-1}(\sigma_2)$. This is shown in the following lemma:

Lemma 8.7. *Let $p \ge 1$, let $(A, X, \sigma_1) \to (B, Y, \sigma_2)$ a rule of $C_{\mathcal{G}}$ with $\sigma_2 \in \mathsf{pop}((f, Y) \blacktriangleleft \sigma_1)$. We have*

$$(A, X)[\mathsf{unf}_p(\sigma_1)] \xrightarrow{*}_{\mathsf{skip}} (A, X)[(f, Y)\mathsf{unf}_{p-1}(\sigma_2)].$$

This is shown in Appendix G.5. Thus, to simulate this pop rule, we can first invoke Lemma 8.7 and then apply $(A, X)(f, Y) \to (B, Y)$.

Simulating the whole derivation. With Lemmas 8.6 and 8.7 in hand, Proposition 8.3 is now easy to show. Indeed, it is straightforward to simulate an entire derivation of $C_{\mathcal{G}}$ in $\overline{\mathcal{G}}$ with pump and skip rules: Simulating pushes and pops is as explained in Lemmas 8.6 and 8.7, and the other productions are immediate. The full proof can be found in Appendix G.2.

We have thus completed Proposition 8.3 and hence Theorem 8.1.

Putting it all together. With Theorem 8.1, we are prepared to prove Theorem 3.1. By Lemma 7.3, the size of a summary is bounded by an exponential in the size of $\mathcal{G}$. As a consequence, the size of $C_{\mathcal{G}}$ is at most doubly exponential in the size of $\mathcal{G}$. Since for a given context-free grammar, one can compute an exponential-sized NFA for its language's downward closure [12, Corollary 6], this yields a tripy exponentially sized NFA for $\mathsf{L}(\mathcal{G})\downarrow$, as desired in Theorem 3.1.

9 LOWER BOUNDS

In this section, we prove the lower bounds in our main results.

NFA Lower bound: Overview. We begin with Theorem 3.2. The overall goal is to have a unique complete derivation tree that is a full binary tree of doubly exponential depth: clearly, such a tree must have triply exponentially many leaves. Here, the challenge is to ensure that the paths have doubly exponential length.

A standard construction can enforce *singly* exponentially long paths: Use a height-n stack over the alphabet $\{\mathtt{0}, \mathtt{1}\}$, and then count up from $\mathtt{0}^n$ to $\mathtt{1}^n$, resulting in $2^n - 1$ steps. This works because when incrementing a binary expansion of the form $\mathtt{1}^m\mathtt{0}w$ (the least significant digit being on the left), we must replace the prefix $\mathtt{1}^m\mathtt{0}$ with $\mathtt{0}^m\mathtt{1}$. Here, we store the number $m \leq n$ in the non-terminal, so as to restore the stack height n when pushing $\mathtt{0}^m\mathtt{1}$.

Doing the same for stack height 2^n is not so easy: To restore a stack height of 2^n, we would need to remember (in the non-terminal) a number $m \leq 2^n$. In fact, enforcing a single run of length 2^{2^n} in a pushdown automaton of polynomial size is not possible: A pushdown automaton that accepts any word must also accept a word of at most exponential length.

Instead, we exploit the fact that an indexed grammar can simulate a pushdown automaton *with alternation*: We implement binary counting on a stack of height 2^n; in order to replace a prefix $\mathtt{1}^m\mathtt{0}$ with $\mathtt{0}^m\mathtt{1}$, we non-deterministically push some number of $\mathtt{0}$'s, but then use alternation to ensure that the stack height is exactly 2^n.

Step I: Checking the stack via alternation. To this end, we introduce syntactic sugar. We will use rules of the form

$$A \xrightarrow{\mathcal{A}_1,\ldots,\mathcal{A}_r} B\,, \tag{1}$$

where $\mathcal{A}_1, \ldots, \mathcal{A}_r$ are DFAs over the stack alphabet I. The rule has the same effect as $A \to B$, but it can only be applied to a term $A[z]$ if for each $i = 1, \ldots, r$, the stack z has a prefix in $\mathsf{L}(\mathcal{A}_i)$. Such rules can be implemented with only polynomial overhead: Introduce non-terminals C_i, D_i for each $i = 1, \ldots, r$ and also E_q for each state q in the DFAs $\mathcal{A}_1, \ldots, \mathcal{A}_r$ (we assume the state sets are disjoint). Then we can simulate (1) using $A \to D_1C_1$, $D_i \to D_{i+1}C_{i+1}$ for $i = 1, \ldots, r-1$, and $D_r \to B$, which split the term $A[z]$ into terms $B[z]$ and $C_1[z], \ldots, C_r[z]$. We then run $\mathcal{A}_i$ using rules $C_i \to E_{q_i}$, where q_i is the initial state of $\mathcal{A}_i$, for each i. The non-terminals E_q simulate the DFAs: for each transition (p, f, q), we have $E_pf \to E_q$. To check acceptance, we have $E_q \to \varepsilon$ for each final state q.

Step II: Implementing a binary counter in DFAs. We want to use rules (1) to check that the current stack height is 2^n, for which we construct automata $(\mathcal{A}_i)_{1\leq i\leq n}$ over some alphabet Σ_n such that: (i) Each $\mathcal{A}_i$ has two states $\mathbf{0}_i$ and $\mathbf{1}_i$, with $\mathbf{0}_i$ being initial and $\mathbf{1}_i$ the only final state, (ii) $|\Sigma_n| = n$, and (iii) the intersection $\bigcap_{i=1}^n \mathsf{L}(\mathcal{A}_i)$ contains a single word of length 2^n. The construction is simpler if we do this for $2^n - 1$ instead of 2^n, which suffices: We can introduce a fresh letter # and build automata $\mathcal{A}_i'$ with $\mathsf{L}(\mathcal{A}_i') = \mathsf{L}(\mathcal{A}_i)\#$.

The idea is simply to use $\Sigma_n = \{\mathtt{inc}_1, \ldots, \mathtt{inc}_n\}$. Each letter is an increment operation over an n-bit binary counter: $\mathtt{inc}_i$ should be read as "flip the i-th bit from 0 to 1 and all lower bits from 1 to 0". Each automaton $\mathcal{A}_i$ keeps track of the value of the i-th bit throughout that sequence of instructions.

Formally, $\mathcal{A}_i = (\{\mathbf{0}_i, \mathbf{1}_i\}, \Sigma_n, \delta_i, \mathbf{0}_i, \{\mathbf{1}_i\})$ where $\delta_i(\mathbf{0}_i, \mathtt{inc}_j)$ is defined as $\mathbf{1}_i$ if $j = i$, it is $\mathbf{0}_i$ if $j < i$, and it is undefined if $j > i$. Meanwhile, $\delta_i(\mathbf{1}_i, \mathtt{inc}_j)$ is $\mathbf{0}_i$ if $j > i$, it is $\mathbf{1}_i$ if $j < i$ and it is undefined if $i = j$. It is easy to check that there is a unique word accepted by those automata, corresponding to the only correct sequence of instructions to increment a n-bit binary counter from 0 to $2^n - 1$, which enforces a single string of length $2^n - 1$, as desired.

Step III: Constructing the indexed grammar. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ the DFAs over Σ_n built above. Since they will check for exponential stack height, but we also need to store the binary digits on the stack, we modify them slightly. For each i, the automaton $\mathcal{B}_i$ will work over the alphabet $\{\bot\} \cup (\Sigma_n \times \{\mathtt{0}, \mathtt{1}\})$ and accept exactly the words of the form $(\alpha_1, b_1) \cdots (\alpha_m, b_m)\bot$ where $\alpha_1 \cdots \alpha_m \in \mathsf{L}(\mathcal{A}_i)$. The $\bot$ letter is used to mark the bottom of the stack.

Consider the following grammar: $\mathcal{G}_n = (N_n, T, I_n, P_n, S)$ with $N_n = \{S, A, B, D, F, Z\}$, $T = \{\mathsf{a}\}$, $I_n = \{\bot\} \cup (\Sigma_n \times \{\mathtt{0}, \mathtt{1}\})$, and P_n contains the following rules:

$$\begin{array}{lll} S \to Z\bot & D \to AA & B \to Z(\alpha, \mathtt{1}) \\ Z \to Z(\alpha, \mathtt{0}) & A(\alpha, \mathtt{1}) \to A & A\bot \to F \\ Z \xrightarrow{\mathcal{B}_1,\ldots,\mathcal{B}_n} D & A(\alpha, \mathtt{0}) \to B & F \to \mathsf{a} \end{array}$$

for each $\alpha \in \Sigma_n$. The grammar works as follows. Initially, it places $\bot$ on the stack and switches to Z. A non-terminal Z will then fill the stack with $\mathtt{0}$'s, which are annotated by $\alpha \in \Sigma_n$. After pushing these, it verifies that the stack height is 2^n, by using $Z \xrightarrow{\mathcal{B}_1,\ldots,\mathcal{B}_n} D$. This D splits into two A's, where an increment is performed on the number encoded on the stack: It removes the prefix of the form $\mathtt{1}^m\mathtt{0}$ and switches to B. After this, it has to put back $\mathtt{0}^m\mathtt{1}$: To this end, it pushes a single $\mathtt{1}$ and then using Z pushes $\mathtt{0}$'s non-deterministically. It then uses $Z \xrightarrow{\mathcal{B}_1,\ldots,\mathcal{B}_n} D$ to verify that the stack height is 2^n. All this repeats until in each branch, all stack contents encode the number $2^{2^n} - 1$. This means, all terms are of the form $A[z\bot]$, where z has length 2^n and all its digits are $\mathtt{1}$'s. Each such $A[z\bot]$ is then rewritten to F, and then to a. Since the terms are duplicated before each increment (using $D \to AA$), the final number of a letters is $\exp_3(n)$, deriving $\mathsf{a}^{\exp_3(n)}$. Moreover, it is straightforward to check that $\mathsf{a}^{\exp_3(n)}$ is the only derivable word. Details are in Appendix H.1.

Computational hardness. The lower bounds for downward closure inclusion and equivalence now follow easily from Theorem 3.2 and results in [58]. In [58], the $\Delta(f)$ property of language classes is introduced. Roughly speaking, it requires simple closure properties and that for given $n \in \mathbb{N}$, one can construct in polynomial time the language $\{\mathsf{a}^{f(n)}\}$. Under additional mild assumptions, [58, Theorem 15] shows that downward closure inclusion and equivalence are co$\mathsf{NTIME}(f)$-hard for $\Delta(f)$ classes. Since all assumptions besides a small grammar for $\{\mathsf{a}^{\exp_3(n)}\}$ are easy to observe, we may conclude that the indexed languages are $\Delta(\exp_3)$ and the two problems are co-3-NEXP-hard. See Appendix H.2 for details.

DFA size. For Theorem 3.5, we adapt an idea from [12, Theorem 7], which shows a doubly exponential lower bound for downward closure DFAs for CFL. It is not difficult to translate the grammar $\mathcal{G}_n$ for $\{\mathsf{a}^{\exp_3(n)}\}$ into one for $L_n = \{uv \mid u, v \in \{\mathtt{0}, \mathtt{1}\}^* \mid |u| = |v| = \exp_3(n),\ u \neq v\}$. It is easy to see that a DFA for $L_n{\downarrow}$ requires $\exp_4(n)$ states: For distinct $u, v \in \{\mathtt{0}, \mathtt{1}\}^*$ with $|u| = |v|$, the DFA must accept uv and vu, but reject uu and vv. It therefore must enter distinct states after reading u and v. See Appendix H.3 for details.

10 CONCLUSION

We have established (asymptotically) tight bounds on the size of an automaton for the downward closure of an indexed language. We rely on an algebraic abstraction of stack contents to translate indexed grammars into context-free ones while preserving the downward closure.

REFERENCES

[1] Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. 2001. Effective Lossy Queue Languages. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2076)*, Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen (Eds.). Springer, 639–651. https://doi.org/10.1007/3-540-48224-5_53

[2] Alfred V. Aho. 1968. Indexed Grammars - An Extension of Context-Free Grammars. *J. ACM* 15, 4 (1968), 647–671. https://doi.org/10.1145/321479.321488

[3] C. Aiswarya, Pascal Baumann, Prakash Saivasan, Lia Schütze, and Georg Zetzsche. 2026. Bounded Treewidth, Multiple Context-Free Grammars, and Downward Closures. *Proc. ACM Program. Lang.* 10, POPL, Article 74 (Jan. 2026), 32 pages. https://doi.org/10.1145/3776716

[4] C. Aiswarya, Pascal Baumann, Prakash Saivasan, Lia Schütze, and Georg Zetzsche. 2026. Bounded Treewidth, Multiple Context-Free Grammars, and Downward Closures. *Proceedings of the ACM on Programming Languages* 10, POPL (2026), 2142–2173. https://doi.org/10.1145/3776716

[5] C. Aiswarya, Soumodev Mal, and Prakash Saivasan. 2022. On the Satisfiability of Context-free String Constraints with Subword-Ordering. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 6:1–6:13. https://doi.org/10.1145/3531130.3533329

[6] Ashwani Anand, Sylvain Schmitz, Lia Schütze, and Georg Zetzsche. 2024. Verifying Unboundedness via Amalgamation. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza (Eds.). ACM, 4:1–4:15. https://doi.org/10.1145/3661814.3662133

[7] Ashwani Anand and Georg Zetzsche. 2023. Priority Downward Closures. In *34th International Conference on Concurrency Theory, CONCUR 2023, Antwerp, Belgium, September 18-23, 2023 (LIPIcs, Vol. 279)*, Guillermo A. Pérez and Jean-François Raskin (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:18. https://doi.org/10.4230/LIPICS.CONCUR.2023.39

[8] Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. 2011. Context-Bounded Analysis For Concurrent Programs With Dynamic Creation of Threads. *Log. Methods Comput. Sci.* 7, 4 (2011). https://doi.org/10.2168/LMCS-7(4:4)2011

[9] Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. 2008. On the Reachability Analysis of Acyclic Networks of Pushdown Systems. In *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5201)*, Franck van Breugel and Marsha Chechik (Eds.). Springer, 356–371. https://doi.org/10.1007/978-3-540-85361-9_29

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

[11] Mohamed Faouzi Atig, Roland Meyer, Sebastian Muskalla, and Prakash Saivasan. 2017. On the Upward/Downward Closures of Petri Nets. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, Aalborg, Denmark, August 21-25, 2017 (LIPIcs, Vol. 83)*, Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 49:1–49:14. https://doi.org/10.4230/LIPICS.MFCS.2017.49

[12] Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. 2015. Finite Automata for the Sub- and Superword Closure of CFLs: Descriptional and Computational Complexity. In *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 8977)*, Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe (Eds.). Springer, 473–485. https://doi.org/10.1007/978-3-319-15579-1_37

[13] David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Pawel Parys. 2022. Cost Automata, Safe Schemes, and Downward Closures. *Fundam. Informaticae* 188, 3 (2022), 127–178. https://doi.org/10.3233/FI-222145

[14] David Barozzini, Pawel Parys, and Jan Wroblewski. 2022. Unboundedness for Recursion Schemes: A Simpler Type System. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, Paris, France, July 4-8, 2022 (LIPIcs, Vol. 229)*, Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 112:1–112:19. https://doi.org/10.4230/LIPICS.ICALP.2022.112

[15] Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. 2023. Checking Refinement of Asynchronous Programs Against Context-Free Specifications. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, Paderborn, Germany, July 10-14, 2023 (LIPIcs, Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 110:1–110:20. https://doi.org/10.4230/LIPICS.ICALP.2023.110

[16] Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. 2023. Context-Bounded Analysis of Concurrent Programs (Invited Talk). In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, Paderborn, Germany, July 10-14, 2023 (LIPIcs, Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:16. https://doi.org/10.4230/LIPICS.ICALP.2023.3

[17] Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. 2021. Context-bounded verification of liveness properties for multithreaded shared-memory programs. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–31. https://doi.org/10.1145/3434325

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

[19] Noam Chomsky. 1959. On Certain Formal Properties of Grammars. *Inf. Control.* 2, 2 (1959), 137–167. https://doi.org/10.1016/S0019-9958(59)90362-6

[20] Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. 2016. The Diagonal Problem for Higher-Order Recursion Schemes is Decidable. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 96–105. https://doi.org/10.1145/2933575.2934527

[21] Thomas Colcombet. 2007. A Combinatorial Theorem for Trees. In *Automata, Languages and Programming*, Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki (Eds.). Springer, 901–912. https://doi.org/10.1007/978-3-540-73420-8_77

[22] Thomas Colcombet. 2011. Green's Relations and Their Use in Automata Theory. In *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6638)*, Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide (Eds.). Springer, 1–21. https://doi.org/10.1007/978-3-642-21254-3_1

[23] Bruno Courcelle. 1991. On Constructing Obstruction Sets of Words. *Bull. EATCS* 44 (1991), 178–186.

[24] Bruno Courcelle and Géraud Sénizergues. 1994. The Obstructions of a Minor-Closed Set of Graphs Defined by Hyperedge Replacement can be Constructed. In *Graph Gramars and Their Application to Computer Science, 5th International Workshop, Williamsburg, VA, USA, November 13-18, 1994, Selected Papers (Lecture Notes in Computer Science, Vol. 1073)*, Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg (Eds.). Springer, 351–367. https://doi.org/10.1007/3-540-61228-9_98

[25] Joost Engelfriet. 1991. Iterated Stack Automata and Complexity Classes. *Inf. Comput.* 95, 1 (1991), 21–75. https://doi.org/10.1016/0890-5401(91)90015-T

[26] Moses Ganardi, Irmak Saglam, and Georg Zetzsche. 2024. Directed Regular and Context-Free Languages. In *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, Clermont-Ferrand, France, March 12-14, 2024 (LIPIcs, Vol. 289)*, Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 36:1–36:20. https://doi.org/10.4230/LIPICS.STACS.2024.36

[27] Hugo Gimbert, Corto Mascle, and Patrick Totzke. 2025. Optimal Sequential Flows. *arXiv preprint arXiv:2511.13806* (2025).

[28] Hermann Gruber, Markus Holzer, and Martin Kutrib. 2009. More on the Size of Higman-Haines Sets: Effective Constructions. *Fundam. Informaticae* 91, 1 (2009), 105–121. https://doi.org/10.3233/FI-2009-0035

[29] Peter Habermehl, Roland Meyer, and Harro Wimmel. 2010. The Downward-Closure of Petri Net Languages. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 6199)*, Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis (Eds.). Springer, 466–477. https://doi.org/10.1007/978-3-642-14162-1_39

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

[31] Leonard H Haines. 1969. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory* 6, 1 (1969), 94–98.

[32] Takeshi Hayashi. 1973. On derivation trees of indexed grammars—an extension of the uvwxy-theorem—. *Publications of the Research Institute for Mathematical Sciences* 9, 1 (1973), 61–92.

[33] Graham Higman. 1952. Ordering by Divisibility in Abstract Algebras. *Proceedings of the London Mathematical Society* s3-2, 1 (1952), 326–336. https://doi.org/10.1112/plms/s3-2.1.326

[34] Ismaël Jecker. 2021. A Ramsey Theorem for Finite Monoids. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference) (LIPIcs, Vol. 187)*, Markus Bläser and Benjamin Monmege (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 44:1–44:13. https://doi.org/10.4230/LIPICS.STACS.2021.44

[35] Ismaël Jecker. 2021. A Ramsey Theorem for Finite Monoids. *CoRR* abs/2101.05895 (2021). arXiv:2101.05895 https://arxiv.org/abs/2101.05895

[36] Prateek Karandikar, Matthias Niewerth, and Philippe Schnoebelen. 2016. On the state complexity of closures and interiors of regular languages with subwords and superwords. *Theor. Comput. Sci.* 610 (2016), 91–107. https://doi.org/10.1016/J.TCS.2015.09.028

[37] Alexander Kartzow. 2011. A Pumping Lemma for Collapsible Pushdown Graphs of Level 2. In *Computer Science Logic - 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, Bergen, Norway, September 12-15, 2011, Proceedings (LIPIcs, Vol. 12)*, Marc Bezem (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 322–336. https://doi.org/10.4230/LIPICS.CSL.2011.322

[38] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. 2002. Higher-Order Pushdown Trees Are Easy. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2303)*, Mogens Nielsen and Uffe Engberg (Eds.). Springer, 205–222. https://doi.org/10.1007/3-540-45931-6_15

[39] Naoki Kobayashi. 2009. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). ACM, 416–428. https://doi.org/10.1145/1480881.1480933

[40] Naoki Kobayashi. 2013. Model Checking Higher-Order Programs. *J. ACM* 60, 3 (2013), 20:1–20:62. https://doi.org/10.1145/2487241.2487246

[41] Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz. 2015. Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015 (LIPIcs, Vol. 42)*, Luca Aceto and David de Frutos-Escrig (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 72–84. https://doi.org/10.4230/LIPICS.CONCUR.2015.72

[42] Théodore Lopez, Benjamin Monmege, and Jean-Marc Talbot. 2025. Regular D-length: A tool for improved prefix-stable forward Ramsey factorisations. *Inf. Process. Lett.* 187 (2025), 106497. https://doi.org/10.1016/J.IPL.2024.106497

[43] Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. 2022. General Decidability Results for Asynchronous Shared-Memory Programs: Higher-Order and Beyond. *Log. Methods Comput. Sci.* 18, 4 (2022). https://doi.org/10.46298/LMCS-18(4:2)2022

[44] Richard Mayr. 2003. Undecidable problems in unreliable computations. *Theor. Comput. Sci.* 297, 1-3 (2003), 337–354. https://doi.org/10.1016/S0304-3975(02)00646-1

[45] Robert McNaughton. 1989. Varieties of Formal Languages (J. E. Pin; A. Howie, trans.). *SIAM Rev.* 31, 2 (1989), 347–348. https://doi.org/10.1137/1031081

[46] Luke Ong. 2015. Higher-Order Model Checking: An Overview. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 1–15. https://doi.org/10.1109/LICS.2015.9

[47] Pawel Parys. 2017. The Complexity of the Diagonal Problem for Recursion Schemes. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, Kanpur, India, December 11-15, 2017 (LIPIcs, Vol. 93)*, Satya V. Lokam and R. Ramanujam (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 45:1–45:14. https://doi.org/10.4230/LIPICS.FSTTCS.2017.45

[48] Pawel Parys. 2018. Intersection Types for Unboundedness Problems. In *Proceedings Twelfth Workshop on Developments in Computational Models and Ninth Workshop on Intersection Types and Related Systems, DCM/ITRS 2018, Oxford, UK, 8th July 2018 (EPTCS, Vol. 293)*, Michele Pagani and Sandra Alves (Eds.). 7–27. https://doi.org/10.4204/EPTCS.293.2

[49] Jacques Sakarovitch. 2009. *Elements of Automata Theory*. Cambridge University Press. http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521844253

[50] Sylvain Schmitz. 2017. *Algorithmic Complexity of Well-Quasi-Orders. (Complexité algorithmique des beaux pré-ordres)*. https://tel.archives-ouvertes.fr/tel-01663266

[51] Sylvain Schmitz and Philippe Schnoebelen. 2011. Multiply-Recursive Upper Bounds with Higman's Lemma. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 6756)*, Luca Aceto, Monika Henzinger, and Jirí Sgall (Eds.). Springer, 441–452. https://doi.org/10.1007/978-3-642-22012-8_35

[52] Imre Simon. 1990. Factorization Forests of Finite Height. *Theor. Comput. Sci.* 72, 1 (1990), 65–94. https://doi.org/10.1016/0304-3975(90)90047-L

[53] Tim Smith. 2017. A new pumping lemma for indexed languages, with an application to infinite words. *Inf. Comput.* 252 (2017), 176–186. https://doi.org/10.1016/J.IC.2016.11.002

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

[55] Georg Zetzsche. 2015. An Approach to Computing Downward Closures. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9135)*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.). Springer, 440–451. https://doi.org/10.1007/978-3-662-47666-6_35

[56] Georg Zetzsche. 2015. An approach to computing downward closures. *CoRR* abs/1503.01068 (2015). arXiv:1503.01068

[57] Georg Zetzsche. 2015. Computing Downward Closures for Stacked Counter Automata. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, Garching, Germany, March 4-7, 2015 (LIPIcs, Vol. 30)*, Ernst W. Mayr and Nicolas Ollinger (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 743–756. https://doi.org/10.4230/LIPICS.STACS.2015.743

[58] Georg Zetzsche. 2016. The Complexity of Downward Closure Comparisons. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, Rome, Italy, July 11-15, 2016 (LIPIcs, Vol. 55)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 123:1–123:14. https://doi.org/10.4230/LIPICS.ICALP.2016.123

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

function	defined where in [37]	growth w.r.t. $\lvert Q\rvert$
f_0	Lemma 14 on p. 329	constant
f_1	Theorem 22 on p. 331	exponential
f_2	Corollary 23 on p. 331	exponential
f_3	Theorem 25 on p. 332	triply exponential
f_4	Lemma 26 on p. 333	exponential
f_5	Corollary 27 on p. 333	exponential
f_6	Theorem 32 on p. 334	triply exponential

Table 1: Growth of functions in the paper [37]

A NORMALISING INDEXED GRAMMARS

When describing indexed grammars we sometimes use production rules of the form not allowed by our definition of indexed grammar

(i) $Af \to u$ with $u \in (N \cup T)^* \setminus N$
(ii) $A \to u$ with $u \in (N \cup T)^* \setminus (N^2 \cup T^*)$.

We now formally define how these rules should be eliminated to obtain an indexed grammar as in Definition 2.

We start by eliminating rules of the first type: we replace each rule $Af \to u$ with $u \in (N \cup T)^* \setminus N$ by two rules $Af \to A'$ and $A' \to u$, with A' a fresh non-terminal.

It remains to eliminate rules of the form $A \to u$ with $u \in (N \cup T)^* \setminus (N^2 \cup T^*)$. If $u = B \in N$ then replace the rule with $A \to BC$ and $C \to \varepsilon$ with C a fresh non-terminal. Otherwise, decompose u as $u = w_0A_1w_1 \dots A_kw_k$ with $A_1, \dots, A_k \in N$ and $w_0, \dots, w_k \in T^*$. Introduce fresh non-terminals $B_1, \cdots, B_k, C_1, \dots, C_{k-1}$. We replace $A \to u$ with rules

- $A \to W_0B_1$,
- $W_i \to w_i$ for all $i \in \{0, \dots, k\}$,
- $B_i \to A_iC_i$ for all $i \in \{1, \dots, k-1\}$,
- $C_i \to W_iB_{i+1}$ for all $i \in \{1, \dots, k-1\}$,
- $B_k \to A_kW_k$

Note that $\sum_{i=0}^{k} |w_i| \leq |u|$ and $k \leq |u|$. Hence, each such rule $A \to u$ is replaced by a set of at most $3|u| + 1$ rules, introducing at most $2|u| - 1$ non-terminals whose lengths sum up to at most $8|u|+6$. As a consequence, each rule of the form $Af \to u$ is replaced by a set of at most $3|u| + 2$ rules whose lengths sum up to at most $8|u| + 8$.

B ADDITIONAL MATERIAL FROM SECTION 3

The conclusion section (Section 7) of [37] claims that the pumping threshold $\mathfrak{P}$ (see Section 3 for the definition) grows at most doubly exponentially. Here, we briefly explain the mistake in this claim.

The corresponding results are Theorems 25, 32, and 33 in [37]. Each of them provides a bound ℓ (in terms of the number of states, the input alphabet, and a target configuration) such that in a (collapsible) order-2 pushdown automaton, if there is an accepting path of length $\geq \ell$, then there are infinitely many. These bounds are in the form of functions f_3 (for Thm. 25) and f_6 (for Thms. 32 and 33). However, both f_3 and f_6 grow at least triply exponentially in the number of states of the pushdown system. To see this, we track the functions $f_0, \dots, f_6$, which are defined across the paper, in Table 1.

C ADDITIONAL MATERIAL FROM SECTION 4

C.1 Proof of Lemma 4.3

In this subsection, we prove:

Lemma 4.3. *For every $f \in I$, $z \in I^*$, and $X \subseteq N$, we have $fz \cdot X = f \cdot (z \cdot X)$. Moreover, $z \cdot \mathbf{U} = z \cdot \emptyset$.*

We will prove this in the two separate lemmas below.

Lemma C.1. *For all $f \in I$, $z \in I^*$ and $X \subseteq N$, we have*

$$fz \cdot X = f \cdot (z \cdot X).$$

Proof. If $A \in f \cdot (z \cdot X)$ then (by definition) $A[f] \xRightarrow{*} u$, with $u \in (z \cdot X \cup T)^*$. Let $u[z]$ be the sentential form obtained by replacing every non-terminal B in u with $B[z]$ (i.e. pushing z onto every stack). Since all those non-terminals are in $z \cdot X$, there exists $v \in (X \cup T)^*$ such that $u[z] \xRightarrow{*} v$, implying that $A[fz] \xRightarrow{*} v$ and so $A \in fz \cdot X$.

To show the other inclusion, suppose that $A \in fz \cdot X$ and consider a derivation tree from $A[fz]$ to some $v \in (X \cup T)^*$. Along every branch there is a first node with a label either in T^* or of the form $B[z]$. In the latter case we have $B \in z \cdot X$ (since the tree from this node is a derivation tree from $B[z]$ to an element of $(X \cup T)^*$). After deleting everything below these nodes and removing the z suffix from the stack in each label, we obtain a derivation tree from $A[f]$ to an element of $(z \cdot X \cup T)^*$, completing the proof. □

Since non-terminals in $\mathbf{U}$ derive terminal words, we have:

Lemma C.2. *For all $z \in I^*$, $z \cdot \mathbf{U} = z \cdot \emptyset = \{A \in N \mid \mathsf{L}_{\emptyset}(A[z]) \neq \emptyset\}$.*

Proof. Note that the second equality is simply the definition of $z \cdot \emptyset$. We proceed by induction on z. For $z = \emptyset$, the statement is equivalent to the definition of $\mathbf{U}$. Assuming the statement holds for z, then two applications of Lemma C.1 yield

$$fz \cdot \mathbf{U} = f \cdot (z \cdot \mathbf{U}) = f \cdot (z \cdot \emptyset) = fz \cdot \emptyset,$$

completing the proof. □

C.2 Proof of Lemma 4.4

In this subsection, we prove:

Lemma 4.4. *$\mathcal{G}$ and $\overline{\mathcal{G}}$ have the same language, and $\overline{\mathcal{G}}$ is productive.*

We prove Lemma 4.4 below, after establishing a preliminary result, which details the relation between derivations in $\mathcal{G}$ and $\overline{\mathcal{G}}$. Define $\pi : \overline{N}\,\overline{I}^* \cup T \to NI^* \cup T$ to be the function projecting each $(A, X) \in \overline{N}$ to A, each $(f, X) \in \overline{I}$ to f and each $a \in T$ to itself. We naturally extend it to a morphism from $(\overline{N}\,\overline{I}^* \cup T)^*$ to $(NI^* \cup T)^*$

Lemma C.3. *Let $(A, Y)[\bar{z}] \in \overline{N}\,\overline{I}^*$ be the X-based annotation of $A[z] \in NI^*$ for some $X \subseteq N$. If $u \in \mathsf{L}_X(A[z])$, then there exists $\bar{u} \in (X \times \{X\} \cup T)^*$ such that*

$$\pi(\bar{u}) = u \text{ and } (A, Y)[\bar{z}] \xRightarrow{*}_{\overline{\mathcal{G}}} \bar{u}.$$

In particular, if $u \in T^*$ then $A[z] \xRightarrow{*}_{\mathcal{G}} u$ implies that $(A, Y)[\bar{z}] \xRightarrow{*}_{\overline{\mathcal{G}}} u$ as well.

Proof. We proceed by induction on the length of the derivation $A[z] \xRightarrow{*}_{\mathcal{G}} u$, and distinguish cases according to the production rule used in the first step.

- If the rule is of the form $A \to w \in T^*$, then $u = w$ and the statement is immediate.
- If the rule is of the form $A \to BC$, then $u = u_B u_C$ where $B[z] \overset{*}{\Rightarrow}_{\mathcal{G}} u_B$ and $C[z] \overset{*}{\Rightarrow}_{\mathcal{G}} u_C$. By our assumption, we have $Y = z \cdot X$, and since $A[z]$, $B[z]$ and $C[z]$ can all produce a word in $(X \cup T)^*$, we must have $A, B, C \in Y$ (also, $A \in Y$ by definition). Consequently, we may apply the corresponding derivation in $\overline{\mathcal{G}}$: $(A, Y)[\bar{z}] \Rightarrow_{\overline{\mathcal{G}}} (B, Y)[\bar{z}](C, Y)[\bar{z}]$. Note that $(B, Y)[\bar{z}]$ and $(C, Y)[\bar{z}]$ are X-based annotations of $B[z]$ and $C[z]$ respectively, so by the induction hypothesis there are $\bar{u}_B, \bar{u}_C \in (X \times \{X\} \cup T)^*$ such that $\pi(\bar{u}_B) = u_B$, $\pi(\bar{u}_C) = u_C$, $(B, Y)[\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}_B$ and $(C, Y)[\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}_C$. Setting $\bar{u} = \bar{u}_B \bar{u}_C$, we thus have $\pi(\bar{u}) = u$ and $(A, Y)[\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}$, as desired.
- If the rule is of the form $A \to Bf$, then $B[fz] \overset{*}{\Rightarrow}_{\mathcal{G}} u$. We have $Y = z \cdot X$, and $A \in Y$ by definition. Let $Y' = f \cdot Y = fz \cdot X$. Since $B[fz]$ can produce a word in $(X \cup T)^*$, it must be the case that $B \in Y'$. Hence, we have the corresponding derivation $(A, Y)[\bar{z}] \Rightarrow_{\overline{\mathcal{G}}} (B, Y')[(f, Y)\bar{z}]$. Note that $(B, Y')[(f, Y)\bar{z}]$ is an X-based annotation of $B[fz]$, so from the induction hypothesis we obtain $\bar{u} \in (X \times \{X\} \cup T)^*$ such that $\pi(\bar{u}) = u$ and $(B, Y')[(f, Y)\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}$. Hence, $(A, Y)[\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}$.
- If the rule is of the form $Af \to B$, then we have $z = fz'$ for some z' such that $B[z'] \overset{*}{\Rightarrow}_{\mathcal{G}} u$. Let $\bar{z}'$ be such that $\bar{z} = (f, X')\bar{z}'$. Since $(A, Y)[\bar{z}]$ is the X-based annotation of $A[z]$, we must have $X' = z' \cdot X$, and since $B[z']$ derives a word in $(X \cup T)^*$ it follows that $B \in X'$. Hence, there is a corresponding derivation $(A, Y)[(f, X')\bar{z}'] \Rightarrow_{\overline{\mathcal{G}}} (B, X')[\bar{z}']$. Since $(B, X')[\bar{z}']$ is easily seen to be the X-based annotation of $B[z']$, the induction hypothesis yields a derivation $(B, X')[\bar{z}'] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}$ for some $\bar{u} \in (X \times \{X\} \cup T)^*$ with $\pi(\bar{u}) = u$. Hence, $(A, Y)[\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} \bar{u}$ as desired.

This concludes the proof. □

Proof of Lemma 4.4. The inclusion $\mathsf{L}(\overline{\mathcal{G}}) \subseteq \mathsf{L}(\mathcal{G})$ is obtained as follows. For all $w \in \mathsf{L}(\overline{\mathcal{G}})$, we have a derivation tree for $\overline{\mathcal{G}}$ from $(S, \mathbf{U})$ to w. Since π maps the rules in $\overline{P}$ to rules in P, it is easy to check that the tree obtained by applying π to each node is a derivation tree from S to w for $\mathcal{G}$, implying that $w \in \mathsf{L}(\mathcal{G})$.

To obtain $\mathsf{L}(\mathcal{G}) \subseteq \mathsf{L}(\overline{\mathcal{G}})$, suppose that $w \in \mathsf{L}(\mathcal{G})$. Then there is a derivation $S \overset{*}{\Rightarrow}_{\mathcal{G}} w$, and since $(S, \mathbf{U})$ is a $\mathbf{U}$-based annotation of S it follows from Lemma C.3 that $(S, \mathbf{U}) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w$, implying $w \in \mathsf{L}(\overline{\mathcal{G}})$.

It remains to prove productiveness. We wish to show that every $\bar{u}$ such that $(S, \mathbf{U}) \Rightarrow_{\overline{\mathcal{G}}} \bar{u}$ is productive.

First observe that for all $\bar{u}$ such that $(S, \mathbf{U}) \Rightarrow_{\overline{\mathcal{G}}} \bar{u}$, every term $(A, X)[\bar{z}]$ of $\bar{u}$ is a $\mathbf{U}$-based annotation of some $A[z] \in NI^*$. This follows from the definition of $\overline{\mathcal{G}}$ and an easy induction on the derivation. Then, it is enough to prove that every such term $(A, X)[\bar{z}]$ is productive, since a sentential form can produce a terminal word if and only if all its terms can.

As $(A, X)[\bar{z}]$ is a $\mathbf{U}$-based annotation of some $A[z]$, we have $X = z \cdot \mathbf{U}$. Since $A \in X$ by definition of $\overline{N}$, there is a derivation from $A[z]$ to a word of T^*. As a result, by Lemma C.3 there is a derivation from $(A, X)[\bar{z}]$ to a word in T^*. □

D ADDITIONAL MATERIAL FROM SECTION 5

D.1 Proof of Lemma 5.1

Lemma 5.1. *Let $\bar{z} = (f_n, X_n) \cdots (f_1, X_1) \in \overline{I}^*$ be a stack content. The following are equivalent:*

(1) $\bar{z}$ is feasible
(2) $\varphi(\bar{z}) \neq \mathbf{0}_{\mathbb{M}}$
(3) for all $i > 1$, $X_i = f_i \cdot X_{i-1}$ and $\beta(f_{i-1})\ \mathcal{R}_{X_i}\ \alpha(f_i)$.

Proof. All three properties are clearly true for $\bar{z} = \varepsilon$. We now focus on non-empty stacks.

From the definitions of $\mathbb{M}$ and φ, one sees that property 3 is necessary and sufficient to ensure that no product of two consecutive infixes in $\varphi(\bar{z}) = \prod_{i=1}^{n} \varphi(f_i, X_i)$ is equal to $\mathbf{0}_{\mathbb{M}}$. It is also clear from the definitions that $\prod_{i=1}^{n} \varphi(f_i, X_i) = \mathbf{0}_{\mathbb{M}}$ if and only if two consecutive infixes multiply to equal $\mathbf{0}_{\mathbb{M}}$, so we immediately obtain $2 \Leftrightarrow 3$.

We now show that $1 \Rightarrow 3$. Let $\bar{z}$ be feasible, so by definition we have a derivation

$$(\alpha(f_1), X_1) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(\beta(f_n), f_n \cdot X_n)[\bar{z}]v. \qquad (*)$$

We proceed by induction on the derivation length. If $(*)$ has length one, then it must be of the form $(\alpha(f_1), X_1) \Rightarrow_{\overline{\mathcal{G}}} (\beta(f_1), f_1 \cdot X_1)[(f_1, X_1)]$, whence 3 holds trivially. We now assume that the length is greater than one, and that the first operation is of the form $(\alpha(f_1), X_1) \to (B, X_1)(C, X_1)$. In this case, we may assume without loss of generality that (B, X_1) derives $u'(\beta(f_n), f_n \cdot X_n)[\bar{z}]v'$ for some $u', v' \in \mathbf{SF}$. Since (f_1, X_1) must eventually be pushed (and $(\alpha(f_1), X_1)$ is the only nonterminal which allows this), it follows that $B\ \mathcal{R}_{X_1}\ \alpha(f_1)$, and that there is a derivation $(\alpha(f_1), X_1) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u''(\beta(f_n), f_n \cdot X_n)[\bar{z}]v''$ for some $u'', v'' \in \mathbf{SF}$ that is strictly shorter than $(*)$. Hence, 3 holds by induction.

Now let us assume that the length of $(*)$ is greater than one, and that the first operation is a push (the only remaining possibility). The push operation must be of the form

$$(\alpha(f_1), X_1) \Rightarrow_{\overline{\mathcal{G}}} (\beta(f_1), f_1 \cdot X_1)[(f_1, X_1)].$$

If $f_1 \cdot X_1 \neq X_2$, then it is easy to see that the right hand side cannot derive any term which pushes (f_2, X_2). Hence, we have

$$X_2 = f_1 \cdot X_1 \text{ and } \beta(f_1)\ \mathcal{R}_{X_2}\ \alpha(f_2), \qquad (**)$$

and there is a derivation $(\alpha(f_2), X_2)[(f_1, X_1)] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'(\beta(f_n), f_n \cdot X_n)[\bar{z}]v'$. Letting $\bar{z}' = (f_n, X_n) \cdots (f_2, X_2)$, this implies that $\bar{z}'$ is feasible with a derivation

$$(\alpha(f_2), X_2) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'(\beta(f_n), f_n \cdot X_n)[\bar{z}']v'$$

that is strictly shorter than $(*)$. Hence, $\bar{z}'$ satisfies 3 by induction, and with $(**)$ we immediately obtain 3 for $\bar{z}$.

Finally, we show that $3 \Rightarrow 1$. Let us assume that $\bar{z}$ satisfies 3. Recall that we assumed that for all $f \in I$ there is a rule pushing f. Then for $i = 1, \ldots, n-1$ we have

$$(\alpha(f_i), X_i) \Rightarrow_{\overline{\mathcal{G}}} (\beta(f_i), X_{i+1})[(f_i, X_i)]$$

and

$$(\beta(f_i), X_{i+1}) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u_i(\alpha(f_{i+1}), X_{i+1})v_i$$

for some $u_i, v_i \in \mathbf{SF}$, as well as

$$(\alpha(f_n), X_n) \Rightarrow_{\overline{\mathcal{G}}} (\beta(f_n), f_n \cdot X_n)[(f_n, X_n)].$$

It is clear that we may combine these derivations to obtain

$$(\alpha(f_1), X_1) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(\beta(f_n), f_n \cdot X_n)[\bar{z}]v,$$

proving that $\bar{z}$ is feasible. ◻

D.2 Proof of Lemma 5.2

Lemma 5.2. *Let $z \in I^+$ a non-empty stack content, let $X \subseteq N$ and let $(B, Y, M, A, X) = \varphi(\overline{z}^X)$. Then for all $C \in X$ and $D \in Y$, the following are equivalent:*

- *$C \mathcal{R}_X A$ and $B \mathcal{R}_Y D$*
- *there exist $u, v \in \mathbf{SF}$ such that $(C, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(D, Y)[\overline{z}^X]v$.*

Proof. From the first condition, we have derivations

$$(C, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u_C(A, X)v_C \text{ and } (B, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u_D(D, X)v_D$$

for some $u_C, v_C, u_D, u_D \in \mathbf{SF}$. Let $z = f_n \cdots f_1$. From the definition of φ (and the fact that $\varphi(\overline{z}^X) \neq \mathbf{0}_{\mathbb{M}}$) it follows easily that $A = \alpha(f_1)$, $B = \beta(f_n)$ and $Y = z \cdot X$. Hence, Lemma 5.1 ensures that there is a derivation

$$(A, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'(B, Y)[\overline{z}^X]v'$$

for some $u', v' \in \mathbf{SF}$, which we combine with the derivations above to obtain

$$(C, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(D, Y)[\overline{z}^X]v,$$

proving one direction.

For the other implication, suppose such a derivation exists. Eventually, (f_1, X) must be pushed onto an empty stack, and since (A, X) is the only non-terminal which facilitates this operation, it follows that $C \mathcal{R}_X A$. Similarly, the derivation must eventually push the topmost symbol in $\overline{z}^X$, and the only non-terminal which can result from this operation is (B, Y). This implies that $(B, Y)[\overline{z}^X] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(D, Y)[\overline{z}^X]v$, hence $B \mathcal{R}_Y D$, completing the proof. ◻

D.3 Proof of Lemma 5.3

Lemma 5.3. *Let $z \in I^+$ a non-empty stack content, let $X \subseteq N$ and let $(B, Y, M, A, X) = \varphi(\overline{z}^X)$. The following are equivalent:*

- *$M(D, C) = \top$*
- *$C \in X$, $D \in Y$ and there exist $u, v \in (X \cup T)^*$ such that*
$$D[z] \overset{*}{\Rightarrow}_{\mathcal{G}} uCv$$
- *$C \in X$, $D \in Y$ and there exist $u, v \in \mathbf{SF}_{\mathcal{G}}$ such that*
$$(D, Y)[\overline{z}^X] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(C, X)v.$$

Proof. Equivalence of the second and third statements follows easily from Lemma C.3. Hence, it suffices to prove equivalence of the first and second statements. We proceed by induction on z. If $|z| = 1$ then we have $\overline{z}^X = (f, X)$ with $\varphi(f, X) = (B, Y, M, A, X)$, and the equivalence holds simply by definition.

If $|z| > 1$, then let w be such that $z = fw$. Let $Z = w \cdot X$, so that $\overline{z}^X = (f, Z)\overline{w}^X$, and let us write $(B_f, Y, M_f, A_f, Z) = \varphi(f, Z)$ and $(B_w, Z, M_w, A_w, X) = \varphi(\overline{w}^X)$. Note that $M = M_f M_w$. We prove the two directions separately.

$\Rightarrow$: Suppose $M(D, C) = \top$. Then there exists E such that $M_f(D, E) = M_w(E, C) = \top$. From the induction hypothesis applied to f, it follows that $D \in Y$, $E \in Z$ and there exist $u_1, v_1 \in (Z \cup T)^*$ such that $D[f] \overset{*}{\Rightarrow}_{\mathcal{G}} u_1 E v_1$. On the other hand, applying the induction hypothesis to w yields $C \in X$ and $u_2, v_2 \in (X \cup T)^*$ such that $E[w] \overset{*}{\Rightarrow}_{\mathcal{G}} u_2 C v_2$.

Since $Z = w \cdot X$ and $u_1, v_1 \in (Z \cup T)$, there must be $u_3, v_3 \in (X \cup T)^*$ such that $u_1[w] \overset{*}{\Rightarrow}_{\mathcal{G}} u_3$ and $v_1[w] \overset{*}{\Rightarrow}_{\mathcal{G}} v_3$. Combining the facts above, we get

$$D[z] \overset{*}{\Rightarrow}_{\mathcal{G}} u_1[w]E[w]v_1[w] \overset{*}{\Rightarrow}_{\mathcal{G}} u_3 E[w] v_3 \overset{*}{\Rightarrow}_{\mathcal{G}} u_3 u_2 C v_2 v_3$$

as desired.

$\Leftarrow$: Suppose $D \in Y$, $C \in X$ and there exist $u, v \in (X \cup T)^*$ such that $D[z] \overset{*}{\Rightarrow}_{\mathcal{G}} uCv$. Then we must have $D[f] \overset{*}{\Rightarrow}_{\mathcal{G}} u_1 U v_1$, $U[w] \overset{*}{\Rightarrow}_{\mathcal{G}} u_2 C v_2$, $u_1[w] \overset{*}{\Rightarrow}_{\mathcal{G}} u_3$ and $v_1[w] \overset{*}{\Rightarrow}_{\mathcal{G}} v_3$ for some $U \in N$ and sentential forms $u_1, u_2, u_3, v_1, v_2, v_3$ such that $u = u_3 u_2$ and $v = v_2 v_3$. As a consequence, we have $u_2, u_3, v_2, v_3 \in (X \cup T)^*$. Since $w \cdot X = Z$, we get that $u_1, v_1 \in (Z \cup T)^*$, and since $C \in X$, the derivation $U[w] \overset{*}{\Rightarrow}_{\mathcal{G}} u_2 C v_2 \in (X \cup T)^*$ proves that $U \in w \cdot X = Z$. By the induction hypothesis, we have $M_w(D, U) = M_f(U, C) = \top$, implying that $M(D, C) = \top$. ◻

E ADDITIONAL MATERIAL FROM SECTION 6

E.1 Proof of Proposition 6.1

We prove the following statement:

Proposition 6.1. $\mathsf{L}_{\mathsf{pump,skip}}(\overline{\mathcal{G}}) \subseteq \mathsf{L}(\overline{\mathcal{G}})\!\downarrow$

The following two auxiliary lemmas are required. The first one shows that we can eliminate pump rules, the second one that we can eliminate skip rules.

Call a sentential form u of $\overline{\mathcal{G}}$ *reachable* if $u \in \mathsf{L}_{\mathbf{SF}}((S, \mathbf{U}))$. Call a term $(B, X)[\bar{z}]$ *reachable* if it appears in a derivation from $(S, \mathbf{U})$.

Lemma E.1. *Let $e = (B, X, M, A, X) \in \mathbf{Idem}(\mathbb{M}) \setminus \{\mathbf{0}_{\mathbb{M}}, \mathbf{1}_{\mathbb{M}}\}$, let $(B, X)[\bar{z}]$ be a reachable term of $\overline{\mathcal{G}}$ and let $z_e \in \overline{I}^*$ be such that $\varphi(z_e) = e$. Then*

$$\mathsf{L}_{\emptyset}((B, X)[z_e \bar{z}]) \subseteq \mathsf{L}_{\emptyset}((B, X)[\bar{z}])\!\downarrow.$$

Proof. Let $w \in \mathsf{L}_{\emptyset}((B, X)[z_e \bar{z}])$, so there is a derivation

$$(B, X)[z_e \bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w.$$

Since $\varphi(z_e) = e \neq \mathbf{0}_{\mathbb{M}}$, by Lemma 5.1, z_e is feasible, so there is a derivation $(A, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u(B, X)[z_e]v$ with $u, v \in \mathbf{SF}$. Furthermore, since e is idempotent, by definition of the product in $\mathbb{M}$ we have $B \mathcal{R}_X A$, i.e., there is a derivation $(B, X) \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'(A, X)v'$. In total, we obtain

$$(B, X)[\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'(A, X)[\bar{z}]v' \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'u(B, X)[z_e \bar{z}]vv' \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} u'uwvv'.$$

Moreover, since $\overline{\mathcal{G}}$ is productive and $(B, X)[\bar{z}]$ is a reachable term of $\overline{\mathcal{G}}$, it follows that $u'uwvv'$ is reachable as well, and can thus derive a terminal word w'. Since w' necessarily contains w as a subword, the proof is complete. ◻

Lemma E.2. *Let $(A, Y)[z'u_1 \ldots u_N z_e u_1 \ldots u_N \bar{z}]$ be a reachable term of $\overline{\mathcal{G}}$, and let $e = (B, X, M, A, X) \in \mathbf{Idem}(\mathbb{M})$ be such that $\varphi(u_1) = \cdots = \varphi(u_N) = \varphi(z_e) = e$. Then*

$$\mathsf{L}_{\emptyset}((A, Y)[z'u_1 \ldots u_N \bar{z}]) \subseteq \mathsf{L}_{\emptyset}((A, Y)[z'u_1 \ldots u_N z_e u_1 \ldots u_N \bar{z}])\!\downarrow.$$

Proof. Let $w \in \mathsf{L}_{\emptyset}((A, Y)[z'u_1 \ldots u_N \bar{z}])$, and let τ be a derivation tree for $(A, Y)[z'u_1 \ldots u_N \bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w$.

Consider the set of nodes v such that either

(1) v is a leaf with a label in T^* and $\bar{z}$ is a suffix of the stack content of each of its ancestors, or
(2) v is labeled $(C,X)[u_{i+1}\dots u_N\bar{z}]$ for some $i \in \{0,\dots,|N|\}$ (where u_{N+1} is the empty string) and $u_{i+1}\dots u_N\bar{z}$ is a suffix of the stack content of each of its ancestors.

A node of the second type which also satisfies $M(C,C) = \top$ is called a *special node*.

Claim. *Every branch of τ contains either a node of the first type or a special node.*

Proof of the claim: Consider a branch of τ. If the leaf of this branch is not of the first type, then the prefix $z'u_1\dots u_N$ must be fully popped. Clearly there must be distinct nodes $v_0,\dots,v_N$ of the second type along this branch, where v_i is labeled $(A_i,X_i)[u_{i+1}\dots u_N\bar{z}]$ for some $A_i \in N$, and each of its ancestors has a stack content with suffix $u_{i+1}\dots u_N\bar{z}$. Consequently, for $i = 0,\dots,|N|-1$ we can define the derivation tree obtained by restricting τ to v_i and its descendants, and then removing all nodes (and their descendants) where $u_{i+1}\dots u_N\bar{z}$ is not a suffix of the stack, as well as all descendants of v_{i+1}. By construction, v_{i+1} is a leaf of the resulting tree, and so we obtain a derivation

$$(A_i,X_i)[u_i\dots u_N\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_i(A_{i+1},X_{i+1})[u_{i+1}\dots u_N\bar{z}]w_i'$$

with $w_i, w_i' \in \mathbf{SF}$. Since $\varphi(u_i) = e$, we must have $X_i = X_{i+1} = X$ and $M(A_i,A_{i+1}) = \top$. Thus, it follows from Lemma 6.2 that there is some i such that $M(A_i,A_i) = \top$, so v_i is a special node. ■

Let v be a special node, labeled $(C,X)[u_{i+1}\dots u_N\bar{z}]$ with $M(C,C) = \top$. Since $\varphi(u_1) = \cdots = \varphi(u_N) = \varphi(z_e) = e$, it follows that $\varphi(u_{i+1}\dots u_N z_e u_1\dots u_i) = e$. As a consequence, since $M(C,C) = \top$, we have a derivation $(C,X)[u_{i+1}\dots u_N z_e u_1\dots u_i] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_-(C,X)w_+$ with $w_-, w_+ \in \mathbf{SF}$ (see Lemma 5.3).

For the following construction we refer to Figure 1 for a visual presentation.

Consider the set V of minimal nodes (for the ancestor relation) which are either of the first type or special. Notice that V intersects every branch exactly once: at most once by the minimality requirement, at least once by the claim above. We can thus define the subtree whose root is the same as τ and whose leaves are V. By construction, every node in this tree is labeled with either a terminal word or a term whose stack has $\bar{z}$ as a suffix. Define τ' the tree obtained by replacing each suffix $\bar{z}$ with $z_e u_1\dots u_N\bar{z}$, and notice that the resulting tree τ' is still a valid derivation tree.

For each leaf v of τ' that was a special node of τ, with a label $(C,X)[u_{i+1}\dots u_N z_e u_1\dots u_N\bar{z}]$ in τ' for some (C,X) and i, we append a derivation tree τ_v for

$$(C,X)[u_{i+1}\dots u_N z_e u_1\dots u_N\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_-(C,X)[u_{i+1}\dots u_N\bar{z}]w_+,$$

where $w_-, w_+ \in \mathbf{SF}$. Now take the subtree of τ rooted at v, and append it at the leaf of τ_v labeled with $(C,X)[u_{i+1}\dots u_N\bar{z}]$.

The result is a derivation tree from $(A,Y)[z'u_1\dots u_N z_e u_1\dots u_N\bar{z}]$ to a sentential form $\overline{w}$ of which w is a subword. Since $\overline{\mathcal{G}}$ is productive and $(A,Y)[z'u_1\dots u_N z_e u_1\dots u_N\bar{z}]$ is a reachable term, there is a word $w' \in T^*$ such that $\overline{w} \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w'$, and since $w \preceq \overline{w}$ we have $w \preceq w'$, completing the proof. □

Proof of Proposition 6.1. We show the stronger statement that for all reachable sentential form u in $\overline{\mathcal{G}}$, for all terminal word $w \in T^*$ such that $u \overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}} w$, there exists $w' \in T^*$ such that $u \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w'$ and $w \preceq w'$. The result then follows by taking $u = (S,\mathbf{U})$.

We proceed by induction on the derivation $u \overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}} w$, distinguishing cases according to the first step. Note that the base case, where $u = w$, is trivial.

- If the first step is a production rule in $\overline{\mathcal{G}}$, say $u \Rightarrow_{\overline{\mathcal{G}}} u'$, then $u' \overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}} w$ with a shorter derivation. By the induction hypothesis, $u' \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w' \in T^*$ with $w \preceq w'$, hence $u \Rightarrow_{\overline{\mathcal{G}}} u' \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w'$.
- If the first step is a pump rule, say

$$u = u_-(B,X)[\bar{z}]u_+ \rightarrow_{\text{pump}} u_-(B,X)[z_e\bar{z}]u_+ = u',$$

with $\varphi(z_e) = e$ for some idempotent $e = (B,X,M,A,X)$. then $u' \overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}} w$, and the induction hypothesis implies that $u' \Rightarrow_{\overline{\mathcal{G}}} w' \in T^*$ with $w \preceq w'$. Let us write $w' = w_- w_B w_+$, where
  - $u_- \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_-$,
  - $u_+ \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_+$, and
  - $(B,X)[z_e\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_B$.

  By Lemma E.1 we have that $u \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_- w_B' w_+$ with $w_B \preceq w_B'$. The desired result follows, since $w \preceq w' \preceq w_- w_B' w_+$.
- If the first step is a skip rule, say

$$u = u_-(A,X)[z'u_1\dots u_N z_e u_1\dots u_N\bar{z}]u_+ \\ \rightarrow_{\text{skip}} u_-(A,X)[z'u_1\dots u_N\bar{z}]u_+ = u',$$

then $u' \overset{*}{\Rightarrow}_{\text{pump, skip}, \overline{\mathcal{G}}} w$, and by the induction hypothesis $u' \Rightarrow_{\overline{\mathcal{G}}} w' \in T^*$ with $w \preceq w'$. Let us write $w' = w_- w_A w_+$ where
  - $u_- \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_-$,
  - $u_+ \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_+$, and
  - $(A,X)[z'u_1\dots u_N\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_A$.

  Then by Lemma E.2 we obtain

$$(A,X)[z'u_1\dots u_N z_e u_1\dots u_N\bar{z}] \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_A'$$

  with $w_A \preceq w_A'$. We thus have $u \overset{*}{\Rightarrow}_{\overline{\mathcal{G}}} w_- w_A' w_+$, and since $w \preceq w' \preceq w_- w_A' w_+$, the result follows and the proof is complete.

□

F ADDITIONAL MATERIAL FROM SECTION 7

F.1 Proof of Lemma 7.3

In this section we prove the following statement.

Lemma 7.3. *For every $\bar{z} \in \bar{I}^*$, the summary $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ is of size at most exponential in $|N|$.*

Its proof is very similar to the one of Theorem 26 in [27]. We start by recalling some classical facts on Green relations. For a more in-depth introduction to those, see for instance [45], or [22].

Lemma F.1. *In a finite monoid $\mathbf{M}$, every $\mathcal{H}$-class contains at most one idempotent.*

Lemma F.2. *In a finite monoid $\mathbf{M}$, for all $x,y \in \mathbf{M}$, if $x\mathcal{J}y$ and $x \leq_{\mathcal{L}} y$ (resp. $x \leq_{\mathcal{R}} y$) then $x\mathcal{L}y$ (resp. $x\mathcal{R}y$).*

Define the *Ramsey function* of $\mathbf{M}$ as follows: for all $k \in \mathbb{N}$, $\mathcal{R}_{\mathbf{M}}(k)$ is the minimal n such that for every word of length n there exists $e \in \mathbf{Idem}(\mathbf{M})$ and $u_1, \dots, u_k \in \mathbf{M}^+$ such that the word contains an infix $u_1 \cdots u_k$ with $\psi_{\mathbf{M}}(u_i) = e$ for all i.

Theorem F.3 ([34], Theorem 1). *For all finite monoid $\mathbf{M}$, for all $k \in \mathbb{N}$,*

$$\mathcal{R}_{\mathbf{M}}(k) \leq (k|\mathbf{M}|^4)^{\mathcal{JL}(\mathbf{M})}.$$

Theorem F.4 ([34], Theorem 2). *The regular $\mathcal{J}$-length of $\mathbb{B}^{N\times N}$ is $\mathcal{JL}(\mathbb{B}^{N\times N}) = \frac{N^2+N+2}{2}$.*

Theorem F.5. $\mathcal{JL}(\mathbb{M}) \leq \frac{(N^2+N+2)}{2} + 2$.

Proof. Let us start by using another definition of the regular $\mathcal{J}$-length. By [35, Appendix B], the regular $\mathcal{J}$-length of $\mathbb{M}$ is the largest m such that there is an injective homomorphism from the max monoid $(\{1, \dots, m\}, \max, m)$ to $\mathbb{M}$.

Let $m \in \mathbb{N}$, let $\theta : \{1, \dots, m\} \to \mathbb{M}$ be such a homomorphism. We must show that $m \leq \frac{(N^2+N+2)}{2} + 2$.

If $\mathbf{0}_{\mathbb{M}}$ is in the image of θ then $\theta(m) = \mathbf{0}_{\mathbb{M}}$. If $\mathbf{1}_{\mathbb{M}}$ is in the image of θ then $\theta(1) = \mathbf{1}_{\mathbb{M}}$. Since all i in $\{2, \dots, m-1\}$ are idempotents, the image of each i by θ must also be an idempotent. As a result, we can set $\theta(i) = (B_i, X_i, M_i, A_i, X_i)$ for all $1 < i < m$, with $M_i^2 = M_i$.

Furthermore, for all $1 < i < j < m$, since $\max(i, j) = j$, we must have

$$\begin{aligned} &(B_j, Y_j, M_j, A_j, X_j) \cdot (B_i, X_i, M_i, A_i, X_i) \\ =&(B_i, X_i, M_i, A_i, X_i) \cdot (B_j, Y_j, M_j, A_j, X_j) \\ =&(B_j, X_j, M_j, A_j, X_j) \end{aligned}$$

We infer that $X_j = X_i$, $B_j = B_i$ and $A_j = A_i$ for all $i < j$. Since θ is injective, $M_2, \dots, M_{m-1}$ must be distinct.

Define the function $\theta' : \{1, \dots, m-2\}$ mapping each i to M'_{i+1}. It suffices to observe that θ_p is an injective homomorphism from the max monoid of size $m-2$ to $\mathbb{B}^{N\times N}$. As a consequence, we have $m - 2 \leq \mathcal{JL}(\mathbb{B}^{N\times N})$. By Theorem F.4, we have $\mathcal{JL}(\mathbb{B}^{N\times N}) \leq \frac{N^2+N+2}{2}$. As a result, $m \leq \frac{(N^2+N+2)}{2} + 2$. ◻

Observe that the size of $\mathbb{M}$ is bounded by $N^2 2^{N^2+2N}$. Define $K = ((2N+1)N^8 2^{4N^2+8N})^{\frac{(N^2+N+2)}{2}+2}$. By combining the results above, we obtain the following corollary.

Corollary F.6. *Let $z \in \mathbb{M}^*$ with $|z| \geq K$, there exist $u_1, \dots, u_N \in \mathbb{M}^*$ and $v_0, \dots, v_N \in \mathbb{M}^*$ and $e \in \mathbf{Idem}(\mathbb{M})$ such that*

- *$u_1 \dots u_N v_0 \dots v_N$ is an infix of z*
- *$\varphi(u_1) = \cdots = \varphi(u_N) = \varphi(v_0) = \cdots = \varphi(v_N) = e$*

In what follows we distinguish the size of a d-summary/d-block from its *length*, which is simply its length as a word of d-atoms, e^+ letters and d'-summaries for various $d' < d$.

Lemma F.7. *For all $\bar{z} \in \bar{I}^*$, $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon) = \sigma' u B_1 \dots B_k$ is such that u has length at most K and all B_j at most $2K$.*

Proof. We first show it for u. If $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon) = \sigma' u B_1 \dots B_k$ then u cannot have an infix of the form $u_1 \dots u_N v_0 \dots v_N$ with $\varphi(u_i) = \varphi(v_i) = \varphi(v_0) = e$ for all i, for any $e \in \mathbf{Idem}(\mathbb{M})$. This is because when such a pattern appears u is turned into a d-block. As a consequence, by Corollary F.6, u has length at most $K - 1$.

For the blocks, we show that in a block $u_1 \cdots u_N e^+ v_1 \cdots v_N w$ appearing in $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$, the lengths of $u_1 \dots u_N e^+$ and $v_1 \dots v_N w$ are always at most K. We show this by induction on $|\bar{z}|$. For $\bar{z} = \varepsilon$ this is trivial. Now suppose $|\bar{z}| > 0$, let $(f, X)\bar{z}' = \bar{z}$. By induction hypothesis $\mathsf{push}(\bar{z}' \blacktriangleright \varepsilon)$ has the property.

Let $\sigma'' u' B'_1 \dots B'_m = \mathsf{push}(\bar{z}' \blacktriangleright \varepsilon)$ We show the property on $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ (which is equal to $\mathsf{push}((f, X) \blacktriangleright \mathsf{push}(\bar{z}' \blacktriangleright \varepsilon))$ by definition) by following the cases of the definition of $\mathsf{push}(\_ \blacktriangleright \_)$.

In cases (1), (a) and (ii) blocks remain the same, thus the property still holds. In case (i), we define a new block $B = u_1 \cdots u_N e^+ v_1 \cdots v_N w$ by concatenating (f, X) with u'. Since we showed that u' has length at most $K - 1$, B has length at most K, hence so do $u_1 \cdots u_N e^+$ and $v_1 \cdots v_N w$. In case (B), we obtain the property immediately. In case (A), we form a new block $u_1 \cdots u_N e^+ v'_1 \cdots v'_N w'$ by merging B with one of the $B'_j = u'_1 \cdots u'_N e^+ v'_1 \cdots v'_N w'$. Since $u_1 \cdots u_N e^+$ and $v'_1 \cdots v'_N w'$ both have length at most K, the property is maintained. ◻

Lemma F.8. *For all $\bar{z} \in \bar{I}^*$, $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ has at most $|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}$ blocks.*

Proof. Let $\sigma = \sigma' u B_1 \dots B_m$ be a d-summary. For each i let $B_i = u_{i,1} \dots u_{i,N} e_i^+ v_{i,1}, \dots, v_{i,N} w_i$ and for each $i < j$ define $\alpha_{i,j} = \varphi(v_{i,1} \dots v_{i,N} w_i B_{i+1} \cdots B_{j-1} u_{j,1} \dots u_{j,N})$.

Note that since $\varphi(v_{i,1}) = e_i$ and $e_i \in \mathbf{Idem}(\mathbb{M})$ for all i we have $\alpha_{i,j} = e_i \cdot \varphi(v_{i,1} \dots v_{i,N} w_i B_{i+1} \cdots B_{j-1} u_{j,1} \dots u_{j,N})$ and thus

$$\alpha_{i,j} \cdot \alpha_{j,k} = \alpha_{i,k} \text{ for all } i < j < k$$

Suppose by contradiction that $m > |\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}$, then by pigeonhole principle there exist $e \in \mathbf{Idem}(\mathbb{M})$ and $i_1 < \cdots < i_p \in \{1, \dots, K\}$ such that $p > (|\mathbb{M}|)^{4\mathcal{JL}(\mathbb{M})}$ and $e_{i_k} = e$ for all k.

Then by Theorem F.3, there exist k, ℓ such that $\varphi(\alpha_{i_k,i_\ell})$ is an idempotent e'.

Since σ is a d-summary, we have $\mathbf{depth}(e') = d = \mathbf{depth}(e)$. In consequence, since $e' \leq_{\mathcal{J}} e$ and they are both idempotent, we must have $e \mathcal{J} e_i$. Furthermore, since $e' \leq_{\mathcal{R}} \alpha_{i_k} \leq_{\mathcal{R}} \varphi_{\mathbb{M}}(\pi(v_{i_k,1})) = e$, we have $e' \leq_{\mathcal{R}} e$ and thus $e \mathcal{R} e'$ by Lemma F.1. Similarly, since $e' \leq_{\mathcal{L}} \alpha_{i_\ell} \leq_{\mathcal{L}} e$ we have $e' \leq_{\mathcal{L}} e$ and thus $e \mathcal{L} e'$.

We obtain $e \mathcal{H} e'$. By Lemma F.2, this implies $e = e'$. This is a contradiction since then the blocks from B_{i_k} to B_{i_ℓ} should have been merged when B_{i_k} was created. As a result, we must have $m \leq |\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}$. ◻

We now have all necessary tools to show Lemma 7.3.

Proof of Lemma 7.3. We show that for all $\bar{z}$ and d, if $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$ is a d-summary then it has size at most $(8|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)^d$.

We do an induction on d. The property trivially holds for $d = 0$. Let $d > 0$, suppose the property holds for $d - 1$.

Let $\mathsf{push}(\bar{z} \blacktriangleright \varepsilon) = \sigma' u B_1 \dots B_k$. By Lemma F.8 we must have $k \leq |\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}$. By Lemma F.7 we have $|u| \leq K$ and $|B_i| \leq 2K$ for all i.

Therefore $\mathsf{push}(z \blacktriangleright \varepsilon)$ has length at most $(|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}+1)K+1$, which is bounded by $4|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K$.

Let $a\sigma''$ be a d-atom appearing in $\bar{z}$, with σ'' a $(d-1)$-summary. Note that there must be a (strict) infix $\bar{z}''$ of $\bar{z}$ such that $\mathsf{push}(\bar{z}'' \blacktriangleright \varepsilon) = \sigma''$. By induction hypothesis σ'' has size at most $(8|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)^{d-1}$.

Thus every d-atom has size at most $(8|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)^{d-1} + 1 \leq 2(8|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)^{d-1}$. With the bound on its length, we conclude that $\mathsf{push}(z \blacktriangleright \varepsilon)$ has size at most

$$\begin{aligned}&2(8|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)^{d-1}(4|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)\\ \leq&(8(|\mathbb{M}|^{4\mathcal{JL}(\mathbb{M})+1}K)^{d}\end{aligned}$$

We obtain the result by applying this bound with $d = \mathcal{JL}(\mathbb{M})$, which is at most $\frac{(N^2+N+2)}{2} + 2$ by Theorem F.5. ◻

G ADDITIONAL MATERIAL FROM SECTION 8

G.1 Proof of Proposition 8.2

Proposition 8.2. $\mathsf{L}(\overline{\mathcal{G}}) \subseteq \mathsf{L}(C_{\mathcal{G}})$.

Proof. Let $w \in \mathsf{L}(\overline{\mathcal{G}})$. There is a derivation tree from $(S, \mathbf{U})$ to w. Let τ its tree structure and $\lambda : \tau \to \overline{NI}^* \cup T^*$ its labeling.

We now define $\mu : \tau \to \mathbf{FT} \cup T^*$ a labeling of τ with non-terminals and (words of) terminals of $C_{\mathcal{G}}$ such that for all $v \in \tau$,

- if $\lambda(v) \in T^*$ then $\mu(v) = \lambda(v)$
- if $\lambda(v) = (A, X)[\bar{z}] \in \overline{NI}^*$ then $\mu(v) = (A, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon))$

We show that the resulting labeled tree is a derivation tree from $(S, \mathbf{U}, \varepsilon)$ to w in $C_{\mathcal{G}}$, thereby showing the lemma.

Clearly the leaf word of τ, μ is w. It remains to show that this is a derivation tree. Since τ, λ is a derivation tree from $(S, \mathbf{U})$ to w, all stack contents appearing in it must be feasible. As a consequence, μ maps all nodes of τ to non-terminals of $C_{\mathcal{G}}$. Let v an internal node of τ, and $(A, X)[\bar{z}] = \lambda(v)$. One of the following cases holds.

- v has one child labeled $w' \in T^*$, and there is a rule $(A, X) \to w'$ in $\overline{\mathcal{G}}$. Then $(A, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)) \to w'$ is a rule of $C_{\mathcal{G}}$.
- v has two children labeled $B[\bar{z}]$ and $C[\bar{z}]$, and there is a rule $(A, X) \to (B, X)(C, X)$ in $\overline{\mathcal{G}}$. Then $(A, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)) \to (B, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon))(C, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon))$ is a rule of $C_{\mathcal{G}}$.
- v has a child labeled $B[(f, X)\bar{z}]$, and there is a rule $(A, X) \to (B, Y)(f, X)$ in $\overline{\mathcal{G}}$. Then, since $\mathsf{push}((f, X) \blacktriangleright \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)) = \mathsf{push}((f, X)\bar{z} \blacktriangleright \varepsilon)$, it must be that $(A, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)) \to (B, Y, \mathsf{push}((f, X)\bar{z} \blacktriangleright \varepsilon))$ is a rule of $C_{\mathcal{G}}$.
- v has a child labeled $B[\bar{z}_-]$ with $\bar{z} = (f, Y)\bar{z}_-$, and there is a rule $(A, X)(f, Y) \to (B, Y)$ in $\overline{\mathcal{G}}$. Then, since by definition $\mathsf{push}((f, X) \blacktriangleright \bar{z}_-) = \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)$, we have $\bar{z}_- \in \mathsf{pop}((f, Y) \blacktriangleleft \bar{z})$.
  As a result, $(A, X, \mathsf{push}(\bar{z} \blacktriangleright \varepsilon)) \to (B, Y, \mathsf{push}(\bar{z}_- \blacktriangleright \varepsilon))$ is a rule of $C_{\mathcal{G}}$.

We have shown that every node satisfies the requirements of a derivation tree for $C_{\mathcal{G}}$. ◻

G.2 Proof of Proposition 8.3

Proposition 8.3. $\mathsf{L}(C_{\mathcal{G}}) \subseteq \mathsf{L}_{\mathsf{pump,skip}}(\overline{\mathcal{G}})$.

Proof. A *pop step* is simply a derivation step of $C_{\mathcal{G}}$ where the production rule applied is of the form $(A, X, \sigma) \to (B, Y, \sigma')$ with $\sigma = \mathsf{push}((f, Y) \blacktriangleright \sigma')$. We prove the following statement:

For all derivation with p pop steps from $(A, X, \sigma) \overset{*}{\Rightarrow}_{C_{\mathcal{G}}} w$ with $w \in T^*$, there is a derivation *with pump and skip* from $(A, X)[\bar{z}]$ to w' in $\overline{\mathcal{G}}$ with $w \preceq w'$, and $\bar{z}$ the p-unfolding of σ.

We show this by induction on the derivation, and distinguish cases according to the rule used in its first step.

- If the rule is of the form $(A, X, \sigma) \to w$ then we apply the corresponding rule $(A, X) \to w$ from $(A, X)[\bar{z}]$, hence $(A, X)[\bar{z}] \Rightarrow_{\overline{\mathcal{G}}} w$.
- If the rule is of the form $(A, X, \sigma) \to (B, X, \sigma)(C, X, \sigma)$ then there exist w_B, w_C such that $w = w_B w_c$ and $(B, X, \sigma) \overset{*}{\Rightarrow}_{C_{\mathcal{G}}} w_B$ and $(C, X, \sigma) \overset{*}{\Rightarrow}_{C_{\mathcal{G}}} w_C$.
  We apply the corresponding rule $(A, X) \to (B, X)(C, X)$ of $\overline{\mathcal{G}}$ from $(A, X)[\bar{z}]$ to obtain $(B, X)[\bar{z}](C, X)[\bar{z}]$. By induction hypothesis, we obtain $(B, X)[\bar{z}] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'_B$, as well as $(C, X)[\bar{z}] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'_C$ with $w_B \preceq w'_B$ and $w_C \preceq w'_C$. As a consequence,
  $$(A, X)[\bar{z}] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'_B w'_C$$
  which yields the result since $w = w_B w_C \preceq w'_B w'_C$.
- If the rule is of the form $(A, X, \sigma) \to (B, Y, \mathsf{push}((f, X) \blacktriangleright \sigma))$, then, by Lemma 8.6, the p-unfolding $\bar{z}'$ of $\mathsf{push}((f, X) \blacktriangleright \sigma)$ satisfies $(B, Y)[(f, X)\bar{z}] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip}} (B, Y)[\bar{z}']$.
  By induction hypothesis, there exists $w' \in T^*$ such that $(B, Y)[\bar{z}'] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'$ and $w \preceq w'$. As a consequence,
  $$(A, X)[\bar{z}] \Rightarrow_{\overline{\mathcal{G}}} (B, Y)[(f, X)\bar{z}] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'.$$
- If the rule is of the form $(A, X, \sigma) \to (B, Y, \sigma')$, with $\sigma = \mathsf{push}((f, Y) \blacktriangleright \sigma')$, then by Lemma 8.7 the $(p-1)$-unfolding $\bar{z}'$ of σ' is such that $(A, X)[\bar{z}] \overset{*}{\rightarrow}_{\mathsf{skip}} (A, X)[(f, Y)\bar{z}']$. By induction hypothesis, there exists $w' \in T^*$ such that
  $$(A, X)[(f, Y)\bar{z}'] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'.$$
  As a consequence,
  $$(A, X)[\bar{z}] \Rightarrow_{\overline{\mathcal{G}}} (A, X)[(f, Y)\bar{z}'] \overset{*}{\Rightarrow}_{\mathsf{pump,\,skip},\,\overline{\mathcal{G}}} w'.$$

We have proven the induction. To obtain the lemma, let $w \in \mathsf{L}(C_{\mathcal{G}})$. There is a derivation in $C_{\mathcal{G}}$ from $(S, \mathbf{U}, \varepsilon)$ to w. Let p be its number of pop steps. Since ε is the p-unfolding of ε, there is a derivation from $(S, \mathbf{U})$ to some w' with $w \preceq w'$. As a result, $w \in \mathsf{L}(\overline{\mathcal{G}})\!\downarrow$. ◻

G.3 Proof of Lemma 8.5

Lemma 8.5. *For each σ and $p \geq 1$:* $\mathsf{unf}_p(\sigma) \overset{*}{\rightarrow}_{\mathsf{skip}} \mathsf{unf}_{p-1}(\sigma)$.

Proof. By induction on the depth d of σ. If $d = 0$ then $\mathsf{unf}_p(\sigma) = \mathsf{unf}_{p-1}(\sigma) = \varepsilon$. If $d > 0$, we distinguish cases according to the shape of σ. If σ is a d-atom $(f, X)\sigma'$ then we simply apply the induction hypothesis. The same goes for a sequence of d-atoms: we apply the previous case to each one of them. In the case of a d-block, we have

$$\begin{aligned}\mathsf{unf}_p(\sigma) = z^u z^v((f, X)\mathsf{unf}_{p-1}(\sigma_1))\cdots\\ \cdots((f, X)\mathsf{unf}_{p-1}(\sigma_r))z^v\mathsf{unf}_p(w)\end{aligned}$$

with $z^u, z^v, \sigma_1, \ldots, \sigma_r$ as in the definition.

- if $p > 1$, then by the previous cases, z^u, z^v, and $\mathsf{unf}_p(w)$ reduce to their $(p-1)$-unfolding counterparts. By induction hypothesis, each $\mathsf{unf}_{p-1}(\sigma_i)$ reduces to $\mathsf{unf}_{p-2}(\sigma_i)$. Hence $\mathsf{unf}_p(\sigma)$ reduces to $\mathsf{unf}_{p-1}(\sigma)$.
- If $p = 1$, then we have
  $$\mathsf{unf}_{p-1}(\sigma) = \mathsf{unf}_0(u_1 \cdots u_N)\mathsf{unf}_0(v_1 \cdots v_N)\mathsf{unf}_0(w).$$
  Further, by definition of a d-block, $\varphi(z^u_i) = \varphi(z^v_i) = e$ for all i. Moreover, for all j we have $\varphi((f, X)\mathsf{unf}_0(\sigma_j)) =$

e, since $\mathsf{push}((f,X)\mathsf{unf}_0(\sigma_j) \blacktriangleright \varepsilon) = \mathsf{push}((f,X) \blacktriangleright \sigma_j) = u_1 \dots u_N e^+ v_1 \dots v_N$ and $\varphi(u_1 \dots u_N e^+ v_1 \dots v_N) = e$. As a consequence, we have

$$\begin{aligned}\mathsf{unf}_p(\sigma) =& z^u z^v ((f,X)\mathsf{unf}_{p-1}(\sigma_1)) \cdots ((f,X)\mathsf{unf}_{p-1}(\sigma_r)) z^v \mathsf{unf}_p(w)\\ &\rightarrow_{\mathsf{skip}} z^u z^v z^w \xrightarrow{*}_{\mathsf{skip}} \mathsf{unf}_{p-1}(\sigma)\end{aligned}$$

Finally, for a summary we can simply apply the previous cases to each of the components. □

G.4 Proof of Lemma 8.6

Lemma 8.6. *Let $p \in \mathbb{N}$, let $(A, X, \sigma_1) \to (B, Y, \sigma_2)$ be a rule of $C_{\mathcal{G}}$ with $\sigma_2 = \mathsf{push}((f,X) \blacktriangleright \sigma_1)$. We have*

$$(B,Y)[(f,X)\mathsf{unf}_p(\sigma_1)] \overset{*}{\Rightarrow}_{\mathsf{pump},\,\mathsf{skip}} (B,Y)[\mathsf{unf}_p(\sigma_2)].$$

Proof. We prove this statement by induction on the depth of σ_2. If σ_2 has depth 0 then it is ε, contradicting $\sigma_2 = \mathsf{push}((f,X) \blacktriangleright \sigma_1)$.

If σ_2 has depth $d > 0$ then we distinguish cases according to its shape. In cases (1) and (ii) we will simply show that $(f,Y)\mathsf{unf}_p(\sigma_1)$ is the p-unfolding of σ_2. In case (a) we will use the induction hypothesis, and in cases (A) and (B) we will actually apply a pump rule and skip rules to obtain $\mathsf{unf}_p(\sigma_2)$.

We decompose σ_1 as $\sigma_1 = \sigma' u B_1 \dots B_k$. By definition we have $\mathsf{unf}_p(\sigma_1) = \mathsf{unf}_p(\sigma')\mathsf{unf}_p(u)\mathsf{unf}_p(B_1)\cdots\mathsf{unf}_p(B_k)$. We follow the cases in the definition of $\mathsf{push}((f,X) \blacktriangleright \sigma_1)$.

(1) If $\mathbf{depth}((f,X)\sigma_1) > d$ then $\sigma_2 = (f,X)\sigma_1$. Then by definition $\mathsf{unf}_p(\sigma_2) = (f,X)\mathsf{unf}_p(\sigma_1)$.

(2) Otherwise, we have $\mathbf{depth}((f,X)\sigma_1) = d$

 (a) if $\mathbf{depth}((f,X)\sigma') < d$ then

$$\sigma_2 = (\mathsf{push}((f,X) \blacktriangleright \sigma'))uB_1 \dots B_k.$$

 By induction hypothesis, we have

$$(B,Y)[(f,X)\mathsf{unf}_p(\sigma')] \overset{*}{\Rightarrow}_{\mathsf{pump},\,\mathsf{skip}} (B,Y)[z'']$$

 with $z'' = \mathsf{unf}_p(\mathsf{push}((f,X) \blacktriangleright \sigma'))$. Therefore,

$$\begin{aligned}&(B,Y)[\mathsf{unf}_p(\sigma')\mathsf{unf}_p(u)\mathsf{unf}_p(B_1)\cdots\mathsf{unf}_p(B_k)]\\ &\overset{*}{\Rightarrow}_{\mathsf{pump},\,\mathsf{skip}} (B,Y)[z'' z^u z_1 \dots z_k]\\ &= (B,Y)[\sigma_2].\end{aligned}$$

 (b) Otherwise, $\mathbf{depth}((f,X)\sigma') = d$ and $(f,X)\sigma'$ is a d-atom.

 (i) If $((f,X)\sigma')u$ is of the form $u_1 \dots u_N v_0 v_1 \dots v_N w$ with $\varphi(u_i) = \varphi(v_i) = \varphi(v_0) = e$ for all $i \geq 1$, for some $e \in \mathbf{Idem}(\mathbb{M})$, then

$$\begin{aligned}(f,X)\mathsf{unf}_p(\sigma') = \mathsf{unf}_p(u_1) \dots \mathsf{unf}_p(u_N)\mathsf{unf}_p(v_0) \dots\\ \dots \mathsf{unf}_p(v_N)\mathsf{unf}_p(w).\end{aligned}$$

 Furthermore since σ_2 is obtained by pushing (f,X) we must have $\beta(f) = B$ and $Y = f \cdot X$. Furthermore, since (B,Y,σ_2) is a non-terminal of $C_{\mathcal{G}}$, σ_2 must be feasible, hence $e \neq \mathbf{0}_{\mathbb{M}}$. This means that $e = (B,Y,M,C,Y)$ for some C and M. We have two cases.

 (A) If there exists j such that B_j is of the form

$$u'_1 \dots u'_N e^+ v'_1 \dots v'_N w'$$

 and $\varphi(v_1 \dots v_N w B_1 \dots B_{j-1} u'_1 \dots u'_N) = e$. Then this implies

$$\sigma_2 = (u_1 \dots u_N e^+ v'_1 \dots v'_N w')B_{j+1} \dots B_k.$$

 In that case, we also have

$$\begin{aligned}\mathsf{unf}_p(B_j) =& \mathsf{unf}_p(u'_1) \dots \mathsf{unf}_p(u'_N) z'_e\\ & \mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')\end{aligned}$$

 with $\varphi(z'_e) = e$.

 Let $\tilde{z}$ be the p-unfolding of the summary $u_1 \dots u_N e^+ v'_1 \dots v'_N w'$. Then this must be of the form $z''\mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')$ with $\varphi(z'') = e$. Since $e = (B,Y,M,C,Y)$ is idempotent, we can apply pump rules and skip rules as follows:

$$\begin{aligned}
&(B,Y) && [(f,X)\mathsf{unf}_p(\sigma_1)]\\
=&(B,Y) && [\mathsf{unf}_p(u_1) \dots \mathsf{unf}_p(u_N)\\
& && \mathsf{unf}_p(v_0) \dots \mathsf{unf}_p(v_N)\mathsf{unf}_p(w)\\
& && \mathsf{unf}_p(B_1) \dots \mathsf{unf}_p(B_{j-1})\\
& && \mathsf{unf}_p(u'_1) \dots \mathsf{unf}_p(u'_N) z'_e\\
& && \mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')\\
& && \mathsf{unf}_p(B_{j+1}) \dots \mathsf{unf}_p(B_k)]\\
\rightarrow_{\mathsf{pump}}&(B,Y) && [\mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\\
& && \mathsf{unf}_p(u_1) \dots \mathsf{unf}_p(u_N)\\
& && \mathsf{unf}_p(v_1) \dots \mathsf{unf}_p(v_N)\mathsf{unf}_p(w)\\
& && \mathsf{unf}_p(B_1) \dots \mathsf{unf}_p(B_{j-1})\\
& && \mathsf{unf}_p(u'_1) \dots \mathsf{unf}_p(u'_N) z'_e\\
& && \mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')\\
& && \mathsf{unf}_p(B_{j+1}) \dots \mathsf{unf}_p(B_k)]\\
\rightarrow_{\mathsf{skip}}&(B,Y) && [\mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')\\
& && \mathsf{unf}_p(B_{j+1}) \dots \mathsf{unf}_p(B_k)]\\
\rightarrow_{\mathsf{pump}}&(B,Y) && [z''\mathsf{unf}_p(v'_1) \dots \mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')\\
& && \mathsf{unf}_p(B_{j+1}) \dots \mathsf{unf}_p(B_k)]\\
=&(B,Y) && [\mathsf{unf}_p(\sigma_2)]
\end{aligned}$$

 (B) Otherwise we have the summary

$$\sigma_2 = (u_1 \dots u_N e^+ v_1 \dots v_N w)B_1 \dots B_k.$$

 We define $B = u_1 \dots u_N e^+ v_1 \dots v_N w$. Then the p-unfolding $\mathsf{unf}_p(B)$ is of the form

$$z''\mathsf{unf}_p(v_1) \dots \mathsf{unf}_p(v_N)\mathsf{unf}_p(w)$$

 with $\varphi(z'') = e$. Since $e = (B,Y,M,C,Y)$ is idempotent, we can apply pump rules and skip rules as follows:

$$
\begin{aligned}
&(B,Y) && [(f,X)\mathsf{unf}_p(\sigma_1)] \\
=&(B,Y) && [\mathsf{unf}_p(u_1)\dots\mathsf{unf}_p(u_N) \\
& && \mathsf{unf}_p(v_0)\mathsf{unf}_p(v_1)\dots\mathsf{unf}_p(v_N)\mathsf{unf}_p(w) \\
& && \mathsf{unf}_p(B_1)\dots\mathsf{unf}_p(B_k)] \\
\rightarrow_{\mathsf{pump}}&(B,Y) && [\mathsf{unf}_p(v_1)\dots\mathsf{unf}_p(v_N) \\
& && \mathsf{unf}_p(u_1)\dots\mathsf{unf}_p(u_N) \\
& && \mathsf{unf}_p(v_1)\dots\mathsf{unf}_p(v_N)\mathsf{unf}_p(w) \\
& && \mathsf{unf}_p(B_1)\dots\mathsf{unf}_p(B_k)] \\
\rightarrow_{\mathsf{skip}}&(B,Y) && [\mathsf{unf}_p(v_1)\dots\mathsf{unf}_p(v_N) \\
& && \mathsf{unf}_p(B_1)\dots\mathsf{unf}_p(B_k)] \\
\rightarrow_{\mathsf{pump}}&(B,Y) && [z''\mathsf{unf}_p(v_1)\dots\mathsf{unf}_p(v_N) \\
& && \mathsf{unf}_p(B_{j+1})\dots\mathsf{unf}_p(B_k)] \\
=&(B,Y) && [\mathsf{unf}_p(\sigma_2)]
\end{aligned}
$$

The resulting stack content is the p-unfolding of σ_2.

(ii) Otherwise, we have $\sigma_2 = ((f,X)\sigma')uB_1\dots B_k$ and thus $(f,X)\mathsf{unf}_p(\sigma_1) = \mathsf{unf}_p(\sigma_2)$.

□

G.5 Proof of Lemma 8.7

Lemma 8.7. *Let $p \geq 1$, let $(A,X,\sigma_1) \rightarrow (B,Y,\sigma_2)$ a rule of $C_{\mathcal{G}}$ with $\sigma_2 \in \mathsf{pop}((f,Y) \blacktriangleleft \sigma_1)$. We have*

$$(A,X)[\mathsf{unf}_p(\sigma_1)] \xrightarrow{*}_{\mathsf{skip}} (A,X)[(f,Y)\mathsf{unf}_{p-1}(\sigma_2)].$$

Proof of Lemma 8.7. We prove this statement by induction on the depth of σ_1. If $\mathbf{depth}(\sigma_1) = 0$ then it is ε and $\mathsf{pop}((f,Y) \blacktriangleleft \sigma_1)$ is empty.

If σ_1 has depth $d > 0$ then we distinguish cases according to its shape. Recall that by Lemma 8.5 a p-unfolding of a summary σ always reduces to its $(p-1)$-unfolding through skip rules. We will use this fact often throughout the proof.

Let $\sigma_2 = \sigma' uB_1\dots B_k$.

(1) If $\mathbf{depth}((f,Y)\sigma_2) > d$ then $\sigma_1 = (f,Y)\sigma_2$. Then by definition $\mathsf{unf}_p(\sigma_1) = (f,Y)\mathsf{unf}_p(\sigma_2)$, hence

$$(A,X)[\mathsf{unf}_p(\sigma_1)] \xrightarrow{*}_{\mathsf{skip}} (A,X)[(f,X)\mathsf{unf}_{p-1}(\sigma_2)]$$

by Lemma 8.5.

(2) Otherwise, we have $\mathbf{depth}((f,Y)\sigma_2) = d$

(a) if $\mathbf{depth}((f,Y)\sigma') < d$ then

$$\sigma_1 = (\mathsf{push}((f,Y) \blacktriangleright \sigma'))uB_1\dots B_k.$$

Then we have

$$\mathsf{unf}_p(\sigma_1) =$$
$$\mathsf{unf}_p(\mathsf{push}((f,Y) \blacktriangleright \sigma'))\mathsf{unf}_p(u)\mathsf{unf}_p(B_1)\dots\mathsf{unf}_p(B_k).$$

Since $(A,X,\sigma_1) \rightarrow (B,Y,\sigma_2)$ is a rule of $C_{\mathcal{G}}$, σ_1 and σ_2 must be feasible, thus so are σ', and $\mathsf{push}((f,Y) \blacktriangleright \sigma')$. Thus, by construction of $C_{\mathcal{G}}$, we must have the rule $(A,X,\mathsf{push}((f,Y) \blacktriangleright \sigma')) \rightarrow (B,Y,\sigma')$ in $C_{\mathcal{G}}$. Hence, by induction hypothesis,

$$(A,X)[\mathsf{unf}_p(\mathsf{push}((f,Y) \blacktriangleright \sigma'))] \xrightarrow{*}_{\mathsf{skip}} (A,X)[(f,Y)\mathsf{unf}_{p-1}(\sigma')].$$

As a result, we have

$$
\begin{aligned}
&(A,X)[\mathsf{unf}_p(\sigma_1)] \\
&\xrightarrow{*}_{\mathsf{skip}} (A,X)[(f,Y)\mathsf{unf}_{p-1}(\sigma')\mathsf{unf}_p(u)\mathsf{unf}_p(B_1)\dots\mathsf{unf}_p(B_k)]
\end{aligned}
$$

By Lemma 8.5, we thus have

$$
\begin{aligned}
&(A,X)[\mathsf{unf}_p(\sigma_1)] \\
\xrightarrow{*}_{\mathsf{skip}}&(A,X)[(f,Y)\mathsf{unf}_{p-1}(\sigma')\mathsf{unf}_{p-1}(u)\mathsf{unf}_{p-1}(B_1)\dots\mathsf{unf}_{p-1}(B_k)] \\
=&(A,X)[(f,Y)\mathsf{unf}_{p-1}(\sigma_2)]
\end{aligned}
$$

(b) Otherwise, $\mathbf{depth}((f,Y)\sigma') = d$ and $(f,Y)\sigma'$ is a d-atom.

(i) If $((f,Y)\sigma')u$ is of the form $u_1\dots u_N v_0 v_1\dots v_N w$ with $\varphi(u_i) = \varphi(v_i) = \varphi(v_0) = e$ for all $i \geq 1$, for some $e \in \mathbf{Idem}(\mathbb{M})$, we have two cases.

(A) If there exists j such that B_j is of the form $u'_1\dots u'_N e^+ v'_1\dots v'_N w'$ and

$$\varphi(v_1\dots v_N w B_1\dots B_{j-1}u'_1\dots u'_N) = e$$

then we pick the maximal such j. We have $\sigma_1 = BB_{j+1}\dots B_k$, where

$$B = u_1\dots u_N e^+ v'_1\dots v'_N w'.$$

In that case,

$$\mathsf{unf}_p(\sigma_1) = \mathsf{unf}_p(B)\mathsf{unf}_p(B_{j+1})\dots\mathsf{unf}_p(B_k).$$

Let $B'_j = u'_1\dots u'_N e^+ v'_1\dots v'_N$, that is, B_j without the w' suffix. Let us also define $z' = \mathsf{unf}_{p-1}(\sigma' uB_1\dots B_{j-1}B'_j)$. Then $(f,Y)z'$ is of the form

$$\mathsf{unf}_{p-1}(u_1)\cdots\mathsf{unf}_{p-1}(u_N)z'_e\mathsf{unf}_{p-1}(v'_1)\cdots\mathsf{unf}_{p-1}(v'_N)$$

for some z'_e such that $\varphi(z'_e) = e$.

By definition of the p-unfolding, since

$$\mathsf{push}((f,X) \blacktriangleright \sigma' uB_1\dots B_{j-1}B'_j) = B$$

and $p \geq 1$, $\mathsf{unf}_p(B)$ is of the form

$$\mathsf{unf}_p(u_1)\dots\mathsf{unf}_p(u_N)z_-(f,Y)z'z_+\mathsf{unf}_p(v'_1)\cdots\mathsf{unf}_p(v'_N)\mathsf{unf}_p(w')$$

with $\varphi(z_-) = \varphi(z_+) = e$.

We can use skip rules:

$$\begin{aligned}
(A,X) \quad & [\mathsf{unf}_p(\sigma_1)] \\
=(A,X) \quad & [\mathsf{unf}_p(B)\mathsf{unf}_p(B_{j+1})\dots\mathsf{unf}_p(B_k)] \\
=(A,X) \quad & [\mathsf{unf}_p(u_1)\dots\mathsf{unf}_p(u_N)z_-(f,Y)z'z_+ \\
& \mathsf{unf}_p(v'_1)\cdots\mathsf{unf}_p(v'_N) \\
& \mathsf{unf}_p(w')\mathsf{unf}_p(B_{j+1})\dots\mathsf{unf}_p(B_k)] \\
\xrightarrow{*}_{\mathsf{skip}}(A,X) \quad & [\mathsf{unf}_{p-1}(u_1)\dots\mathsf{unf}_{p-1}(u_N)z_- \\
& (f,Y)z'z_+\mathsf{unf}_{p-1}(v'_1)\cdots\mathsf{unf}_{p-1}(v'_N) \\
& \mathsf{unf}_{p-1}(w')\mathsf{unf}_{p-1}(B_{j+1})\dots\mathsf{unf}_{p-1}(B_k)] \\
=(A,X) \quad & [\mathsf{unf}_{p-1}(u_1)\dots\mathsf{unf}_{p-1}(u_N)z_- \\
& \mathsf{unf}_{p-1}(u_1)\cdots\mathsf{unf}_{p-1}(u_N)z'_e \\
& \mathsf{unf}_{p-1}(v'_1)\cdots\mathsf{unf}_{p-1}(v'_N) \\
& z_+\mathsf{unf}_{p-1}(v'_1)\cdots\mathsf{unf}_{p-1}(v'_N)\mathsf{unf}_{p-1}(w') \\
& \mathsf{unf}_{p-1}(B_{j+1})\dots\mathsf{unf}_{p-1}(B_k)] \\
\rightarrow_{\mathsf{skip}}{}^2(A,X) \quad & [\mathsf{unf}_{p-1}(u_1)\dots\mathsf{unf}_{p-1}(u_N)z'_e \\
& \mathsf{unf}_{p-1}(v'_1)\cdots\mathsf{unf}_{p-1}(v'_N) \\
& \mathsf{unf}_{p-1}(w')\mathsf{unf}_{p-1}(B_{j+1})\dots\mathsf{unf}_{p-1}(B_k)] \\
=(A,X) \quad & [(f,Y)z'\mathsf{unf}_{p-1}(B_{j+1})\dots\mathsf{unf}_{p-1}(B_k)] \\
=(A,X) \quad & [(f,Y)\mathsf{unf}_{p-1}(\sigma_2)]
\end{aligned}$$

(B) Otherwise our summary is of the form $\sigma_1 = (u_1\dots u_N e^+ v_1\dots v_N w)B_1\dots B_k$. In particular, its p-unfolding is the stack content $\mathsf{unf}_p(\sigma_1) = \mathsf{unf}_p(B)\mathsf{unf}_p(B_1)\cdots\mathsf{unf}_p(B_k)$. Since $(f,Y)\sigma' u = u_1\dots u_N v_0\dots v_N w$, the $(p-1)$-unfolding of $u_1\dots u_N v_0\dots v_N$ must be of the form $(f,Y)z'$ for some z'. Hence

$$(f,Y)z' = \mathsf{unf}_{p-1}(u_1)\cdots\mathsf{unf}_{p-1}(u_N)\mathsf{unf}_{p-1}(v_0)\cdots\mathsf{unf}_{p-1}(v_N).$$

By definition of the p-unfolding, $\mathsf{unf}_p(B)$ is of the form

$$\mathsf{unf}_p(u_1)\cdots\mathsf{unf}_p(u_N)z_-(f,Y)z'z_+\mathsf{unf}_p(v_1)\cdots\mathsf{unf}_p(v_N)\mathsf{unf}_p(w)$$

with $\varphi(z_-) = \varphi(z_+) = e$.

We can use skip rules:

$$\begin{aligned}
(A,X) \quad & [\mathsf{unf}_p(\sigma_1)] \\
=(A,X) \quad & [\mathsf{unf}_p(B)\mathsf{unf}_p(B_1)\cdots\mathsf{unf}_p(B_k)] \\
=(A,X) \quad & [\mathsf{unf}_p(u_1)\cdots\mathsf{unf}_p(u_N)z_- \\
& (f,Y)z'z_+\mathsf{unf}_p(v_1)\cdots\mathsf{unf}_p(v_N) \\
& \mathsf{unf}_p(w)\mathsf{unf}_p(B_1)\cdots\mathsf{unf}_p(B_k)] \\
=(A,X) \quad & [\mathsf{unf}_p(u_1)\cdots\mathsf{unf}_p(u_N)z_- \\
& \mathsf{unf}_{p-1}(u_1)\cdots\mathsf{unf}_{p-1}(u_N) \\
& \mathsf{unf}_{p-1}(v_0)\mathsf{unf}_{p-1}(v_1)\cdots\mathsf{unf}_{p-1}(v_N) \\
& z_+\mathsf{unf}_p(v_1)\cdots\mathsf{unf}_p(v_N)\mathsf{unf}_p(w) \\
& \mathsf{unf}_p(B_1)\cdots\mathsf{unf}_p(B_k)] \\
\xrightarrow{*}_{\mathsf{skip}}(A,X) \quad & [\mathsf{unf}_{p-1}(u_1)\cdots\mathsf{unf}_{p-1}(u_N)z_- \\
& \mathsf{unf}_{p-1}(u_1)\cdots\mathsf{unf}_{p-1}(u_N) \\
& \mathsf{unf}_{p-1}(v_0)\mathsf{unf}_{p-1}(v_1)\cdots\mathsf{unf}_{p-1}(v_N) \\
& z_+\mathsf{unf}_{p-1}(v_1)\cdots\mathsf{unf}_{p-1}(v_N)\mathsf{unf}_{p-1}(w) \\
& \mathsf{unf}_{p-1}(B_1)\cdots\mathsf{unf}_{p-1}(B_k)] \\
\rightarrow_{\mathsf{skip}}{}^2(A,X) \quad & [(f,Y)z'\mathsf{unf}_{p-1}(w)\mathsf{unf}_{p-1}(B_1)\cdots\mathsf{unf}_{p-1}(B_k)] \\
=(A,X) \quad & [\mathsf{unf}_{p-1}(u_1\dots u_N v_0\dots v_N) \\
& \mathsf{unf}_{p-1}(w)\mathsf{unf}_{p-1}(B_1)\cdots\mathsf{unf}_{p-1}(B_k)] \\
=(A,X) \quad & [\mathsf{unf}_{p-1}(\sigma_2)]
\end{aligned}$$

(ii) Otherwise, we have the summary

$$\sigma_1 = ((f,Y)\sigma')uB_1\dots B_k$$

and thus $\mathsf{unf}_p(\sigma_1) = (f,Y)\mathsf{unf}_p(\sigma_2)$. According to Lemma 8.5, we obtain the derivation

$$(A,X)[\sigma_1] \xrightarrow{*}_{\mathsf{skip}} (A,X)[(f,Y)\mathsf{unf}_{p-1}(\sigma_2)].$$

□

H ADDITIONAL MATERIAL FROM SECTION 9

H.1 Proofs for NFA lower bound

Lemma H.1. *There is an derivation of $\mathcal{G}_n$ that produces the word $\mathsf{a}^{\exp_3(n)}$*

Proof. Let $\alpha_1\cdots\alpha_m \in \Sigma_n^*$ be the unique word accepted by all $\mathcal{A}_i$, with $m = 2^n$. For all $b_1,\dots,b_m \in \{0,1\}$, we write $\overline{b_1\cdots b_m}$ for the number in $[0, 2^m - 1]$ whose binary representation over m bits is $b_1\cdots b_m$, where b_1 is the least significant digit.

We show that for all $b_1,\dots,b_m \in \{0,1\}$, if $M = \overline{b_1\cdots b_m}$ then $Z(\alpha_1,b_1)\cdots(\alpha_m,b_m)\bot$ produces $\mathsf{a}^{2^{2^m-M}}$, by induction on $2^m - M$.

We can apply $Z \xrightarrow{\mathcal{B}_1,\dots,\mathcal{B}_n} D$ and $D \rightarrow AA$ to obtain two copies of $A(\alpha_1,b_1)\cdots(\alpha_m,b_m)\bot$. This is because $\alpha_1\cdots\alpha_m$ is accepted by all $\mathcal{A}_i$. We are left with

$$A[(\alpha_1,b_1)\cdots(\alpha_m,b_m)\bot]A[(\alpha_1,b_1)\cdots(\alpha_m,b_m)\bot].$$

If $b_i = 1$ for all i, then we can apply $A(\alpha_i,1) \rightarrow A$ for each i and then $A\bot \rightarrow F$ and $F \rightarrow \mathsf{a}$ to obtain aa, which is what we want since we would then have $M = 2^m - 1$ and thus $\mathsf{a}^{2^{2^m-M}} = \mathsf{a}^2$.

Otherwise, let j be the least index such that $b_j = 0$. It suffices to show that $A(\alpha_1, b_1)\cdots(\alpha_m, b_m)\bot$ produces $a^{2^{\overline{2^m-M-1}}}$. We have $M+1 = \overline{1^{j-1}0b_{j+1}\cdots b_m} + 1 = \overline{0^{j-1}1b_{j+1}\cdots b_m}$.

We can apply:

- $A(\alpha_i, 1) \to A$ for each $i < j$ until we get $A(\alpha_j, 0)(\alpha_{j+1}, b_{j+1})\cdots(\alpha_m, b_m)\bot$,
- then apply $A(\alpha_j, 0) \to B$ and $B \to Z(\alpha_j, 1)$ to get $Z(\alpha_j, 1)(\alpha_{j+1}, b_{j+1})\cdots(\alpha_m, b_m)\bot$,
- and $Z \to Z(\alpha_i, 0)$ for each $i < j$, in decreasing order, until we obtain $Z(\alpha_1, 0)\cdots(\alpha_{j-1}, 0)(\alpha_j, 1)(\alpha_{j+1}, b_{j+1})\cdots(\alpha_m, b_m)\bot$, which produces $a^{2^{2^m-M-1}}$ by induction hypothesis.

The induction is proved. To obtain the lemma, it suffices to start with S, apply $S \to Z\bot$ and then $Z \to Z(\alpha_i, 0)$ for each $i \in [1, n]$ in decreasing order. We get $Z(\alpha_1, 0)\cdots(\alpha_n, 0)$, which produces $a^{\exp_3(n)}$ by applying the induction with $M = 0$. ◻

Lemma H.2. *For all $E \in N_n$ and $z \in I_n^*$ the language $\mathsf{L}_\emptyset(Ez)$ contains at most one word. In particular, $\mathsf{L}(\mathcal{G}_n)$ is empty or a singleton.*

Proof. We prove this for each $E \in N_n$, one by one.

(1) We first observe that from a configuration Zz there is always at most one rule which can lead to a complete derivation tree: if z does not contain $\bot$ then $\mathsf{L}_\emptyset(Zz) = \emptyset$. If $z = (\alpha_1, b_1)\cdots(\alpha_m, b_m)\bot z'$ then:
   - if $m \neq 2^n$ we cannot apply $Z \xrightarrow{\mathcal{B}_1,\ldots,\mathcal{B}_n} D$ since z has no prefix accepted by all $\mathcal{B}_i$. Furthermore, if $m > 2^n$, we have $\mathsf{L}_\emptyset(Zz) = \emptyset$ since we can only push more pairs $(\alpha, 0)$ on the stack, so we will never be able to apply $Z \xrightarrow{\mathcal{B}_1,\ldots,\mathcal{B}_n} D$.
   - if $m = 2^n$ then we cannot apply $Z \to Z(\alpha, 0)$, by the previous item, as we would obtain more than 2^n symbols before the first $\bot$.

(2) Note that B, D and S all have a single rule. Meanwhile, A has several but the top stack symbol determines which rule can be applied. In conclusion, from every configuration Ez, there is at most one rule that can be applied to lead to a complete derivation. As a consequence, the language of $\mathcal{G}$ contains at most one word.

◻

By combining the two previous statements we conclude that $\mathcal{G}_n$ recognizes the singleton language $\{a^{\exp_3(n)}\}$, while having size only quadratic in n. A trim NFA for this language must be acyclic, as otherwise it would recognize an infinite language, and thus have at least $\exp_3(n)$ states.

H.2 Computational hardness

We now use methods from [58] to derive Theorem 3.5 and co-3-NEXP-hardness in Theorem 3.6 from our construction above. For this, we rely on the notion of $\Delta(f)$ language classes [58], which requires some terminology. A *transducer* is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, E, q_0, F)$, where Q is a finite set of *states*, Σ is its *input alphabet*, Γ is its *output alphabet*, $E \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$ is its finite set of *edges*, $q_0 \in Q$ is its *initial state*, and $F \subseteq Q$ is its set of *final states*. It describes a relation $\mathsf{R}(\mathcal{T}) \subseteq \Sigma^* \times \Gamma^*$, namely the set of all pairs (u, v) for which there are decompositions $u = u_1\cdots u_n$ and $v = v_1\cdots v_n$, states $q_0, q_1, \ldots, q_n$, and edges $(q_{i-1}, u_i, v_i, q_i) \in E$ for $i = 1, \ldots, n$ with $q_n \in F$. For a language $L \subseteq \Sigma^*$, we write $\mathcal{T}(L) = \{v \in \Gamma^* \mid \exists u \in L\colon (u, v) \in \mathsf{R}(\mathcal{T})\}$.

A *language class* is a class of formal languages, together with some means to represent them, such as grammars or automata. A language class C is an *effective full trio* if for a given language L from C, we can effectively compute a description of $\mathcal{T}(L)$. Now suppose $f\colon \mathbb{N} \to \mathbb{N}$ is an amplifying function meaning there is a polynomial p such that $f(p(n)) \geq f(n)^2$. Then, C is said to be $\Delta(f)$ if (i) computing $\mathcal{T}(L)$ can be done in polynomial time and (ii) given n, one can compute a description of the language $\{a^{f(n)}\}$ in polynomial time.

In these terms, Theorem 3.2 tells us that the class of indexed languages (represented by indexed grammars) are $\Delta(\exp_3)$: Applying rational transductions to indexed languages is well-known to be possible in polynomial time (see, e.g. [56, Section 3.1]).

Proposition H.3. *The indexed languages (represented by indexed grammars) are $\Delta(\exp_3)$.*

We will also need the notion of simple substitutions. For alphabets Σ, Γ, a *substitution* is a map $\sigma\colon \Sigma \to 2^{\Gamma^*}$ that replaces each letter in Σ by a language over Γ. For language $L \subseteq \Sigma^*$, the language $\sigma(L)$ is defined in the obvious way. The substitution σ is said to be *simple for* $L \subseteq \Sigma^*$ if $\Sigma \subseteq \Gamma$ and there is a letter $a \in \Sigma$ such that $\sigma(a') = \{a'\}$ for each $a' \in \Sigma \setminus \{a\}$. We say that a language class C is *closed under simple subsititutions* if for any given L from C, and any simple substitution σ for L, the language $\sigma(L)$ belongs to C, and a representation can be computed in polynomial time. It is easy to see that the indexed languages are closed under simple substitutions. In [58, Theorem 15], it is shown that downward closure inclusion and equivalence are both coNTIME(t)-hard for any language class that is $\Delta(t)$ and closed under simple substitutions. Hence, Theorem 3.2 implies co-3-NEXP-hardness of downward closure inclusion and equivalence.

H.3 Proofs for DFA lower bound

Here, we prove a slightly more general result than discussed in Section 9: We show that for any $\Delta(f)$ language class, DFAs for downward closures of languages with polynomial-sized descriptions require at least size $2^{f(n)}$, provided that the language class is also closed under simple subsitution:

Proposition H.4. *Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function such that a language class C is $\Delta(f)$ and let C be closed under simple substitutions. Then there is a family languages $(L_n)_{n\geq 1}$ with polynomial description sizes such that any DFA for $L_n\!\downarrow$ requires at least $2^{f(n)}$ states.*

Proof. We claim that we can construct a representation of

$$L_n = \{uv \mid u, v \in \{0, 1\}^*,\ |u| = |v| = f(n),\ u \neq v\}$$

in polynomial time. Note that a DFA for $L_n\!\downarrow$ requires at least $2^{f(n)}$ states: After reading distinct prefixes $u, u' \in \{0, 1\}^*$ of length $f(n)$, the DFA must enter distinct states, as otherwise, it would accept uu, which does not belong to $L_n\!\downarrow$.

To construct L_n for given $n \in \mathbb{N}$, we begin by building a representation of $\{a^{f(n)}\}$. Using a transducer, we then insert a single occurrence of a letter b into every word, and then substitute this b with

$\{\mathtt{ba}^{f(n)}\mathtt{c}\}$. This yields the language $\{\mathtt{a}^r\mathtt{ba}^{f(n)}\mathtt{ca}^s \mid r+s=f(n)\}$. Using another transducer, we can remove two occurrences of a within $\mathtt{a}^r$ and $\mathtt{a}^s$ to obtain $\{\mathtt{a}^r\mathtt{ba}^{f(n)}\mathtt{ca}^s \mid r+s=f(n)-2\}$. A final transducer then replaces (i) each $\mathtt{a}$ with $\mathtt{0}$ or $\mathtt{1}$ and (ii) $\mathtt{b}$ and $\mathtt{c}$ by distinct letters in $\{\mathtt{0},\mathtt{1}\}$. This results in the language L_n. □

Now, Proposition H.4 and Proposition H.3 together directly imply Theorem 3.5.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009