Subcubic Certificates for CFL Reachability

DMITRY CHISTIKOV, University of Warwick, UK
RUPAK MAJUMDAR, Max Planck Institute for Software Systems, Germany
PHILIPP SCHEPPER, CISPA Helmholtz Center for Information Security, Germany

Many problems in interprocedural program analysis can be modeled as the context-free language (CFL) reachability problem on graphs and can be solved in cubic time. Despite years of efforts, there are no known truly sub-cubic algorithms for this problem. We study the related *certification* task: given an instance of CFL reachability, are there small and efficiently checkable certificates for the existence and for the non-existence of a path? We show that, in both scenarios, there exist succinct certificates ($O(n^2)$ in the size of the problem) and these certificates can be checked in subcubic (matrix multiplication) time. The certificates are based on grammar-based compression of paths (for reachability) and on invariants represented as matrix inequalities (for non-reachability). Thus, CFL reachability lies in nondeterministic and co-nondeterministic *subcubic* time.

A natural question is whether faster algorithms for CFL reachability will lead to faster algorithms for combinatorial problems such as Boolean satisfiability (SAT). As a consequence of our certification results, we show that there cannot be a fine-grained reduction from SAT to CFL reachability for a conditional lower bound stronger than n^ω, unless the nondeterministic strong exponential time hypothesis (NSETH) fails. In a nutshell, reductions from SAT are unlikely to explain the cubic bottleneck for CFL reachability.

Our results extend to related subcubic equivalent problems: pushdown reachability and 2NPDA recognition; as well as to all-pairs CFL reachability. For example, we describe succinct certificates for pushdown non-reachability (inductive invariants) and observe that they can be checked in matrix multiplication time. We also extract a new hardest 2NPDA language, capturing the "hard core" of all these problems.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; **Formal languages and automata theory**.

Additional Key Words and Phrases: CFL reachability, subcubic certification, pushdown reachability

ACM Reference Format:
Dmitry Chistikov, Rupak Majumdar, and Philipp Schepper. 2022. Subcubic Certificates for CFL Reachability. *Proc. ACM Program. Lang.* 6, POPL, Article 41 (January 2022), 29 pages. https://doi.org/10.1145/3498702

1 INTRODUCTION

Context-free reachability is a fundamental problem in interprocedural program analysis, verification of recursive programs, and database theory [Alur et al. 2005; Bouajjani et al. 1997; Dolev et al. 1982; Melski and Reps 2000; Reps et al. 1995; Yannakakis 1990]. For a fixed context-free language (CFL) $\mathcal{L}$ over an alphabet Σ, given a directed graph $G = (V, E)$, an edge-labeling function $\lambda : E \to \Sigma$, and two vertices $s, t \in V$, the $\mathcal{L}$-reachability problem asks if there is a path from s to t in G such that the word formed by concatenating the labels along the path belongs to $\mathcal{L}$. It is well-known that the problem can be solved in time cubic in the size of the graph for any fixed CFL.

Authors' addresses: Dmitry Chistikov, Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, Coventry, UK, d.chistikov@warwick.ac.uk; Rupak Majumdar, Max Planck Institute for Software Systems, Kaiserslautern, Germany, rupak@mpi-sws.org; Philipp Schepper, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany, philipp.schepper@cispa.saarland.

2475-1421/2022/1-ART41
https://doi.org/10.1145/3498702

However, despite many years of efforts, we only know speedups by logarithmic factors (i.e., to $O(n^3/\log n)$, where $n = |V|$) [Chaudhuri 2008; Rytter 1985], leading to a conjecture that no better algorithms (that is, $O(n^{3-\varepsilon})$ for $\varepsilon > 0$) are possible for this and several related problems [Heintze and McAllester 1997]. In recent years, a number of results in fine-grained complexity give credence to the conjecture by demonstrating various conditional lower bounds for the problem [Abboud et al. 2015; Chatterjee et al. 2018; Mathiasen and Pavlogiannis 2021]. Even so, the possibility of algorithms with running time n^ω or above has not been ruled out. Here, $\omega < 2.4$ is the matrix multiplication exponent [Coppersmith and Winograd 1990; Vassilevska Williams 2012].

In this paper, we study the related problem of *certifying* an instance of CFL reachability. Intuitively, this problem asks for easily verifiable proofs of inclusion or non-inclusion. Given a (positive or negative) instance of CFL reachability, we ask if there is an efficiently checkable proof that will convince anyone that the instance is indeed positive or negative.

Formally, a *certificate system* for CFL reachability consists of two algorithms (the checkers), one for positive instances and one for negative instances. Each checker takes as input an instance of the problem and an additional string (called the certificate) and accepts or rejects. The positive (resp. negative) checker is *complete* if for each positive (resp. negative) instance, there is a certificate that makes it accept, and *sound* if for each negative (resp. positive) instance, there is no certificate that makes it accept. Of course, since the instance can be decided in cubic time, a certificate system is non-trivial only if the checkers run in subcubic time (in the size of the instance).

Our main result shows that **CFL reachability has subcubic certificate systems**: every positive or negative instance has a *quadratic* certificate and a checker that runs in $O(n^\omega)$ time, $n = |V|$.

- For a positive instance of the problem, a naive certificate is a path from s to t witnessing inclusion. Unfortunately, this is not an efficient certificate, since it is known that the shortest path can be exponentially long in the size of the graph. We show that the shortest path is well-compressible by a context free grammar of size $O(n^2)$ in the number of vertices of the graph. Moreover, given such a compressed representation, there is a checker verifying in time $O(n^2)$ that the grammar indeed encodes a witness path.
- For a negative instance of the problem, a certificate is an inductive invariant that demonstrates *non*-reachability. We show that such an inductive invariant can be represented using relations between a constant number of $n \times n$ matrices, and there is a checker verifying in time $O(n^\omega)$ that such an encoding does represent an inductive invariant. Additionally, if we allow randomization, there is a randomized checker running in $O(n^2)$ time.
- Certificates of the above two kinds can be combined into a single subcubic certificate for the *all-pairs* version of the problem, where the task is to identify all pairs of vertices s, t such that t is $\mathcal{L}$-reachable from s. (The definition of certificate systems is extended appropriately.)

Summing up, CFL reachability can be certified in subcubic time. This result illuminates a conceptually new aspect of an old problem. Certificate systems make it possible to separate two possibly independent phases of computation, *finding* a solution to a computational problem and *verifying* it.

We consider complexity-theoretic implications of our result. Impagliazzo and Paturi [2001] introduced the strong exponential time hypothesis (SETH), which informally states that SAT has no algorithms better than exhaustive search. Over the years, SETH has become a fundamental assumption relative to which many fine-grained complexity results are proved [Vassilevska Williams 2018]. For example, SETH implies current (quadratic) algorithms for orthogonal vectors or edit distance problems are optimal. A natural question is if SETH also implies that cubic algorithms for CFL reachability are optimal.

Our result shows that such a reduction would be very difficult to find. Carmosino et al. [2016] extended SETH to the nondeterministic strong exponential time hypothesis (NSETH), which states

that there is no algorithm for Boolean tautology better than exhaustive search, even with nondeterministic guessing. They show that both proving and refuting NSETH imply breakthroughs in computational complexity. Our subcubic certification result implies that any conditional lower bound for CFL reachability from SAT and SETH will show that NSETH does not hold. In particular, such a conditional lower bound will lead to a breakthrough in circuit lower bounds.

Thus, the importance of $O(n^\omega)$ verification of certificates is that it rules out fine-grained reductions from SAT to CFL reachability. This is a (conditional) negative answer to the question above, and possibly a step towards resolving the question of Chaudhuri [2008] whether the *all-pairs* version of CFL reachability can be reduced to *Boolean* matrix multiplication. In fact, even the yes/no version of the problem (with s and t given) is already very difficult. We discuss further implications in fine-grained complexity with other related work below.

A model checking problem closely related to CFL reachability is pushdown reachability [Bouajjani et al. 1997; Finkel et al. 1997]. Our results lead to **a subcubic certificate system for pushdown reachability** too, by extracting quadratic certificates from the standard saturation-based algorithm and the triplet construction for PDA to CFG conversion. Indeed, by exploiting fine-grained reductions between CFL reachability, pushdown reachability, the emptiness problem for pushdown automata, and the recognition problem for two-way nondeterministic pushdown automata (2NPDA), we show all these problems (as well as other related problems known in the literature) have subcubic certificate systems. Our constructions and reductions have several implications. First, succinct certificates for pushdown (non-)reachability checkable in subcubic time can have potential applications in *checking* proofs of programs [Necula 1997] and in "exports" of model checking such as certificate set analysis in trust management systems [Jha and Reps 2004]. Second, our reductions lead to a new insight beyond certification. We identify **a new hardest 2NPDA language**, that is, a fixed 2NPDA language L_0 such that for every 2NPDA language L there is a homomorphism h such that $w \in L$ iff $h(w) \in L_0$. A different hardest language was previously found by Rytter [1981] using language-theoretic techniques. However, our proof and reductions strengthen the link between 2NPDA language recognition and CFL reachability, pointing to the hardest instances of the latter.

Contributions. We make the following contributions.

(1) We propose certificate systems for positive and negative instances of CFL reachability, with certificates of size $O(n^2)$ which can be verified in time subcubic in the size of the graph. We prove these systems sound and complete (Section 3).
(2) We show that our certificate systems extend: to subcubic equivalent problems such as pushdown reachability, the emptiness problem for PDA, and 2NPDA language recognition (Subsections 5 and 6.1); as well as to all-pairs CFL reachability (Subsection 3.3). We also show they support verification in randomized quadratic time (with correct certificates never rejected), based on Freivalds' algorithm for verifying matrix multiplication [Freivalds 1979] (Subsection 3.2).
(3) We show that the influential strong exponential-time hypothesis (SETH) is unlikely to explain the cubic bottleneck for CFL reachability. We prove that a hypothetical fine-grained reduction from SAT for a conditional lower bound stronger than n^ω — roughly speaking, stronger lower bounds for program analysis — would lead to a breakthrough in circuit complexity (Section 4).
(4) Based on fine-grained reductions, we find a new hardest 2NPDA language (Subsection 6.2).

Related work. In a quest to classify the complexity of problems in P, *fine-grained reductions* interlink the asymptotic running time of algorithms for various problems. A fine-grained reduction shows that a faster algorithm for one problem automatically implies a faster algorithm for another

problem. Conversely, the existence of fine-grained reductions can be interpreted as *conditional lower bounds:* no faster algorithm exists, unless a state-of-the-art algorithm for a well-known problem is actually suboptimal. For example, a truly sub-quadratic algorithm for Orthogonal Vectors will lead to a $2^{(1-\varepsilon)n}$-time algorithm for SAT, breaking SETH [Vassilevska Williams 2018]. Similarly, the k-Clique conjecture states that no (randomized or deterministic) algorithm can detect a k-Clique on an n-vertex graph in time $O(n^{\frac{\omega k}{3}-\varepsilon})$ for $\varepsilon > 0$. Abboud et al. [2015] show a reduction from the k-Clique problem to CFL *recognition*, giving a conditional lower bound of order n^ω and matching Valiant's $\tilde{O}(n^\omega)$ upper bound for the problem [Valiant 1975]. This lower bound applies to CFL reachability as well. Chatterjee et al. [2018], using Lee's result [Lee 2002], reduce Boolean matrix multiplication to Dyck-k reachability (for growing k), showing that faster algorithms for the latter avoiding matrix multiplication would be a breakthrough. Chatterjee and Osang [2017] show a similar reduction to PDA emptiness.

More broadly, a range of problems in formal languages are now being approached with tools from modern algorithms and complexity [Fernau 2019]. Our work contributes to this ongoing effort. Backurs and Indyk [2016] and Bringmann et al. [2017] find SAT-based conditional lower bounds for regular expression matching problems. De Oliveira Oliveira and Wehar [2018] show reductions between triangle finding, 3SUM, and the non-emptiness of intersection of two or three DFA. Potechin and Shallit [2020] show a reduction from Orthogonal Vectors to the acceptance problem for (a subclass of) NFA and a reduction from triangle finding to (unary) NFA acceptance. Fernau and Krebs [2017] establish conditional lower bounds for a variety of automata-theoretic problems beyond P. Wehar and co-authors have shown that faster algorithms for various intersection non-emptiness problems have consequences for structural complexity classes [de Oliveira Oliveira and Wehar 2020; Swernofsky and Wehar 2015; Wehar 2014].

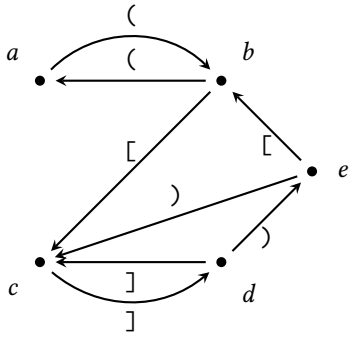
The idea of certification has a long history in computing and in algorithmics [McConnell et al. 2011]. In the context of pushdown reachability in verification, witness generation for yes-instances is supported by model checking tools such as Moped [Schwoon 2002]. In data management, the need to explain query results (data provenance) in graph databases has motivated Hellings [2020] to define certificates for yes-instances of CFL reachability, polynomial in the size of the input instance (and in the size of the grammar). These are, however, not subcubic. In comparison, our certificates are $O(n^2)$ in size, and we also provide certificates for no-instances. These two features are key for our result that SETH is unlikely to explain the cubic bottleneck for CFL reachability.

We discuss further related work in Section 6.

2 CONTEXT-FREE REACHABILITY AND DYCK-2 REACHABILITY

Let $\mathcal{L}$ be a fixed language. Given a directed graph $G = (V, E)$, an edge-labeling function $\lambda\colon E \to \Sigma$, and two vertices $s, t \in V$, the $\mathcal{L}$-reachability problem asks if there is a path from s to t (possibly repeating vertices and edges) such that the word formed by concatenating the labels along the path belongs to $\mathcal{L}$ [Yannakakis 1990]. When $\mathcal{L}$ is a fixed context-free language, the problem is called *CFL reachability*. CFL reachability plays a foundational role in several areas within computer science. To the best of our knowledge, it first appears in the work by Dolev, Even, and Karp [1982] as the combinatorial core in the security analysis of a cryptographic protocol. Yannakakis [1990] and Melski and Reps [2000] elucidate the role of this problem in the context of database theory and interprocedural program analysis, respectively, providing in particular a historical sketch.

In formal language theory, $\mathcal{L}$-reachability and CFL reachability can be seen as providing an algorithmic perspective on the classic definition of rational index of a language [Boasson et al. 1981; Pierre 1992]. These problems have also been studied under the name "regular realizability"



(a)

$$\overrightarrow{ac} \to \overrightarrow{ae} \cdot \overrightarrow{ec}$$
$$\overrightarrow{ae} \to (a,b) \cdot \overrightarrow{bd} \cdot (d,e)$$
$$\overrightarrow{ec} \to (e,b) \cdot \overrightarrow{bd} \cdot (d,c)$$
$$\overrightarrow{bd} \to (b,c) \cdot \overrightarrow{cc} \cdot (c,d)$$
$$\overrightarrow{cc} \to \varepsilon$$

(b)

$$\begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

(c)

Fig. 1. (a) Example: labelled graph (G, λ), main input to D_2Reach. (b) Yes-certificate: walk scheme for (G, λ, a, c). (c) No-certificate: separator for (G, λ, e, b). Vertices are ordered alphabetically: a is 1, b is 2, etc.

(see, e.g., [Vyalyi 2011; Vyalyi and Rubtsov 2015]). Lang [1994] explains a connection between $\mathcal{L}$-reachability and chart parsing.

For CFL reachability, without loss of generality, the fixed language can be assumed to be the *Dyck-2 language*. This is the language of balanced parentheses with two kinds of parenthesis symbols. Formally, it is the context free language over the alphabet $\{(,), [,]\}$ defined by the following context-free grammar:

$$S \to SS \mid (\ S\ ) \mid [\ S\ ] \mid \varepsilon$$

The *Dyck-2 reachability* problem, denoted D_2Reach, is the $\mathcal{L}$-reachability problem when $\mathcal{L}$ is the Dyck-2 language. Figure 1a shows an example input graph for this problem. The following claim (an algorithmic version of Exercise 6.2.7 in [Hopcroft et al. 2006]) shows restricting to D_2Reach is without loss of generality.

Claim 1. *Let (G, λ, s, t) be an instance of the CFL reachability problem. There is a linear-time reduction (in the bit-size of the input) to an instance (G', λ', s', t') of the Dyck-2 reachability problem.*

In addition to the decision (yes/no) version of the problem, we also study the *all-pairs* version, denoted All-Pairs-D_2Reach: on input (G, λ), the output of the All-Pairs-D_2Reach problem is a 0–1 matrix R of size $|V| \times |V|$ in which $R_{st} = 1$ if and only if (G, λ, s, t) is a yes-instance of D_2Reach.

We call an algorithm *truly subcubic* if it has (worst-case) running time $O(n^{3-\varepsilon})$ for some constant $\varepsilon > 0$, where n denotes the bit length of the input. Practical implementations use a summarization-based $O(|V|^3)$ algorithm [Reps et al. 1995]; note that $|V| \le n$. Using Rytter's trick [Rytter 1985], Chaudhuri [2008] shows that the $\mathcal{L}$-reachability problem is $O(|V|^3/\log|V|)$ for any fixed context-free language.[1] However, no truly subcubic algorithm is known for this problem. Although Boolean matrix multiplication reduces to this problem, which diminishes the hope for faster *combinatorial* algorithms (avoiding fast matrix multiplication), the best known conditional lower bound for the problem has order $|V|^\omega$, where ω is the matrix multiplication exponent. On the other hand, Dyck-1 reachability (the language of balanced parentheses with one kind of parentheses) can be solved in time $\tilde{O}(|V|^\omega)$ [Bradford 2018; Bringmann 2018; Mathiasen and Pavlogiannis 2021; Vyalyi 2019], matching best conditional lower bounds.

[1] If the CFL is not fixed, solving CFL reachability seems to require a product construction of the PDA with the graph, which incurs a blowup quadratic in the size of the graph. We do not know any algorithm better than $O(n^6/\log n)$ for this problem, where n is the bit size of the input.

3 CERTIFICATES FOR REACHABILITY AND NON-REACHABILITY

In this section we show that, while truly subcubic algorithms for Dyck-2 reachability are not known, solutions to Dyck-2 reachability have small and efficiently checkable certificates. In particular, regardless of whether a given instance x of $\mathsf{D}_2\mathsf{Reach}$ is a yes- or a no-instance, there exists a certificate y of truly subcubic size which witnesses this fact and can be checked in time truly subcubic in the size of x.

An instance $\boldsymbol{x} = (G, \lambda, s, t)$ of $\mathsf{D}_2\mathsf{Reach}$ is a yes-instance if there is a walk from s to t labeled with a string from Dyck-2, and a no-instance otherwise.

In the following definitions, $\|\boldsymbol{x}\|$ denotes the size of an instance $\boldsymbol{x}$. In the context of Dyck-2 reachability, this will be the number of vertices in the graph: if $G = (V, E)$, we write $\|\boldsymbol{x}\| = |V|$.

Definition 2. *We say that a decision problem $\mathcal{D}$ has* subcubic certificates *for yes-instances (respectively, no-instances) if, for some real number $\varepsilon > 0$, there is an algorithm M and a function $p\colon \mathbb{N} \to \mathbb{N}$, $p(n) = O(n^{3-\varepsilon})$, such that for every instance $\boldsymbol{x}$ of $\mathcal{D}$:*

- **(completeness)** *if $\boldsymbol{x}$ is a yes-instance (respectively, no-instance), then there is a string u of length at most $p(\|\boldsymbol{x}\|)$, called a* certificate, *such that M accepts $(\boldsymbol{x}, u)$ in time $p(\|\boldsymbol{x}\|)$, and*
- **(soundness)** *if $\boldsymbol{x}$ is a no-instance (respectively, yes-instance), then for every string u the algorithm M rejects $(\boldsymbol{x}, u)$ in time $p(\|\boldsymbol{x}\|)$.*

Note that the running time of M is subcubic in $|V|$, which is at most the bit size of the instance, and *not* in the size of the certificate. Also note that, in the soundness condition, if the length of the supplied string u exceeds $p(\|\boldsymbol{x}\|)$, we can assume that the algorithm M will reject the pair $(\boldsymbol{x}, u)$ without having to read the entire input. (Intuitively, only certificates of length $\leq p(\|\boldsymbol{x}\|)$ are expected.)

Theorem 3. $\mathsf{D}_2\mathsf{Reach}$ *has subcubic certificates for both yes- and no-instances.*

These certification schemes allow us to verify, given the additional certificate, whether an instance of $\mathsf{D}_2\mathsf{Reach}$ is a positive or a negative instance in sub-cubic time. Theorem 3 follows by combining Lemma 8 and Lemma 16 proved in Subsections 3.1 and 3.2 below.

Theorem 3 can be extended to the all-pairs setting, based on the following definition.

Definition 4. *Let $f\colon X \to Y$ be a computable function. We say that f* has subcubic certificates *if, for some real number $\varepsilon > 0$, there is an algorithm M and a function $p\colon \mathbb{N} \to \mathbb{N}$, $p(n) = O(n^{3-\varepsilon})$, such that for every instance $\boldsymbol{x} \in X$:*

- **(completeness)** *there is a string u of length at most $p(\|\boldsymbol{x}\|)$, called a* certificate, *such that M accepts $(\boldsymbol{x}, f(\boldsymbol{x}), u)$ in time $p(\|\boldsymbol{x}\|)$, and*
- **(soundness)** *for every $y \neq f(\boldsymbol{x})$ and for every string u of length at most $p(\|\boldsymbol{x}\|)$, the algorithm M rejects $(\boldsymbol{x}, y, u)$ in time $p(\|\boldsymbol{x}\|)$.*

Definition 4 is related to the properties required of so-called certifying algorithms, see McConnell et al. [2011, Section 5].

Theorem 5. All-Pairs-$\mathsf{D}_2\mathsf{Reach}$ *has subcubic certificates.*

Theorem 5 is proved in Subsection 3.3 below.

We will refer extensively to *walks* in labelled directed graphs. For a labelled directed graph $(V, E, \lambda\colon E \to \{(,), [,]\})$, a walk from $u \in V$ to $v \in V$ is a sequence of edges $\pi := e_0 \dots e_k$ from E, for $k \geq 0$, such that for each $i \in \{1, \dots, k\}$, edge e_{i-1} arrives at the same vertex that edge e_i departs from, and moreover e_0 departs from u and e_k arrives at v. This walk is *valid* if the word $\lambda(e_0) \dots \lambda(e_k)$ belongs to the Dyck-2 language. A subwalk of a walk $e_0 \dots e_k$ is a contiguous subsequence $e_i \dots e_j$ of edges, possibly empty.

3.1 Certificates for Yes-Instances: Compressed Walks

We describe our certificate system for yes-instances of D_2Reach. These certificates are witnesses for reachability. We fix an instance of D_2Reach: $G = (V, E)$ a directed graph, $\lambda\colon E \to \{(,), [,]\}$ an edge-labeling function, and $s, t \in V$ source and target vertices.

A first attempt is to provide a valid walk as a certificate (witness). However, it is well-known that the shortest valid walk can be exponential in the size of the input, namely it can be of length $\exp(\Theta(|V|^2/\log|V|))$, and this bound is tight [Pierre 1992]. (For an intuition, one can think of a pushdown automaton accepting only words of length exponential in its size and longer.)

One can construct a polynomial-sized certificate by providing the summaries constructed during CFL reachability [Reps et al. 1995]. However, the size of summaries can be $\Theta(|V|^3)$, and we need an additional idea for a *subcubic* certificate.

Our main observation to get subcubic certificates is that there is always some valid walk (including the shortest one in particular) that is well-compressible and has a small representation (of size $O(|V|^2)$); and it is efficient to check (in time $O(|V|^2)$) that such a compressed walk is indeed a valid walk. Moreover, for every no-instance, one cannot get any valid walks, compressed or otherwise.

The following definition "inlines" the concept of a straight-line program, which is an "acyclic" context-free grammar that generates one word only. Straight-line programs are at the core of general-purpose compression algorithms such as LZ77 (see, e.g., [Lohrey 2012]).

Definition 6. *For an instance of* D_2Reach*, denote by* $\overrightarrow{V^2}$ *a fresh copy of the set* V^2*, written as* $\overrightarrow{V^2} = \{\overrightarrow{uv} \mid (u, v) \in V^2\}$*. A* walk scheme *is a context-free grammar with the set of terminal symbols* E*, a set of nonterminal symbols* $\mathsf{NT} \subseteq \overrightarrow{V^2}$*, and the axiom* $\overrightarrow{st} \in \mathsf{NT}$*, where:*

- *for each nonterminal* $\overrightarrow{uv} \in \mathsf{NT}$ *there is exactly one production, which moreover has the form:*
  (a) $\overrightarrow{uv} \to \overrightarrow{uw}\ \overrightarrow{wv}$ *for some* $w \in V$*, or*
  (b) $\overrightarrow{uv} \to e\ \overrightarrow{xy}\ f$ *for some edges* $e = (u, x) \in E$ *and* $f = (y, v) \in E$ *with* $\lambda(e) \cdot \lambda(f) \in \{(), []\}$*, or*
  (c) $\overrightarrow{uu} \to \varepsilon$ *for some* $u \in V$*, and*
- *the directed graph with vertices* NT *and the following set of edges is acyclic:*

$$\left\{ (\overrightarrow{ab}, \overrightarrow{cd}) \mid \overrightarrow{cd} \text{ occurs on the right-hand side of the production of } \overrightarrow{ab} \right\}. \tag{1}$$

Example 7. For the input graph we saw previously in Fig. 1a, a walk scheme that witnesses reachability for the pair (a, c) is shown in Fig. 1b.

Lemma 8. *The following statements hold:*

- *Every walk scheme has size* $O(|V|^2)$ *and bit size* $O(|V|^2 \log|V|)$*.*
- *An instance of* D_2Reach *is a yes-instance if and only if there exists a walk scheme for it.*
- *There is a deterministic algorithm that runs in time* $O(|V|^2)$ *and decides if a given grammar is a walk scheme for a given instance of* D_2Reach*.*

For the proof of Lemma 8, we need the following auxiliary result.

Claim 9. *Let* $\mathcal{G}$ *be a context-free grammar with* $L(\mathcal{G}) \neq \emptyset$*. Suppose* $\mathcal{G}$ *contains more than one production with the same nonterminal on the left-hand side. Then by removing all of them but one we can obtain a grammar* $\mathcal{G}'$ *with* $L(\mathcal{G}') \neq \emptyset$*.*

We first show the proof of the lemma using Claim 9.

Proof of Lemma 8. The first assertion is straightforward. We split the rest of the proof into three parts.

Soundness. We first suppose that for a given instance of D_2Reach there exists a walk scheme, $\mathcal{W}$, and show that the instance must be a yes-instance. Consider the directed graph from the acyclicity condition in the definition of walk schemes, denote it D. We will consider all vertices of D, i.e., nonterminals from NT, in any reversed topological ordering. In other words, whenever $\overrightarrow{cd}$ occurs on the right-hand side of the production of $\overrightarrow{ab}$, we will consider $\overrightarrow{cd}$ before $\overrightarrow{ab}$. We will show by induction that, for every $\overrightarrow{uv} \in \mathsf{NT}$, the (one) word generated by $\overrightarrow{uv}$ is a valid walk from u to v. (Recall that a walk is valid if it is labelled by a Dyck-2 word.) Indeed, it suffices to consider the three types of productions:

(1) for a production of the form $\overrightarrow{uv} \to \overrightarrow{uw}\,\overrightarrow{wv}$, we know from the inductive hypothesis that $\overrightarrow{uw}$ generates a valid walk from u to w, and $\overrightarrow{wv}$ a valid walk from w to v, so their concatenation is a valid walk from u to v;
(2) for a production of the form $\overrightarrow{uv} \to e\,\overrightarrow{xy}\,f$ with edges $e = (u, x) \in E$ and $f = (y, v) \in E$, we know from the inductive hypothesis that $\overrightarrow{xy}$ generates a valid walk from x to y, and since $\lambda(e) \cdot \lambda(f) \in \{(), []\}$, the result of the concatenation is a valid walk from u to v;
(3) finally, productions of the form $\overrightarrow{uu} \to \varepsilon$ correspond to trivial valid walks (containing no edges) and represent the induction base.

As the axiom of the grammar $\mathcal{W}$ is $\overrightarrow{st}$, we conclude that there is a valid walk from s to t, which means that the instance of D_2Reach we consider is a yes-instance.

Completeness. In the converse direction, let us prove that that every yes-instance of D_2Reach has a walk scheme. Consider such an instance, (G, λ, s, t), and consider a walk from s to t, call it π. We construct a walk scheme in several steps.

First consider a context-free grammar $\mathcal{G}$ with the set of terminal symbols E, set of nonterminal symbols $\overrightarrow{V^2}$, and axiom $\overrightarrow{st}$. The set of productions is determined as follows. For each nonterminal $\overrightarrow{uv} \in \overrightarrow{V^2}$, we include all productions of the form:

- $\overrightarrow{uv} \to \overrightarrow{uw}\,\overrightarrow{wv}$ for all $w \in V$;
- $\overrightarrow{uv} \to e\,\overrightarrow{xy}\,f$ where $e = (u, x) \in E$ and $f = (y, v) \in E$ such that $\lambda(e) \cdot \lambda(f) \in \{(), []\}$;
- $\overrightarrow{uu} \to \varepsilon$ for all $u \in V$.

Induction on the structure of π shows that $\pi \in L(\mathcal{G})$, so $L(\mathcal{G}) \neq \varnothing$.

We can now prune the set of productions of the grammar $\mathcal{G}$ using Claim 9, as well as apply standard procedures of removing useless (non-productive or unreachable) nonterminals in context-free grammars (see, e.g., [Hopcroft et al. 2006, Section 7.1]). We perform these steps until all three have no effect on the grammar. The resulting grammar $\mathcal{W}$ satisfies all conditions in the definition of walk schemes, except possibly the acyclicity condition. We claim that $\mathcal{W}$ must satisfy that condition too. Indeed, the transformations applied so far ensure that $L(\mathcal{W}) \neq \emptyset$. Let $\mathsf{NT} \subseteq \overrightarrow{V^2}$ be the set of nonterminals of $\mathcal{W}$. Assume for the sake of contradiction that the directed graph with vertices NT and edges (1) contains a directed cycle. Let $\overrightarrow{ab} \in \mathsf{NT}$ be a vertex on this cycle. Since all nonterminals of $\mathcal{W}$ are reachable and productive, there exists a valid parse tree with respect to $\mathcal{W}$ that contains a node labelled by $\overrightarrow{ab}$. By definition of the graph, and since every nonterminal in $\mathcal{W}$ has exactly one production, this node has a descendant labelled with $\overrightarrow{ab}$. By the same reasoning, this descendant also has a descendant labelled with $\overrightarrow{ab}$, etc., which cannot be the case as the tree is finite. This contradiction means that the graph must be acyclic, so $\mathcal{W}$ is in fact a walk scheme.

Verification algorithm. The condition $\mathsf{NT} \subseteq \overrightarrow{V^2}$ and the choice of the axiom can be checked in time $O(|V|^2)$. The fact that there is exactly one production per nonterminal can be checked under the same time constraints; and so can the form of these productions and compatibility with the instance of D_2Reach. Finally, depth-first search–based topological sort procedure can be used to detect the existence of directed cycles; it runs in time linear in the number of edges, which is at most $|V|^2$. □

Remark 10. There is nothing special about Dyck-2 in the construction, and a similar certificate can be constructed for any fixed CFG.

We now finish the proof by proving Claim 9.

Proof of Claim 9. Let N be the nonterminal from the statement of the lemma. If N is not productive, i.e., cannot derive any word, then *all* of its productions can be removed without any effect on $L(\mathcal{G})$. This is simply because N cannot appear in any successful derivation. We will therefore assume that N is productive.

Consider the parse tree of any successful derivation from N. We can find in this parse tree a vertex labelled with N such that none of its descendants is labelled with N. The subtree T_N rooted at this vertex corresponds to a derivation that applies some production $P\colon N \to \xi$ first and never uses N again.

By removing all other productions with left-hand side N from $\mathcal{G}$, we obtain a new grammar $\mathcal{G}'$. Let us show that $L(\mathcal{G}') \neq \emptyset$. Indeed, let S be the axiom of $\mathcal{G}$. As S is productive, $u \in L(\mathcal{G})$ for some word u. Consider any parse tree T of u in $\mathcal{G}$. If T contains no occurrence of N, then it is already a valid parse tree with respect to $\mathcal{G}'$, and we are done. Otherwise, for every node labelled with N in T from which the shortest path to the root has no other occurrence of N, we replace the corresponding subtree by T_N. This results in a valid parse tree with respect to $\mathcal{G}'$, because T_N has one occurrence of N only, namely at its root, where the production applied is P. The new parse tree is a derivation of some word in $L(\mathcal{G}')$, which concludes the proof. □

We already mentioned a link to compressed words above. Our proof of Lemma 8 finds a context-free grammar that generates exactly one word and, if brought to Chomsky normal form, has $O(|V|^2)$ nonterminals. Importantly, while it is in general a PSPACE-complete problem to decide whether such a compressed word is accepted by a pushdown automaton (see, e.g., the survey [Lohrey 2012, section 9.4] and references therein), our grammar has special structure, leading to an efficient verification algorithm.

3.2 Certificates for No-Instances: Inductive Invariants

Fix an instance of D_2Reach. For ease of notation, we will assume that $V = \{1, \ldots, |V|\}$. A certificate for no-instances will be a *separator*, as defined next. Such a certificate is essentially an inductive invariant, certifying non-reachability.

Let $A_(, A_[, A_), A_]$ be four 0–1 matrices of size $|V| \times |V|$ that are adjacency matrices for the graph G restricted to sets of edges with labels (, [,),], respectively.

Let I denote the $|V| \times |V|$ identity matrix. We write $A \leq B$ for matrices $A = (a_{ij})$ and $B = (b_{ij})$ of the same size whenever $a_{ij} \leq b_{ij}$ for all i, j.

Definition 11. *A* separator *for an instance of* D_2Reach *is a* 0–1 *matrix* M *of size* $|V| \times |V|$ *such that the following five conditions are satisfied:*

$$\begin{aligned} I &\leq M, & A_(\cdot M \cdot A_) &\leq M, \\ M \cdot M &\leq M, & A_[\cdot M \cdot A_] &\leq M, \quad \text{and } M_{s,t} = 0, \end{aligned} \tag{2}$$

where addition and multiplication are Boolean, and s and t are the source and target vertex in the instance of $\mathsf{D}_2\mathsf{Reach}$.

Example 12. For the input graph in Fig. 1a, a separator for the pair (e, b) is shown in Fig. 1c. Denoting this matrix by $M = (m_{ij})$, we easily see that $I \le M$, $M \cdot M \le M$, and $m_{52} = 0$. Instead of performing the other required multiplications, we observe that since both (-edges in the graph depart from vertices a and b, we can conclude that $A_(\cdot M \cdot A_) \le M$. Likewise, both]-edges arrive at vertices c and d, and hence $A_[\cdot M \cdot A_] \le M$.

Lemma 13. *The following statements hold:*

- *Every separator has bit size $O(|V|^2)$.*
- *An instance of* $\mathsf{D}_2\mathsf{Reach}$ *is a no-instance if and only if there exists a separator for it.*
- *There is a deterministic algorithm that runs in time $O(|V|^\omega)$ and decides if a given matrix is a separator for a given instance of* $\mathsf{D}_2\mathsf{Reach}$.

Proof. We split the proof into four parts, with the first assertion being an easy observation.

Completeness. First consider a no-instance of $\mathsf{D}_2\mathsf{Reach}$. Take the matrix $M = (m_{ij})$, where each m_{ij} is 1 if there is a valid walk from vertex i to vertex j. It is clear that $m_{st} = 0$, because the instance is a no-instance. We now show that the four matrix constraints in Equation (2) are satisfied:

- $I \le M$ because for each vertex i the empty walk from i to i is valid;
- $M \cdot M \le M$ because the concatenation of two valid walks is a valid walk;
- $A_(\cdot M \cdot A_) \le M$ and $A_[\cdot M \cdot A_] \le M$ because every walk $e \cdot \pi \cdot e'$ is valid whenever π is valid and e and e' are labelled by a matching pair of parentheses, either (,) or [,].

This shows that there is a separator for each no-instance.

Remark 14. Our completeness proof gives a separator in which the (i, j)-entry of M is 1 iff (G, λ, i, j) is a yes-instance of $\mathsf{D}_2\mathsf{Reach}$. This fact will be significant in the sequel.

Soundness. In the converse direction, consider an arbitrary instance of $\mathsf{D}_2\mathsf{Reach}$. We show that for every valid walk π from a vertex u to a vertex v in the graph, all separators must satisfy the condition $m_{uv} = 1$ where $M = (m_{ij})$. (It then follows that yes-instances have no separators.) We use induction on the label of walk π, which is simply the concatenation of individual edge labels:

- The base case is the empty label, ε. The walk π must then be the empty walk, from some vertex u to itself. We recall that $I \le M$ for every separator; so indeed m_{ii} must be set to 1 for all vertices i, and for the chosen vertex $i = u$ in particular.
- If the walk π is labelled by $\alpha \cdot \beta$, where both α and β are nonempty Dyck-2 words, then there exists a vertex w such that $\pi = \pi' \cdot \pi''$ and π' and π'' are valid walks from u to w and from w to v, respectively. By the inductive hypothesis, $m_{u,w} = m_{w,v} = 1$. Since $M \cdot M \le M$, we conclude that $m_{u,v} = 1$ in this case as well.
- Finally, suppose the label of the walk π is (α), for some Dyck-2 word α. (The case $[\alpha]$ is analogous.) Then $\pi = e \cdot \pi' \cdot f$, where e and f are individual edges, say from u to u' and from v' to v (for some $u', v' \in V$), and π' is a valid walk from u' to v'. The edges $e = (u, u')$ and $f = (v', v)$ have labels (and), respectively. By the inductive hypothesis, $m_{u'v'} = 1$. We now recall that $A_(\cdot M \cdot A_) \le M$. On the left-hand side, the matrix product has a positive entry in position uv, because $\big(A_(\big)_{u,u'} = \big(A_)\big)_{v',v} = 1$ by the definition of $A_($ and $A_)$. Therefore $m_{uv} = 1$.

This concludes the proof of the second assertion of the lemma.

Verification algorithm. The algorithm from the third assertion of the lemma verifies all conditions in the definition of separator directly. This means, in particular, performing five Boolean matrix multiplications (worst-case time $O(|V|^\omega)$, computing over the integers first), four inequalities between matrices (worst-case time $O(|V|^2)$), and a single equality constraint on one of the entries (constant time). □

The deterministic algorithm from Lemma 13 reduces the verification of separators to 5 Boolean matrix multiplications. While this result has complexity-theoretic consequences (see Section 4 below), it may appear unsatisfactory, as many theoretical algorithms for fast matrix multiplication are impractical. This leads to the idea of verification with a randomized, faster algorithm.

For a nonnegative integer matrix N, denote by $\mathrm{bool}(N)$ the matrix obtained from N by replacing every nonzero element by 1.

Definition 15. *A* composite separator *for an instance of* $\mathsf{D}_2\mathsf{Reach}$ *is a sextuple of* $|V| \times |V|$ *matrices,* $(M_S, M_{SS}, M_{(S}, M_{[S}, M_{(S)}, M_{[S]})$*, where all entries belong to* $\{0, 1, \ldots, |V|^2\}$*, and moreover all entries of* M_S *belong to* $\{0, 1\}$*, and such that the following ten conditions are satisfied:*

$$\begin{aligned} I &\le M_S, & A_(\cdot M_S &= M_{(S}, & A_[\cdot M_S &= M_{[S}, \\ M_S \cdot M_S &= M_{SS}, & M_{(S} \cdot A_) &= M_{(S)}, & M_{[S} \cdot A_] &= M_{[S]}, \\ \mathrm{bool}(M_{SS}) &\le M_S, & \mathrm{bool}(M_{(S)}) &\le M_S, & \mathrm{bool}(M_{[S]}) &\le M_S, \quad \textit{and } (M_S)_{s,t} = 0, \end{aligned} \tag{3}$$

where s *and* t *are the source and target vertex in the instance of* $\mathsf{D}_2\mathsf{Reach}$.

We emphasize that addition and multiplication in Equation (3) are integer (unlike Equation (2)).

Lemma 16. *The following statements hold:*

- *Every composite separator has* $O(|V|^2)$ *entries and bit size* $O(|V|^2 \log |V|)$.
- *An instance of* $\mathsf{D}_2\mathsf{Reach}$ *is a no-instance if and only if there exists a composite separator for it.*
- *There is a deterministic algorithm that runs in time* $O(|V|^\omega)$ *and decides if a given sextuple of* $|V| \times |V|$ *matrices is a composite separator for a given instance of* $\mathsf{D}_2\mathsf{Reach}$.
- *There is a randomized algorithm that runs in time* $O(|V|^2)$ *and decides if a given sextuple of* $|V| \times |V|$ *matrices is a composite separator for a given instance of* $\mathsf{D}_2\mathsf{Reach}$. *In the case it is, the algorithm never errs; otherwise the algorithm flags an issue with probability* ≥ 0.5.

Proof. The first three assertions are shown similarly to the proof of Lemma 13. For example, in the completeness proof we choose M_S as M from Lemma 13 and show that picking the other matrices $M_{SS}, M_{(S}, M_{[S}, M_{(S)}, M_{[S]}$ so that all the five matrix equalities among the constraints (3) are satisfied leads to the satisfaction of the remaining (four) inequality constraints.

Randomized verification algorithm. The algorithm from the final assertion of the lemma, instead of *computing* matrix products, runs Freivalds' algorithm for *verifying* them [Freivalds 1979].

Recall that Freivalds' algorithm for verifying $A \cdot B = C$ for some $n \times n$ matrices A, B, and C proceeds by picking a 0–1 vector $u \in \{0, 1\}^n$ uniformly at random and checking if $A \cdot (Bu) = Cu$. The algorithm runs in $O(n^2)$ time and has error probability $1/2$. The properties of the algorithm are transferred directly to give a $O(|V|^2)$ bound. Since we have five products in Equation (3) to check, we reduce the error probability in an individual check to $1/16$ by running it 4 times, so that the overall error probability is at most $5/16 \le 1/2$. □

Remark 17. Once again, there is nothing special about the Dyck-2 language in our certificate systems. One can readily see that the conditions we impose on (e.g., composite) separators correspond to the following context-free grammar for the Dyck-2 language:

$$S \to SS \mid P) \mid Q] \mid \varepsilon \qquad P \to (S \qquad Q \to [S \ .$$

Replacing this grammar with a different one, we obtain a certificate system (for no-instances) for the CFL reachability problem where the fixed CFL is represented by any fixed CFG.

Remark 18. Could the speedup offered by Freivalds' algorithm be matched by a deterministic procedure? In a model of computation with unit-cost integer arithmetic, integer matrix multiplication can be verified in deterministic time $O(n^2)$ [Kimbrel and Sinha 1993; Korec and Wiedermann 2014]. For RAM with $O(\log n)$-bit arithmetic operations, derandomization of Freivalds' algorithm is an open problem even in the nondeterministic setting. However, if the number of errors in the product is guaranteed to be $O(n^{2-\varepsilon})$, then a deterministic $O(n^{3-\varepsilon})$-time algorithm is known [Künnemann 2018]. Over an appropriate class of algebraic structures, the *verification* of matrix products has been linked with their *computation* by Vassilevska Williams and Williams [2018].

3.3 Certificates for All-Pairs CFL Reachability

An appropriate combination of certificates for the yes/no version of Dyck-2 reachability can be used as a certificate for the all-pairs setting. The number of required yes-certificates (walk schemes) can, however, be large. For a quadratic upper bound on the total size, an additional argument is needed.

Definition 19. *A* combined witness *for an instance* (G, λ) *of* All-Pairs-D_2Reach *is a pair* $(\mathcal{W}, \mathcal{S})$ *where:*

- $\mathcal{W}$ *is a walk scheme as in Definition 6, except no axiom is distinguished;*
- $\mathcal{S}$ *is a separator as in Definition 11;*
- *for all pairs* $(s, t) \in V^2$*, if the* (s, t)*-entry of* M *in the separator* $\mathcal{S}$ *is* 1*, then the walk scheme* $\mathcal{W}$ *has a nonterminal* $\overrightarrow{st}$*.*

Example 20. For our running example from Fig. 1a, a combined witness is shown in Fig. 2.

Lemma 21. *The following statements hold:*

- *Every combined witness has size* $O(|V|^2)$ *and can be encoded with* $O(|V|^2 \log |V|)$ *bits.*
- *Every instance of* All-Pairs-D_2Reach *has a combined witness.*
- *There is a deterministic algorithm that runs in time* $O(|V|^\omega)$ *and checks if its input is a valid combined witness for a given instance of* All-Pairs-D_2Reach*.*

Proof. The first and last assertions are consequences of Lemmas 8 and 13. The proof of the second assertion relies on the properties of certification schemes for yes- and no-instances of D_2Reach. Indeed, let us first invoke the completeness of no-certificates (Lemma 13). Given an instance of All-Pairs-D_2Reach, consider the set Y of all pairs (s, t) such that (G, λ, s, t) is a yes-instance of D_2Reach. There is an appropriate separator $\mathcal{W}$ in which the (s, t)-entry of M is 1 iff $(s, t) \in Y$. This is by Remark 14 (page 10). A combination of walk schemes for all these yes-instances could be used for certification; however, their number is only known to be bounded by $|V|^2$, so we need to find a single walk scheme that captures all relevant (s, t).

We will apply the same argument as in the proof of Lemma 8: construct a grammar $\mathcal{G}$ and then prune it accordingly. The only change is that, initially, we will introduce an auxiliary nonterminal in the grammar, denoted X. This nonterminal X will be the axiom of the grammar, and it will have a single production $X \to \overrightarrow{s_1t_1}\overrightarrow{s_2t_2} \dots \overrightarrow{s_kt_k}$ where $\{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\} = Y$. This ensures that all nonterminals on the right-hand side of this production are reachable; they are all productive for the same reason as in Lemma 8. We transform the grammar in exactly the same way as previously, observing that X and its defining production will stay unchanged. In particular, all k nonterminals $\overrightarrow{s_it_i}$ will be retained. As X never occurs on the right-hand side of productions, the grammar will

$$\overrightarrow{be} \to (b,a)\cdot\overrightarrow{ad}\cdot(d,e) \qquad \overrightarrow{ac} \to \overrightarrow{ae}\cdot\overrightarrow{ec}$$
$$\overrightarrow{ad} \to \overrightarrow{ae}\cdot\overrightarrow{ed} \qquad \overrightarrow{ae} \to (a,b)\cdot\overrightarrow{bd}\cdot(d,e)$$
$$\overrightarrow{ed} \to (e,b)\cdot\overrightarrow{bc}\cdot(c,d) \qquad \overrightarrow{ec} \to (e,b)\cdot\overrightarrow{bd}\cdot(d,c)$$
$$\overrightarrow{bc} \to (b,a)\cdot\overrightarrow{ae}\cdot(e,c) \qquad \overrightarrow{bd} \to (b,c)\cdot\overrightarrow{cc}\cdot(c,d)$$

$$M = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

$$A_(\cdot M \cdot A_) = \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \leq M$$

$$A_[\cdot M \cdot A_] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \end{pmatrix} \leq M$$

$$M \cdot M = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \leq M$$

Fig. 2. Above: combined witness for the graph in Fig. 1a, consisting of walk scheme (shown without ε-productions, and with all other productions already sorted) and separator. Below: verification of matrix inequalities (except $I \leq M$) for the separator.

again be acyclic. Thus, removing X at the end of the process will lead to an axiom-less walk scheme; and all the conditions of Definition 19 will be satisfied. □

Theorem 5 follows from Lemma 21.

Replacing separators with composite separators (Definition 15) in our construction of combined certificates (Definition 19) enables a randomized checker that runs in time $O(|V|^2)$, based on Lemma 16. The checker never errs on correct certificates and rejects any incorrect one with probability ≥ 0.5.

4 COMPLEXITY IMPLICATIONS

Fine-grained complexity research shows that even small improvements in (the exponent of) the running time of many algorithmic problems, such as orthogonal vectors or edit distance, would automatically give faster algorithms for Boolean satisfiability (SAT) [Vassilevska Williams 2018]. Would improvements over Chaudhuri's $O(n^3/\log n)$-time algorithm for D_2Reach also have consequences for SAT? Here we show that subcubic certificates give an answer to this question.

Complexity-theoretic summary of Section 3. Leaving out sharper bounds on certificate size and polylog(n) factors (required in the Turing model), Lemmas 8 and 16 imply:

Theorem 22. $\mathsf{D_2Reach} \in \mathsf{NTIME}(n^2) \cap \text{co-}\mathsf{NTIME}(n^\omega) \cap \text{co-}\mathsf{MATIME}_1(n^2)$.

For this summary, we recall that a language $L \in \mathsf{NTIME}(t)$ iff there is a nondeterministic Turing machine M that runs in time $O(t(|x|))$ for input x and accepts L and $L \in \text{co-}\mathsf{NTIME}(t)$ iff its complement is in $\mathsf{NTIME}(t)$. Also, (cf. [Tell 2019]) $L \in \mathsf{MATIME}_1(t)$ (Merlin-Arthur time, introduced by Babai [1985]) iff there exists a deterministic machine M that takes inputs x, y, z where $|y| = |z| = O(t(|x|))$, runs in time $O(t(|x|))$, and such that for every x,

$$x \in L \implies \exists y. \Pr_z[M(x,y,z) \text{ accepts}] = 1, \qquad x \notin L \implies \forall y. \Pr_z[M(x,y,z) \text{ accepts}] \le 1/2,$$

where the probability is with respect to the uniform distribution of z in $\{0,1\}^{t(|x|)}$. Finally, $\text{co-}\mathsf{MATIME}_1(t)$ is the class of *complements* of languages in $\mathsf{MATIME}_1(t)$.

In a nutshell, Merlin can perform arbitrary computation (think nondeterministic guessing) but Arthur does not trust him and verifies the results in order to be convinced of their validity. Importantly, the certification itself involves no chance taking (i.e., the certificate system is sound and complete), and it is only the verification procedure that may rely on statistical evidence to convince Arthur.

Theorem 22 is a restatement of Theorem 3. Membership in $\mathsf{NTIME}(n^2)$ follows because the nondeterministic machine guesses a $O(n^2)$ certificate for a yes-instance and checks it in $O(n^2)$ time. Membership in $\text{co-}\mathsf{NTIME}(n^\omega)$ follows because a nondeterministic machine for the complement of $\mathsf{D_2Reach}$ guesses a $O(n^2)$ certificate for a no-instance and verifies it in $O(n^\omega)$ time. Finally, membership in $\text{co-}\mathsf{MATIME}_1(n^2)$ follows because Merlin provides a $O(n^2)$ certificate for a no-instance and Arthur verifies the certificate using a randomized $O(n^2)$ algorithm. (In the above, we ignore polylogarithmic factors.)

Fine-grained complexity of $\mathsf{D_2Reach}$. In fine-grained complexity, perhaps the most influential hypothesis, and the ultimate source of many lower bounds, is the *strong exponential-time hypothesis* (SETH) [Impagliazzo and Paturi 2001], stating (roughly) that there is no algorithm for SAT (or, equivalently, for TAUT) better than exhaustive enumeration. The *nondeterministic strong exponential-time hypothesis* (NSETH) [Carmosino et al. 2016] extends it further.

*Hypothesis 23 (*SETH*).* For every $\varepsilon > 0$, there exists a k so that k-SAT is not in $\mathsf{DTIME}[2^{n(1-\varepsilon)}]$, where k-SAT is the language of all satisfiable Boolean formulas in k-CNF.

*Hypothesis 24 (*NSETH*).* For every $\varepsilon > 0$, there exists a k so that k-TAUT is not in $\mathsf{NTIME}[2^{n(1-\varepsilon)}]$, where k-TAUT is the language of all Boolean tautologies in k-DNF.

In both hypotheses, n is the number of variables. It is unknown whether SETH and NSETH are true. NSETH implies SETH, and SETH implies $\mathsf{P} \neq \mathsf{NP}$. Carmosino, Gao, Impagliazzo, Mihajlin, Paturi, and Schneider [2016] explore consequences of NSETH and show that both proving and refuting it would lead to interesting consequences. In particular, NSETH implies the *absence* of fine-grained reductions from SAT to a number of problems and ¬NSETH implies circuit lower bounds.

Intuitively, a fine-grained reduction from $(L, t(n))$ to $(\mathsf{D_2Reach}, n^c)$ means that, for every $\varepsilon > 0$, an $O(n^{c-\varepsilon})$-time algorithm for $\mathsf{D_2Reach}$ implies a $O(t(n)^{1-\delta})$ algorithm for problem L for some $\delta = \delta(\varepsilon) > 0$. This is not unlike usual Turing reductions (allowing multiple queries), tracking the precise exponents in the running time bounds.

We show that, because of our subcubic certificate systems (Section 3), there exists no fine-grained reduction from SAT (as well as from any SETH-hard problem) to $\mathsf{D_2Reach}$ that would imply hardness beyond n^ω, unless NSETH fails. Since disproving NSETH would mean breakthroughs in proof complexity as well as in circuit complexity, a fine-grained reduction from SAT to $\mathsf{D_2Reach}$, if one exists, will be difficult to find.

We first state the result informally and provide the intuition behind the proof. Then we provide the formal details.

Theorem 25 (Informal Version). *Unless* NSETH *fails, there is no fine-grained reduction from* $(\mathsf{SAT}, 2^n)$ *to* $(\mathsf{D_2Reach}, n^{\omega+\gamma})$ *for any* $\gamma > 0$.

Intuition of the proof. This sketch expands the idea of Carmosino et al. [2016]. We will argue that if there is a reduction from SAT that gives a conditional lower bound for $\mathsf{D_2Reach}$ stronger than n^ω, say n^α for $\alpha > \omega$, then there is a nondeterministic algorithm for TAUT that runs in time $O(2^{n(1-\varepsilon)})$ on an input CNF with n variables.

The idea of the proof is to combine the hypothetical reduction with a fast guess-and-check procedure for $\mathsf{D_2Reach}$. A fine-grained reduction can be thought of as a computer program that makes one or more "black-box" calls to $\mathsf{D_2Reach}$. In the case at hand, this program implements a deterministic algorithm for SAT. Given an instance of TAUT, we negate the input and run the algorithm for SAT. We will run a slightly modified program, as follows:

- Firstly, instead of running standard deterministic algorithms for $\mathsf{D_2Reach}$ for each call to $\mathsf{D_2Reach}$ in the reduction, we will substitute a *nondeterministic* guess-and-check procedure. This procedure relies on nondeterminism and subcubic certification systems for $\mathsf{D_2Reach}$ (Theorem 3). It guesses if the current instance is a positive or a negative instance, and a corresponding certificate (a walk scheme or separator). It then runs a subcubic verification algorithm, confirming that the instance is a yes- or no-instance, respectively. Accordingly, it then returns true or false. The upshot is that the answer to every call to $\mathsf{D_2Reach}$ can be certified with a short (subcubic) witness.
- Secondly, whenever the original reduction returns a (Boolean) answer, this answer is flipped: instead of "true" we return "false", and instead of "false" we return "true". This is because we want to supply an answer to the TAUT input.

Since some of the calls to $\mathsf{D_2Reach}$ have positive answers and some negative answers, we require both certification schemes for this algorithm to work.

In computational complexity terms, this composition of the two programs gives us a nondeterministic algorithm that solves TAUT. One can think of this as follows: given a SAT/TAUT instance, consider the execution of the program as described above. The protocol of this execution, containing certificates for each call to $\mathsf{D_2Reach}$, can be used to certify that the given SAT/TAUT instance has whichever answer it does. This is due to the correctness of the original reduction and the properties of the certification schemes (Section 3).

But we can now recall that, for the original reduction, the running times must "compose well": that is, if we substitute any $O(n^{\alpha-\varepsilon})$-time algorithm for $\mathsf{D_2Reach}$, then the reduction will yield an $O(2^{n(1-\delta)})$-time algorithm for SAT, for some $\delta = \delta(\varepsilon) > 0$. (Notice the α in the first exponent; this parameter comes with the reduction, which gives a conditional lower bound of n^α.) We effectively substituted *nondeterministic* and *co-nondeterministic* algorithms for $\mathsf{D_2Reach}$, both running in matrix multiplication time. Since $\alpha > \omega$ where ω is the matrix multiplication exponent, we can take any $\varepsilon \in (0; \alpha - \omega)$. Instead of running time, let us now look at the length of the entire execution protocol: this should give us the bound $O(2^{n(1-\delta)})$ with the same $\delta = \delta(\varepsilon)$. As argued above, this protocol witnesses the answer to the given SAT/TAUT instance. In other words, we have a nondeterministic algorithm for TAUT with this running time, breaking NSETH.

Fine-grained reductions and proof of Theorem 25: formal details. To formally state Theorem 25, we need the following definitions (see [Carmosino et al. 2016; Vassilevska Williams 2018]). Let L_1 and L_2 be languages, and let T_1 and T_2 be time bounds, i.e., functions $\mathbb{N} \to \mathbb{N}$. We interpret pairs (L_i, T_i) as problems with their conjectured (or presumed) complexities. We say that (L_1, T_1)

fine-grained reduces to (L_2, T_2), written $(L_1, T_1) \leq_{\mathsf{FGR}} (L_2, T_2)$, if (a) for all $\varepsilon > 0$, there is $\delta > 0$ and a deterministic Turing reduction M^{L_2} from L_1 to L_2 such that the running time of M is at most $T_1^{1-\delta}$ and such that (b) if $Q(M, x)$ denotes the set of queries made by M to the L_2 oracle on an input x of length n, then the query lengths obey the time bound

$$\sum_{q \in Q(M,x)} (T_2(|q|))^{1-\varepsilon} \leq (T_1(n))^{1-\delta}.$$

Intuitively, a fine-grained reduction from (L_1, T_1) to (L_2, T_2) enables algorithmic savings for L_2 to be transferred to L_1. That is, if L_2 can be solved in time $T_2^{1-\varepsilon}$, then L_1 can be solved in time $T_1^{1-\delta}$. A language L with time complexity T is SETH-*hard* if $(\mathsf{SAT}, 2^n) \leq_{\mathsf{FGR}} (L, T)$.

Theorem 26 ([Carmosino et al. 2016, Theorem 2 and Corollary 2]). *Suppose* NSETH *holds and a problem* L *belongs to* $\mathsf{NTIME}[T] \cap \text{co-}\mathsf{NTIME}[T]$. *Then* $(\mathsf{SAT}, 2^n) \not\leq_{\mathsf{FGR}} (L, T^{1+\gamma})$ *for any* $\gamma > 0$. *Also, for any* L' *that is* SETH-*hard with time* T', *and any* $\gamma > 0$, *we have* $(L', T') \not\leq_{\mathsf{FGR}} (L, T^{1+\gamma})$.

We are now ready to formulate Theorem 25 rigorously.

Theorem 27 (Theorem 25 restated). *Unless* NSETH *fails,* $(\mathsf{SAT}, 2^n) \not\leq_{\mathsf{FGR}} (\mathsf{D_2Reach}, n^{\omega+\gamma})$ *for any* $\gamma > 0$.

It remains to observe that Theorem 25 follows from Theorem 3 and 26.

5 CERTIFICATES FOR PUSHDOWN NON-REACHABILITY

While CFL reachability is a central problem in program analysis, an analogous problem in model checking is *pushdown reachability* [Bouajjani et al. 2000, 1997; Finkel et al. 1997; Schwoon 2002], formalized as follows.

We are given a pushdown automaton (PDA) $\mathcal{P} = (Q, \Gamma, \Delta)$, where Q is a finite set of states, Γ is a finite alphabet of stack symbols, and $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma^{\leq 2})$ is a set of transitions, and an initial configuration $(q_0, \gamma_0) \in Q \times \Gamma$. We are additionally given a regular set of configurations R specified by a $\mathcal{P}$-*automaton*: this is a usual, ε-free nondeterministic finite automaton (NFA) over the alphabet Γ in which the set of control states is $S \supseteq Q$ and the transition relation is $\delta \subseteq S \times \Gamma \times S$. A set of final states, $F \subseteq S$, is usually taken to be disjoint from Q. Such a $\mathcal{P}$-automaton is said to accept a configuration $(q, w) \in Q \times \Gamma^*$ of the PDA $\mathcal{P}$ iff there is a walk from control state q to some $\bar{q} \in F$ labelled by the word w; in other words, if w is accepted by this NFA when started from q as initial state. We ask if the PDA $\mathcal{P}$ has a run from (q_0, γ_0) to some configuration from R.

We adapt our certificate system to pushdown reachability. For yes-certificates of size $O(|\Gamma||S|^2)$, we can convert the PDA to an equivalent CFG using the standard triplet construction (see, e.g., [Hopcroft et al. 2006, Chapter 6]) on the "sum" of $\mathcal{P}$ and the $\mathcal{P}$-automaton and repeat the second half of the completeness argument from Subsection 3.1. Explicitly, a certificate is a "sub-grammar" of this CFG that is a straight-line program.

We now show how to certify that a given initial configuration *cannot reach* any configuration from a given regular set R. The classic saturation algorithm for computing $\mathrm{Pre}^*(R)$, the set of (reflexive, transitive) predecessors of configurations in R, takes a $\mathcal{P}$-automaton $\mathcal{A}$ as input and iteratively adds transitions to it by the following rule:

$$\mathcal{P} \text{ has transition } (p, A) \to (q, w), \mathcal{A} \text{ has walk } q \xrightarrow{w} s \implies \text{add transition } p \xrightarrow{A} s \text{ to } \mathcal{A}. \tag{4}$$

By the following claim, saturation under (4) implies overapproximation of $\mathrm{Pre}^*(R)$. The converse inclusion is more subtle and will not be required.

Claim 28 (see, e.g., [Carayol and Hague 2014, Section 3.2]). *A $\mathcal{P}$-automaton $\mathcal{A}$ accepts all configurations from* $\mathrm{Pre}^*(R)$ *if (i) it contains all transitions of the original $\mathcal{P}$-automaton and (ii) it is saturated, i.e., applying rule (4) does not change the transition relation.*

Our certificate system for non-reachability relies on the observation that the update rule (4) can be expressed using matrix multiplication. A *certificate* is a finite family of matrices, M^A, $M^{A,B}$, $M_1^{A,B,C}$, $M_2^{A,B,C}$, for all $A, B, C \in \Gamma$, satisfying the following conditions:

$$\begin{aligned} P^A &\le M^A, & (M^{\gamma_0})_{q_0,f} &= 0 \quad \text{for all } f \in F,\\ T^{A,\varepsilon} &\le M^A, & & \\ \mathrm{bool}(M^{A,B}) &\le M^A, & M^{A,B} &= T^{A,B} \cdot M^B, \\ \mathrm{bool}(M_2^{A,B,C}) &\le M^A, & M_1^{A,B,C} &= T^{A,BC} \cdot M^B, \quad M_2^{A,B,C} = M_1^{A,B,C} \cdot M^C, \end{aligned} \tag{5}$$

where we assume with no loss of generality that $S = \{1, \dots, |S|\}$ and denote by P^A the A-transition matrix of the original $\mathcal{P}$-automaton and, for all $A \in \Gamma$, $w \in \Gamma^{\le 2}$, by $T^{A,w} = (t_{ij}^{(A,w)})$ the 0–1 matrix of size $|S| \times |S|$ in which $t_{ij}^{(A,w)} = 1$ if $i, j \in Q$ and $\mathcal{P}$ contains a transition $(i, A) \to (j, w)$. The following proposition summarises the properties of this system:

Proposition 29.

- *Certificates have $O(|\Gamma|^3|S|^2)$ entries.*
- *An instance of PDA emptiness is a no-instance iff there exists a certificate for it.*
- *The conditions can be verified by a deterministic algorithm with running time $O(|\Gamma|^3|S|^\omega)$ or a randomized algorithm with running time $O(|\Gamma|^3|S|^2)$ that accepts valid certificates with probability one and rejects invalid ones with probability ≥ 0.5.*

Proof. Let $\mathcal{A}$ be a $\mathcal{P}$-automaton (saturated or not). For each $A \in \Gamma$, let $M^A = (m_{ij})$ denote the A-transition matrix of $\mathcal{A}$, that is, the 0–1 matrix of size $|S| \times |S|$ in which $m_{ij} = 1$ if $\mathcal{A}$ contains a transition $i \xrightarrow{A} j$ and $m_{ij} = 0$ otherwise. Then rule (4) can be decomposed into the following updates, for all $A, B, C \in \Gamma$:

$$\begin{aligned} M^A &:= \mathrm{bool}(M^A + T^{A,\varepsilon}),\\ M^A &:= \mathrm{bool}(M^A + T^{A,B} \cdot M^B),\\ M^A &:= \mathrm{bool}(M^A + T^{A,BC} \cdot M^B \cdot M^C). \end{aligned}$$

The composition of certificates (5) and the verification algorithms follow as in Subsection 3.2. □

Matrix constraints of Equation (5) define a *backwards invariant* for the pushdown system $\mathcal{P}$ in question, an overapproximation of the set of configurations from which R is reachable.

Corollary 30. *PDA emptiness has subcubic certificates if $|\Gamma| = O(|S|^\beta)$ for $\beta < 1 - \omega/3$, where ω is the matrix multiplication exponent.*

6 DISCUSSION: FINE-GRAINED LANDSCAPE AND A HARDEST D_2Reach INSTANCE

6.1 2NPDA Completeness

In interprocedural program analysis, the lack of algorithms with running time $O(n^{3-\varepsilon})$ is referred to as "the cubic bottleneck". Heintze and McAllester [1997] captured this phenomenon by the class of "2NPDA-complete" problems. Here "2NPDA" stands for *two-way nondeterministic pushdown automata,* a model of computation that extends standard PDA with the ability to move back and forth on the (read-only) input tape [Aho et al. 1968]. A problem is 2NPDA-complete (following

Neal [1989]) if it is subcubic equivalent to *2NPDA recognition*: given a word, does it belong to the language of a fixed 2NPDA. Heintze and McAllester show a number of 2NPDA-complete problems, including ground monadic rewriting reachability (see also [Neal 1989]), data flow reachability, control flow reachability, and certain (non-)typability problems. Melski and Reps [2000] show a reduction from CFL reachability to data flow reachability and set constraints (and thus to 2NPDA recognition) and a reverse reduction from data flow reachability to an instance of CFL reachability where the language is *not* fixed.

The following result appears to be folklore but is not found in the literature, strengthening the reduction of Melski and Reps to show hardness of CFL reachability for the *fixed* Dyck-2 language. The equivalence between problems (1) and (2) is sketched by Chaudhuri [2008]. While we state the result for PDA emptiness, one can equivalently (or additionally) state it for pushdown reachability. We defer technical details to Section 7.

Proposition 31. *The following problems either all have truly subcubic algorithms, or none of them do: (1) 2NPDA language recognition, (2) PDA language emptiness, and (3)* D_2Reach.

Proof (sketch). We show three reductions:

- In 2NPDA recognition to PDA emptiness, each control state of the PDA remembers the position of the 2NPDA on the input tape and the control state of the 2NPDA. The size of PDA is linear in the length of the input word, because the 2NPDA is fixed.
- In PDA emptiness to D_2Reach, the graph mimics the transition diagram of the PDA. Stack symbols from Γ are encoded by sequences of opening parentheses of two kinds of length $\lceil \log |\Gamma| \rceil$. Push transitions are modelled by sequences of edges with these labels, and pop transitions by sequences with matching closing parentheses. The reduction is linear-time, because the bit size of the PDA accounts for the $\log |\Gamma|$ factor.
- In the last reduction, we give a fixed 2NPDA that solves D_2Reach. The 2NPDA guesses a path through the graph, maintaining at the bottom of the stack a sequence $\sigma \in \{(, [\}^*$, and the current vertex at the top of the stack. The length of the input word is proportional to the bit size of the graph (adjacency lists). □

As a corollary, all of these problems have subcubic certificate schemes, and an analogue of Theorem 3 holds for them too (worked out for PDA emptiness in Section 5). Theorem 25 on the absence of SETH-hardness also extends to PDA emptiness and 2NPDA recognition.

For upper bounds, note that 2NPDA recognition is solvable in time $O(|w|^3/\log|w|)$ [Rytter 1985], and language emptiness for PDA in time $O(n^3/\log n)$. Language emptiness is undecidable for 2NPDA, even when the automata are deterministic and use a counter instead of a pushdown. (This observation makes the cubic recognition algorithm somewhat surprising in comparison.)

The PDA emptiness to Dyck-2 reachability reduction from Proposition 31, combined with Chaudhuri's algorithm for CFL reachability [Chaudhuri 2008], implies a slightly subcubic bound for PDA emptiness. Indeed, Chaudhuri shows how to solve instances of CFL reachability for a fixed language (including the Dyck-2 language) in time $O(n^3/\log n)$, where n is the number of nodes in the graph. Suppose we start with a PDA emptiness instance with s states, t transitions, and r stack symbols. Note that we can safely ignore the input alphabet symbols. The bit size of the instance is $b = O(t \log(s + r))$. The reduction from Lemma 36 gives an instance of Dyck-2 reachability with $O(b)$ nodes. Chaudhuri solves it in time $O(b^3/\log b)$, which is subcubic in the bit size of the input of PDA emptiness (although not necessarily subcubic in $s + t$). This complexity seems folklore but was never made explicit. In particular, "textbook" algorithms for PDA emptiness go through equivalent context-free grammars [Hopcroft et al. 2006], for which a cubic blow-up is unavoidable in the worst case [Goldstine et al. 1982].

6.2 A Hardest 2NPDA Language

We observe that the hardness of 2NPDA recognition is witnessed by a single "hardest" 2NPDA language: recognition for an arbitrary 2NPDA can be reduced to a single 2NPDA. Suppose some 2NPDA $\mathcal{A}$ over Σ is given and the input to 2NPDA recognition for $\mathcal{A}$ is a word w. Applying our cycle of reductions from Proposition 31 (to PDA emptiness, then to Dyck-2 reachability, and then back to 2NPDA recognition), we get another word $u = u(\mathcal{A}, w)$ and a 2NPDA $\mathcal{B} = \mathcal{B}(\mathcal{A}, w)$ such that $\mathcal{B}$ accepts u iff $\mathcal{A}$ accepts w. But $\mathcal{B}$ in fact doesn't depend on $\mathcal{A}$ or w, because it is a fixed 2NPDA for $\mathsf{D}_2\mathsf{Reach}$. One refers to such languages as *hardest* 2NPDA languages, because the recognition problem for $L(\mathcal{B})$ cannot be easier than the recognition problem for any 2NPDA language L. The following theorem states this result in language-theoretic terms. (Recall that a homomorphism is a mapping, say $h\colon \Sigma^* \to \Sigma_0^*$, such that $h(uv) = h(u)h(v)$ for all $u, v \in \Sigma^*$.)

Theorem 32. *There exists a 2NPDA $\mathcal{A}_0$ over an input alphabet Σ_0 with the following property: for every 2NPDA $\mathcal{A}$ over every finite Σ there is a homomorphism $h\colon \Sigma^* \to \Sigma_0^*$ such that, for all $w \in \Sigma^+$, $w \in L(\mathcal{A})$ if and only if $h(w) \in L(\mathcal{A}_0)$.*

Essentially, $\mathcal{B} = \mathcal{A}_0$. Working out the details shows that the mapping $u(\mathcal{A}, \cdot)$ can be made a homomorphism for every $\mathcal{A}$. This requires an appropriate encoding for inputs to $\mathcal{A}_0$. Technical details are provided in Section 8; we only sketch the intuition here.

Remark 33. Rytter [1981] showed there is a fixed hardest 2NPDA language L_0,[2] based on the classic hardest context-free language by Greibach [1973]. Theorem 32 identifies a different hardest 2NPDA language. In contrast with Rytter's proof, our construction is self-contained and does not depend on Greibach's hardest CFL. Instead, our new hardest 2NPDA language is an encoding of a restricted version of Dyck-2 Reachability.

We now describe the hardest language $L(\mathcal{A}_0)$. The alphabet is $\Sigma_0 = \{(,), [,], \#, 1, -, *\}$. The language contains only words of the form

$$\underbrace{\# \ell_1 o_1 * \ldots * \ell_q o_q}_{\text{block}} \underbrace{\# \ell_{q+1} o_{q+1} * \ldots * \ell_r o_r}_{\text{block}} \# \ldots \# \underbrace{\ldots * \ell_m o_m}_{\text{block}} \tag{6}$$

and the membership of such words in the language is determined as follows. Consider a directed graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ where n is the number of *blocks* separated by the vertex marker #. An edge $e = (i, j)$ belongs to E if and only if the ith block has a subword $\ell_p o_p$ with $\ell_p \in \{(,), [,]\}$, $o_p = 1^k$ or $o_p = -1^k$ where $j = i + k$ (or $j = i - k$, respectively) and this subword is preceded and followed by symbols from $\{\#, *\}$ or tape endmarker. The edge label is in this case $\lambda(e) = \ell_i$. (If for some i and k the index j is "off the tape", the tape endmarker counts as one virtual vertex and then the counting reverses the direction, "reflecting" off the endmarker.) The word belongs to $L(\mathcal{A}_0)$ if and only if $(G, \lambda, 1, n)$ is a yes-instance of $\mathsf{D}_2\mathsf{Reach}$, i.e., if G contains a walk from 1 to n labelled by a word from the Dyck-2 language.

To sum up, this restricted version of 2NPDA recognition is the "hard core" of the problem: by Theorems 32 and 31, in order to find subcubic algorithms for $\mathsf{D}_2\mathsf{Reach}$, it suffices to handle instances obtained from it (exploiting any structural properties). PDA emptiness and $\mathsf{D}_2\mathsf{Reach}$ are already hard for sparse graphs: a truly subcubic algorithm for either problem restricted to graphs with a linear number of edges would already result in a breakthrough algorithm for 2NPDA recognition.

[2] Actually, Rytter only proves that, for all $w \in \Sigma^+$, one has $w \in L$ iff $h(w\$) \in L_0$.

7 APPENDIX I: PROOF OF PROPOSITION 31

Preliminary Definitions. Two-way nondeterministic pushdown automata (2NPDA) [Gray, Harrison, and Ibarra 1967] are a powerful formalism introduced in the 1960s. 2NPDA have the form $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q is a finite set of states, Σ are Γ are finite alphabets of input and stack symbols, respectively, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and a transition relation $\delta \subseteq Q \times \Sigma \times \Gamma \times Q \times \Gamma^* \times \{-1, 0, +1\}$. We assume Σ contains two designated "end of tape" symbols $\triangleleft$ and $\triangleright$. We assume that Γ contains a designated "end of stack" symbol Z_0 such that any transition $(q, \sigma, Z_0, q', w, d) \in \delta$ satisfies $w = Z_0$. Thus, no transition of $\mathcal{A}$ replaces Z_0 on the stack with a different symbol; also, no transition pushes Z_0.

Informally, the 2NPDA $\mathcal{A}$ has a finite control (states from Q) which reads a symbol of Σ on its input tape and the top symbol in Γ of a pushdown store. Based on the transition relation δ, 2NPDA moves by changing the control state, replacing the top symbol of the pushdown store by a finite string of symbols (possibly the empty string), and moving its input head at most one symbol left or right. Initially, the 2NPDA is in state q_0, and its pushdown store consists of the single symbol Z_0. The input tape consists of a word $w \in (\Sigma \setminus \{\triangleleft, \triangleright\})^*$ surrounded by a left marker $\triangleleft$ and a right marker $\triangleright$ and the 2NDPA scans the left marker $\triangleleft$.

Remark 34. We include the endmarkers $\triangleleft$ and $\triangleright$ in the set Σ here, even though we did not mention them back in Section 6 when specifying the alphabet Σ_0 for our hardest 2NPDA language. Naturally, all symbols used by automata (including the endmarkers) should be included in the tape alphabet of these automata.

A configuration of the 2NPDA $\mathcal{A}$ is a triple $(q, w\hat{a}x, \gamma)$, where $q \in Q$, $w, x \in \Sigma^*$, $a \in \Sigma$, and $\gamma \in \Gamma^+$. The "hat" on a denotes that the machine is currently scanning the letter a. We write $(q_1, a_1 \dots \hat{a}_i \dots a_n, Z\gamma) \to (q_2, a_1 \dots \hat{a}_j \dots a_n, \gamma'\gamma)$ whenever $(q_1, a_i, Z, q_2, \gamma', d) \in \delta$ for $d \in \{-1, 0, +1\}$, and $j = i + d$. We require $j \in \{1, \dots, n\}$, that is, the scan position does not "fall off" the input word. Note that the input tape is not changed, only the scan position may change. We write $\to^*$ for the reflexive and transitive closure of $\to$. A word $w \in (\Sigma \setminus \{\triangleleft, \triangleright\})^*$ is *accepted* by the 2NPDA if $(q_0, \hat{\triangleleft}w\triangleright, Z_0) \to^* (q, \triangleleft w\hat{\triangleright}, Z_0)$ for some $q \in F$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is the set of all accepted words (in $(\Sigma \setminus \{\triangleleft, \triangleright\})^*$).

Informally, the 2NPDA has some run that leads it from the initial configuration with the word on the input tape to a final state. Without loss of generality, we can assume above that a word is accepted in a final state with the 2NPDA scanning the right end marker and the pushdown store only contains Z_0. The transition relation is nondeterministic; we only require that some run is accepting. For the reader familiar with one-way automata, we remark that the role of epsilon-transitions is played by explicit specification of head movements.

A 1NPDA, or just PDA for short, is a 2NPDA such that $\delta \subseteq Q \times \Sigma \times \Gamma \times Q \times \Gamma^* \times \{0, +1\}$. Informally, the transitions of a PDA do not allow the scan position to move left, so PDA can only move left to right. PDA accept exactly the context-free languages. In comparison, 2NPDA are surprisingly powerful devices. In fact, even their deterministic counterparts (introduced by Stearns et al. [1965]) can recognize languages such as $\{a^n b^{p(n)} \mid n \geq 0\}$ where p is a fixed polynomial with natural coefficients and $\{x\#y \mid x \text{ is a subword (factor) of } y\}$ [Galil 1977; Rytter 1987].

We consider the following decision problems for these machine classes. The *recognition* problem for a class of machines C asks, for a fixed machine $M \in C$ and an input word $w \in \Sigma^*$, if w is accepted by M, i.e., if $w \in L(M)$. The *emptiness* problem for class C asks, given a machine $M \in C$, if $L(M) = \emptyset$. Everywhere below, $|M|$ denotes the bit size of M's description.

Proposition 31 follows from Lemmas 35, 36, and 37, which we prove next.

Lemma 35. *There exists a linear-time algorithm that, given a 2NPDA $\mathcal{B}$ and a word w, outputs a PDA $\mathcal{P}$ such that:*

- $|\mathcal{P}| \leq O(|w|)$ *for any fixed* $\mathcal{B}$ *and*
- *the language of* $\mathcal{P}$ *is nonempty iff* $\mathcal{B}$ *accepts* w.

Remark. *In fact,* $|\mathcal{P}| \leq O(|\mathcal{B}| \cdot |w|)$.

Proof. Denote $n = |w|$ and let S be the set of control states of $\mathcal{B}$. Construct a PDA $\mathcal{P}$ with the set of control states $Q = \{0, 1, \ldots, n+1\} \times S$. The first component of the states of $\mathcal{P}$ corresponds to a possible position of the input head of the 2NPDA $\mathcal{B}$ run on w. Indeed, when $\mathcal{B}$ is run on the word w, its head has $n+2$ possible positions: over any of the n letters of w, over the left endmarker, and over the right endmarker.

PDA $\mathcal{P}$ has the initial state $(0, s_0)$, where s_0 is the initial state of $\mathcal{B}$. Transitions of the (nondeterministic) PDA $\mathcal{P}$ are defined so that $\mathcal{P}$ would simulate the (nondeterministic) computation of $\mathcal{B}$ on w. The stack of $\mathcal{P}$ is always the same as the stack of $\mathcal{B}$, and the second component of the control state of $\mathcal{P}$ the same as the control state of $\mathcal{B}$. Transitions of $\mathcal{B}$ depend on the input letter, which is available to $\mathcal{P}$, because $\mathcal{P}$ 'remembers' in the control state where the input head of $\mathcal{B}$ is positioned—and the input word w is fixed. Transitions of $\mathcal{P}$ need not read any letter from the input; $\mathcal{P}$ accepts (rejects) whenever so does $\mathcal{B}$. It is straightforward to see that both assertions of the lemma hold. □

Lemma 36. *There exists a linear-time algorithm that, given a PDA $\mathcal{P}$, outputs a directed graph $G = (V, E)$, labels $\lambda \colon E \to \{(,), [,]\}$ and two vertices $s, t \in V$ such that (G, λ, s, t) is a yes-instance of* $\mathsf{D}_2\mathsf{Reach}$ *iff the language of $\mathcal{P}$ is nonempty.*

Proof. We show how to construct the required instance of $\mathsf{D}_2\mathsf{Reach}$ given a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

The idea is that we encode stack symbols from Γ by sequences of words over the alphabet $\{(, [\}$; pushing symbols on the stack corresponds to traversing edges of G labeled by opening brackets, and popping symbols—to traversing edges labeled by closing brackets.

Let $\ell = \lceil \log |\Gamma| \rceil$. Fix any injective maps $\phi \colon \Gamma \to \{(, [\}^\ell$ and $\psi \colon \Gamma \to \{),]\}^\ell$ such that, for all $Z \in \Gamma$, the words $\psi(Z)$ is obtained from $\phi(Z)$ by switching opening brackets to closing brackets without changing their type—i.e., `(` is replaced by `)` and `[` by `]`; and then reversing the word.

We next construct an auxiliary graph $G' = (V', E')$ with labels $\lambda \colon E \to \{(,), [,]\}$. The set V' contains Q as a subset. For each transition $(q, a, Z, q', \gamma, d) \in \delta$, the graph G' contains a path from q to q' of length $\ell \cdot (1 + |\gamma|)$. The edges of this path are labelled by consecutive letters of the word $\psi(Z) \cdot \phi(\gamma)$; all intermediate vertices are distinct and are incident to no other edge of G'. It is easy to see that the number of edges of G' does not exceed $|\mathcal{P}|$. (Notice that the input letter a is ignored in this construction.)

Recall that the automaton $\mathcal{P}$ has a nonempty language if and only if there is a path from its initial configuration to a final configuration, enabled by some input word from Σ^*. The initial configuration c_0 of $\mathcal{P}$ has control state q_0 and stack content Z_0; and any final configuration c has some control state $q \in F$ and the same stack content Z_0. By construction, $c_0 \to^* c$ in the PDA $\mathcal{P}$ if and only if the graph G' has a walk from q_0 to q labeled by a word $u \in \{(,), [,]\}^*$ such that $\phi(Z_0) \cdot u \cdot \psi(Z_0)$ is a Dyck-2 word.

It now remains to obtain the graph G from G' by adding fresh states s and t and connecting them to the other vertices by (1) a path from s to q_0 labeled by $\phi(Z_0)$ and (2) paths from each $q \in F$ to t labeled by $\psi(Z_0)$. Each of these paths has length ℓ; paths of type (2) have $\ell - 1$ edges in common. Now (G, λ, s, t) is the instance of $\mathsf{D}_2\mathsf{Reach}$ with the required property. □

Lemma 37. *There exist a 2NPDA language $\mathcal{L}'$ and a linear-time algorithm that, given a directed graph $G = (V, E)$ with labels $\lambda\colon E \to \{(,), [,]\}$ and two vertices $s, t \in V$, outputs a word w such that $w \in \mathcal{L}'$ iff (G, λ, s, t) is a yes-instance of* $\mathsf{D}_2\mathsf{Reach}$.

Remark 38. For the linear time bound, we assume that G is encoded in binary in the input. If this is not the case, the running time of the algorithm suffers a slowdown by a factor of $O(\log |V|)$.

Proof. Words of the language $\mathcal{L}'$ are encodings of the quadruples (G, λ, s, t), where the vertices of G are encoded in binary. In more detail, every $w \in \mathcal{L}'$ has the following form: first an encoding of s, then an encoding of t, and finally a sequence of encodings of edges of G, where every edge $e \in E$ is followed by its label $\lambda(e)$. All these encodings are separated by delimiters.

The language $\mathcal{L}'$ is over an alphabet of size $O(1)$; a word belongs to $\mathcal{L}'$ iff it follows the format we have just described and the graph G has a walk from s to t labeled with a sequence from the Dyck-2 language over $\{(,), [,]\}$. The algorithm from the assertion of the lemma simply writes down the encodings in the required format; it is clear that the algorithm runs in linear time and the obtained word belongs to $\mathcal{L}'$ iff (G, λ, s, t) is a yes-instance of $\mathsf{D}_2\mathsf{Reach}$.

It remains to prove that the language $\mathcal{L}'$ is recognized by a 2NPDA. Let us describe this 2NPDA $\mathcal{R}$. It first reads the input word and checks that it follows the format described above. If this is not the case, $\mathcal{R}$ rejects, otherwise it guesses the required walk in G from s to t as follows. A configuration of $\mathcal{R}$ stores on the stack the following data:

- (at the bottom) a sequence $\sigma \in \{(, [\}^*$, and
- (at the top) a vertex $v \in V$.

In this configuration, $\mathcal{R}$ has already found a walk from $s \in V$ to $v \in V$ labeled with some word $\sigma' \in \{(,), [,]\}^*$ that reduces to σ. (A word $\sigma' \in \{(,), [,]\}^*$ reduces to σ if σ can be obtained from σ' by a sequence of transformations that replace subwords `()` and `[]` with ε.) Here is how $\mathcal{R}$ works:

(1) At the beginning, initialize σ with the empty word and v with $s \in V$, pushing them to the stack.
(2) Repeatedly guess the next edge $e \in E$ in the walk (leaving the loop nondeterministically after some iteration):
   (a) move the head to the encoding of $e = (u_1, u_2)$ written on the input tape;
   (b) pop the encoding of $v \in V$ from the stack, reading the encoding of u_1 from the input tape in sync; if $u_1 \neq v$, reject;
   (c) look at the label $\lambda(e)$:
      - if $\lambda(e) \in \{(, [\}$, then push $\lambda(e)$ onto the stack, extending the current $\sigma \in \{(, [\}^*$, and
      - if $\lambda(e) \in \{),]\}$, then pop the last symbol of $\sigma \in \{(, [\}^*$; proceed if the two symbols form a matching pair, otherwise reject (also reject if σ is empty);
   (d) push the encoding of u_2 to the stack.
(3) Check if the current vertex v is equal to t and σ is empty. Accept if the check succeeds, otherwise reject.

An accepting computation of $\mathcal{R}$ exists iff (G, λ, s, t) is a yes-instance of $\mathsf{D}_2\mathsf{Reach}$. □

8 APPENDIX II: PROOF OF THEOREM 32

Fix an arbitrary 2NPDA $\mathcal{A}$ over a finite alphabet Σ. We can assume with no loss of generality that $\mathcal{A}$ has a single final state and that it is different from its initial state: $|F| = 1$, $q_0 \notin F$. (It is an easy exercise to modify $\mathcal{A}$ to ensure this assumption holds.)

Suppose an input word $w \in \Sigma^*$ is given. Lemma 35 reduces $L(\mathcal{A})$ to the emptiness problem for a PDA defined as a product of the word w and 2NPDA $\mathcal{A}$. More concretely, this PDA has control states $Q = \{0, 1, \ldots, n{+}1\} \times S$ where S is the set of control states of $\mathcal{A}$. Note that $|Q| = O(|w| \cdot |\mathcal{A}|) = O(|w|)$

since $\mathcal{A}$ is fixed. Here and below, the constant behind $O(\cdot)$ depends on $\mathcal{A}$ but not on w. Similarly, the stack alphabet of the PDA is fixed too. We now give this PDA as input to a further reduction to Dyck-2 Reachability (Lemma 36), which produces an instance (G, λ, s, t).

Claim 39. *The graph G has the following properties:*

(a) *it has $O(|w|)$ vertices (including intermediate ones, resulting from mapping the stack alphabet into binary words);*
(b) *its edges are labeled with symbols from $\{(,), [,]\}$;*
(c) *there is a linear order on the vertices such that each edge connects two vertices that are $O(1)$ positions away from each other in this order;*
(d) *the source is first and the sink is last in the order.*

Proof of Claim 39. Property (a) is due to the fact that $\mathcal{A}$, and thus its stack alphabet, is fixed. Property (b) is immediate. Property (c) ultimately reflects the fact that $\mathcal{A}$, as a two-way pushdown automaton, cannot jump cells of the input tape, that is, its head can only move one cell left or right if it moves at all —this is represented by $d \in \{-1, 0, +1\}$ in the syntax of 2NPDA. Thus, the linear order on vertices of the graph is inherited from the natural ordering of letters of the input tape, $\triangleleft w \triangleright$. Reductions to PDA emptiness and $\mathsf{D}_2\mathsf{Reach}$ effectively apply a direct product construction with a constant factor expansion. Within each block corresponding to an input letter, vertices can be ordered arbitrarily, provided that the initial state of $\mathcal{A}$ comes first and the final state last—ensuring property (d). Note that our preprocessing of $\mathcal{A}$ ensures that these two states are different, and our acceptance condition and subsequent reductions do the rest of the work. □

We refer to instances (G, λ, s, t) with the properties stated in Claim 39 as those of *Restricted Dyck-2 Reachability*.

Suppose $k \in \mathbb{N}$ is chosen such that the constants behind $O(\cdot)$ in conditions (a) and (c) are at most k and every vertex has at most k outgoing edges. We think of this $k = O(1)$ as the "width" of the instance, which depends on the original 2NPDA $\mathcal{A}$ but not on w.

Remark 40. The constant $O(1)$ in property (c) is reminiscent of the bounded pathwidth condition (see, e.g, [Bienstock et al. 1991]). However, in our case the graph has an even more "regular" structure. We leave it open whether this structure can be characterized by constant pathwidth *and* constant degree (and restricting the direction and labels of the edges). In comparison, Chatterjee and Osang look at pushdown reachability with constant *treewidth* [Chatterjee and Osang 2017].

It remains to map this instance of Dyck-2 Reachability to an instance of 2NPDA recognition, for a fixed 2NPDA which we now define.

For each vertex v, let $\mathsf{index}(v)$ denote the position of v in the order specified in property (c), ranging from 1 to $O(|w|)$. (Once again, the constant behind $O(\cdot)$ depends on $\mathcal{A}$ but not on w.) The construction below follows in spirit the proof of Lemma 37 and refines the details in order to produce a homomorphism h. The key difference is that, to produce the new input word, we will not write edges as "$(u, v), \lambda(u, v)$". Instead we will:

1) sort the vertices u according to their $\mathsf{index}(u)$ ascending and, for each u, group all the edges departing from u together (each u will have at most k outgoing edges);
2) write edges (u, v) as pairs $(\lambda(u, v), \mathsf{offset}(u, v))$ where $\mathsf{offset}(u, v) = \mathsf{index}(v) - \mathsf{index}(u)$, i.e., how many vertices to the right the destination of the edge is; this difference is written in unary notation (without incurring blowup, as this difference cannot exceed k);
3) write vertices as "separators" between groups of edges.

Putting everything together, the input to the new 2NPDA has the form (6) (see page 19), where # is the vertex marker symbol, $\ell_i \in \{(,), [,]\}$, and each o_i is either the empty word, or $1 \dots 1$ or $-1 \dots 1$.

Claim 41. *The set of valid encodings (6) of Restricted Dyck-2 Reachability can be recognized by a fixed 2NPDA.*

The construction of the 2NPDA in Claim 41 is similar to the reduction of Lemma 37. Instead of guessing the next vertex, this new 2NPDA $\mathcal{A}'_0$ "scrolls" left and right in a deterministic way to the destination of the current edge, counting in unary with the help of its stack. The nondeterministic choices that $\mathcal{A}'_0$ makes are which outgoing edge from the current vertex to choose next.

Note that the construction of $\mathcal{A}'_0$ is independent of k, thus identifying a single hardest language, $L(\mathcal{A}'_0)$. Moreover, for a given initial 2NPDA $\mathcal{A}$ this reduction replaces each symbol in w with $O(1)$ vertices and $O(1)$ edges, where this $O(1)$ depends just on $\mathcal{A}$ and not w. The exact collection of these vertices and edges is fully determined by each symbol of w, independently of its position within w. The vertices are not addressed in any "absolute" numbering scheme — so this mapping can be realised as a homomorphism.

Remark 42. The use of relative rather than absolute addresses (to encode offsets) appears in a related context but for a different problem in Neal's work on taxonomic inference [Neal 1989], which is at the origin of the connection between 2NPDA and program analysis.

Summary and the endmarkers problem. We now have achieved the following: for every 2NPDA $\mathcal{A}$ there is a homomorphism h_1 such that $w \in L(\mathcal{A})$ if and only if $h_1(\trianglelefteq w \trianglerighteq) \in L(\mathcal{A}'_0)$. Note the appearance of the endmarkers here. (We use $\trianglelefteq$ instead of $\triangleleft$ and $\trianglerighteq$ instead of $\triangleright$ to avoid a notation clash in the discussion that follows.) They reflect the fact that, in the chain of our reductions, the set of control states of the PDA is $\{0, 1, \dots, n+1\} \times S$ not $\{1, \dots, n\} \times S$.

To lift our construction from $\trianglelefteq w \trianglerighteq$ to just w, it may be tempting to appeal to the following fact, which is not difficult to prove. Let $x, y \in \Sigma^*$ be fixed. Suppose a 2NPDA accepts a language $L \subseteq x \cdot \Sigma^* \cdot y$. Then there exists another 2NPDA which accepts the language $\{w \mid xwy \in L\}$.

Unfortunately, this fact does not quite achieve our goal. This is because the new 2NPDA we would obtain from it depends on x and y. In our context, x and y should be the images of the original endmarkers, i.e., we would like to have $x = h_1(\trianglelefteq)$ and $y = h_1(\trianglerighteq)$. But these two words depend on the homomorphism h_1, and thus on the 2NPDA $\mathcal{A}$ that we started from. This is at odds with our objective: we need a single 2NPDA for our hardest language, not an entire family dependent on $\mathcal{A}$.

There are several ways to deal with this issue. One is reminiscent of Rytter's approach [Rytter 1981]: we can decide we are content with keeping a single endmarker in, i.e., we would only like to find an L_0 such that, for all $w \in \Sigma^+$, one has $w \in L$ iff $h(w\$) \in L_0$. Here \$ is a fresh symbol. It is not very difficult to find such an h and L_0 based on our construction: essentially, the word $h_1(\trianglelefteq)$ needs to be merged with the word $h_1(\trianglerighteq)$ and placed to the right of $h_1(w)$. So we would like to choose $h(\$) = h_1(\trianglerighteq)h_1(\trianglelefteq)$ and $h(a) = h_1(a)$ for all other symbols a. The 2NPDA for L_0 is the same as our 2NPDA $\mathcal{A}'_0$ constructed above, with the following modification. Suppose it starts following an edge from some vertex (block) to the left but hits the left end of the tape, i.e., the left endmarker $\triangleleft$. We now use this symbol to refer to the tape alphabet of the 2NPDA $\mathcal{A}'_0$ (and not the tape alphabet of the original machine $\mathcal{A}$). The new 2NPDA will move all the way to the right end of the tape and continue its search for the destination vertex from the right endmarker $\triangleright$. Edges within $h_1(\trianglerighteq)$ need not be changed, but the ones among them that lead to the right ($\mathrm{offset}(u, v) > 0$, or equivalently $o_i \in 1^+$) will make the 2NPDA hit $\triangleright$, go back to the left of the tape and continue the search from $\triangleleft$.

One further technicality that needs to be dealt with is the beginning and end of the computation. Recall that our Restricted Dyck-2 Reachability asked for a path from the very first vertex to the

very last one. Since $h_1(\trianglelefteq)$ moved, we now need to change this convention. More concretely, the only two vertices that we can distinguish correspond to the *last two* control states and the head position over the *left* endmarker. So the original 2NPDA $\mathcal{A}$ needs to be changed accordingly.

While this approach recovers Rytter's result, we show below that there is a way to eliminate the extra symbol \$ altogether.

Merging endmarker blocks into other symbols. Our solution to the endmarkers problem acknowledges that the words $h_1(\trianglelefteq)$ and $h_1(\trianglerighteq)$ cannot be eliminated completely. Indeed, the vertices and edges that these two words encode correspond to the behaviour of the original 2NPDA $\mathcal{A}$ over the tape endmarkers, and this behaviour can contribute to the computations of $\mathcal{A}$ in a nontrivial way.

However, what we can do is to embed all this information into words $h(a)$ for *all* other symbols a. For a first intuition (to be amended later), we would like to set $h(a) = h_1(\trianglelefteq) \parallel h_1(a) \parallel h_1(\trianglerighteq)$ for all non-endmarker symbols a, where $\parallel$ denotes a specially tailored ternary version of the perfect shuffle operation. More concretely, let w_1, w_2, w_3 be arbitrary words such that, for some single ℓ, we have $w_i = \prod_{j=1}^{\ell} \#w_{i,j}$ where none of the words $w_{i,j}$ contains the vertex marker symbol #. Then

$$w_1 \parallel w_2 \parallel w_3 := \prod_{j=1}^{\ell} \#w_{1,j} \# w_{2,j} \# w_{3,j} \quad .$$

This shuffling relies on ℓ being the same for all three arguments, which ultimately means the same number of vertices (blocks) in all words $h_1(a)$. This is in fact ensured by our constructions above (although we could always achieve this by introducing extra dummy vertices where necessary).

As a result of this shuffling arrangement, we can think of new input words as having three interleaving "tracks", each containing a separate sequence of vertices. Naturally, this requires some changes to the wiring, as follows.

First, the offsets that specify the edges of the graph departing from the vertices of $h_1(a)$ need to be updated. This is not difficult. Recall that edge destinations are specified using relative addresses of vertices. For every edge from a vertex in $h_1(a)$, its offset needs to be multiplied by 3, so that the edge skips intermediate vertices from copies of $h_1(\trianglelefteq)$ and $h_1(\trianglerighteq)$.

Second, we need to provide a way for the new 2NPDA $\mathcal{A}_0$ to reach the vertices in $h_1(\trianglelefteq)$ and $h_1(\trianglerighteq)$. To achieve this, we consider the scenario in which $\mathcal{A}_0$ will traverse edges leading to a vertex in $h_1(\trianglelefteq)$. (The case of $h_1(\trianglerighteq)$ is handled in a symmetric way.) Suppose the head of the 2NPDA is over a block (vertex) within the leftmost $h_1(a)$. Taking an edge with a negative offset, it moves left but then hits the left tape endmarker $\triangleleft$. When it does so, the stack of the 2NPDA still contains the number of vertices to be skipped. The 2NPDA then needs to change from the second (main) track, which contains the information from $h_1(a)$s, to the first track, which stores multiple copies of the word $h_1(\trianglelefteq)$. Effectively, this amounts to treating $\triangleleft$ as just another # that on top of its usual function makes the machine change direction. After that, however, we see that the number of vertices to be skipped was counted from the right of $h_1(\trianglelefteq)$ and not from the left where the head of the automaton is now located. Thus, we *reverse* the encoding of each of $h_1(\trianglelefteq)$ and $h_1(\trianglerighteq)$, as follows: we re-define our special shuffle as

$$w_1 \parallel w_2 \parallel w_3 := \prod_{j=1}^{\ell} \#\overline{w}_{1,\ell+1-j} \# w_{2,j} \# \overline{w}_{3,\ell+1-j} \quad ,$$

where $\overline{u}$ is the same word as u in which every maximal subword of the form -1^m is replaced with 1^m and each 1^m, without a preceding $-$, with -1^m. The operation of our 2NPDA will depend on which "track" of the input it is over. The second track is the main mode. Over the first track:

- Edges previously specified by positive offset need to followed to the *left* instead of to the right (hence the $\overline{w}_{1,\ell+1-j}$ above and not $w_{1,\ell+1-j}$). If the left tape endmarker $\triangleleft$ is encountered,

the automaton transitions to the second (main) track, and only then continues to the right (in the normal mode).
- Edges specified by the negative offset need to be followed to the *right* instead of to the left (again, this matches the $\overline{w}_{1,\ell+1-j}$ above). We note that if the input word for our 2NPDA is the homomorphic image under h of some word in Σ^*, then the automaton will never leave the leftmost $h(a)$ while being on the first track, because the original 2NPDA $\mathcal{A}$ cannot move left from the left endmarker.

The third track is arranged in a similar way.

Importantly, while we apply these changes to h and the "wiring" of the graph, we can keep the semantics of our hardest language untouched. The "tracks" themselves need not enter the description of the language. The only new "feature" that is necessary is *changing* the tracks — and this can be achieved simply by specifying that when our new 2NPDA $\mathcal{A}_0$ encounters a tape endmarker during its operation, this endmarker is counted as a virtual vertex and "reflects" off it, continuing the countdown in the opposite direction.

However, as was the case with the approach described above and involving \$, our new construction of h breaks the convention about the source and target vertices in the Dyck-2 reachability instance (albeit in a slightly different way). Because of the effective reversal of vertex ordering within $h_1(\trianglelefteq)$ and $h_1(\trianglerighteq)$, the required adjustment to the original 2NPDA $\mathcal{A}$ is that its initial control state needs to be the *last* and its (only) final control state the *first* in the ordering.

To sum up, by applying these adjustments and "compiling" the homomorphism h the way we have described, we arrive at the desired 2NPDA $\mathcal{A}_0$. (Note that there is freedom in whether we take $\varepsilon \in L(\mathcal{A}_0)$ or $\varepsilon \notin L(\mathcal{A}_0)$, but as some 2NPDA languages contain ε and some do not, their homomorphic images will necessarily disagree on ε, no matter our choice of the homomorphism.)

ACKNOWLEDGMENTS

This work started in 2016 when Dmitry Chistikov was a postdoctoral researcher in Joël Ouaknine's group, supported by the European Research Council (ERC) consolidator grant AVS-ISS (648701). Rupak Majumdar was supported in part by the the Deutsche Forschungsgemeinschaft project 389792660 TRR 248–CPEC and by the European Research Council under the Grant Agreement 610150 (http://www.impact-erc.eu/) (ERC Synergy Grant ImPACT). Philipp Schepper is part of Saarbrücken Graduate School of Computer Science, Germany and was partially supported by the European Research Council (ERC) consolidator grant no. 725978 SYSTEMATICGRAPH.

REFERENCES

Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2015. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. IEEE Computer Society, 98–117.

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1968. Time and Tape Complexity of Pushdown Automaton Languages. *Information and Control* 13, 3 (1968), 186–206.

Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas W. Reps, and Mihalis Yannakakis. 2005. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.* 27, 4 (2005), 786–818. https://doi.org/10.1145/1075382.1075387

László Babai. 1985. Trading Group Theory for Randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, Robert Sedgewick (Ed.). ACM, 421–429. https://doi.org/10.1145/22145.22192

Arturs Backurs and Piotr Indyk. 2016. Which Regular Expression Patterns Are Hard to Match?. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, Irit Dinur (Ed.). IEEE Computer Society, 457–466. https://doi.org/10.1109/FOCS.2016.56

Daniel Bienstock, Neil Robertson, Paul D. Seymour, and Robin Thomas. 1991. Quickly excluding a forest. *J. Comb. Theory, Ser. B* 52, 2 (1991), 274–283. https://doi.org/10.1016/0095-8956(91)90068-U

Luc Boasson, Bruno Courcelle, and Maurice Nivat. 1981. The Rational Index: A Complexity Measure for Languages. *SIAM J. Comput.* 10, 2 (1981), 284–296. https://doi.org/10.1137/0210020

Ahmed Bouajjani, Javier Esparza, Alain Finkel, Oded Maler, Peter Rossmanith, Bernard Willems, and Pierre Wolper. 2000. An efficient automata approach to some problems on context-free grammars. *Inf. Process. Lett.* 74, 5-6 (2000), 221–227. https://doi.org/10.1016/S0020-0190(00)00055-7

Ahmed Bouajjani, Javier Esparza, and Oded Maler. 1997. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1243)*. Springer, 135–150.

Phillip G. Bradford. 2018. Efficient Exact Paths For Dyck and semi-Dyck Labeled Path Reachability. *CoRR* abs/1802.05239 (2018). arXiv:1802.05239

Karl Bringmann. 2018. Personal communication. (2018).

Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. 2017. A Dichotomy for Regular Expression Membership Testing. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, Chris Umans (Ed.). IEEE Computer Society, 307–318. https://doi.org/10.1109/FOCS.2017.36

Arnaud Carayol and Matthew Hague. 2014. Saturation algorithms for model-checking pushdown systems. In *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014 (EPTCS, Vol. 151)*, Zoltán Ésik and Zoltán Fülöp (Eds.). 1–24. https://doi.org/10.4204/EPTCS.151.1

Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. 2016. Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*. ACM, 261–270.

Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2018. Optimal Dyck reachability for data-dependence and alias analysis. *Proc. ACM Program. Lang.* 2, POPL (2018), 30:1–30:30. https://doi.org/10.1145/3158118

Krishnendu Chatterjee and Georg Osang. 2017. Pushdown reachability with constant treewidth. *Inf. Process. Lett.* 122 (2017), 25–29.

Swarat Chaudhuri. 2008. Subcubic algorithms for recursive state machines. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, George C. Necula and Philip Wadler (Eds.). ACM, 159–169. https://doi.org/10.1145/1328438.1328460

Don Coppersmith and Shmuel Winograd. 1990. Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.* 9, 3 (1990), 251–280.

Mateus de Oliveira Oliveira and Michael Wehar. 2018. Intersection Non-emptiness and Hardness Within Polynomial Time. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11088)*, Mizuho Hoshi and Shinnosuke Seki (Eds.). Springer, 282–290. https://doi.org/10.1007/978-3-319-98654-8_23

Mateus de Oliveira Oliveira and Michael Wehar. 2020. On the Fine Grained Complexity of Finite Automata Non-emptiness of Intersection. In *Developments in Language Theory - 24th International Conference, DLT 2020, Tampa, FL, USA, May 11-15, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12086)*, Natasa Jonoska and Dmytro Savchuk (Eds.). Springer, 69–82. https://doi.org/10.1007/978-3-030-48516-0_6

Danny Dolev, Shimon Even, and Richard M. Karp. 1982. On the Security of Ping-Pong Protocols. *Inf. Control.* 55, 1-3 (1982), 57–68. https://doi.org/10.1016/S0019-9958(82)90401-6

Henning Fernau. 2019. Modern Aspects of Complexity Within Formal Languages. In *Language and Automata Theory and Applications - 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11417)*, Carlos Martín-Vide, Alexander Okhotin, and Dana Shapira (Eds.). Springer, 3–30. https://doi.org/10.1007/978-3-030-13435-8_1

Henning Fernau and Andreas Krebs. 2017. Problems on Finite Automata and the Exponential Time Hypothesis. *Algorithms* 10, 1 (2017), 24. https://doi.org/10.3390/a10010024

Alain Finkel, Bernard Willems, and Pierre Wolper. 1997. A direct symbolic approach to model checking pushdown systems. In *Second International Workshop on Verification of Infinite State Systems, Infinity 1997, Bologna, Italy, July 11-12, 1997 (Electronic Notes in Theoretical Computer Science, Vol. 9)*, Faron Moller (Ed.). Elsevier, 27–37. https://doi.org/10.1016/S1571-0661(05)80426-8

Rusins Freivalds. 1979. Fast Probabilistic Algorithms. In *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979 (Lecture Notes in Computer Science, Vol. 74)*, Jirí Becvár (Ed.). Springer, 57–69. https://doi.org/10.1007/3-540-09526-8_5

Zvi Galil. 1977. Some Open Problems in the Theory of Computation as Questions about Two-Way Deterministic Pushdown Automaton Languages. *Mathematical Systems Theory* 10 (1977), 211–228.

Jonathan Goldstine, John K. Price, and Detlef Wotschke. 1982. A pushdown automaton or a context-free grammar: which is more economical? *Theoret. Comput. Sci.* 18 (1982), 33–40.

Jim Gray, Michael A. Harrison, and Oscar H. Ibarra. 1967. Two-Way Pushdown Automata. *Information and Control* 11, 1/2 (1967), 30–70.

Sheila A. Greibach. 1973. The Hardest Context-Free Language. *SIAM J. Comput.* 2, 4 (1973), 304–310. https://doi.org/10.1137/0202025

Nevin Heintze and David A. McAllester. 1997. On the Cubic Bottleneck in Subtyping and Flow Analysis. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science (LICS), Warsaw, Poland, June 29 - July 2, 1997*. IEEE Computer Society, 342–351. https://doi.org/10.1109/LICS.1997.614960

Jelle Hellings. 2020. Explaining Results of Path Queries on Graphs - Single-Path Results for Context-Free Path Queries. In *Software Foundations for Data Interoperability and Large Scale Graph Data Analytics - 4th International Workshop, SFDI 2020, and 2nd International Workshop, LSGDA 2020, held in Conjunction with VLDB 2020, Tokyo, Japan, September 4, 2020, Proceedings (Communications in Computer and Information Science, Vol. 1281)*, Lu Qin, Wenjie Zhang, Ying Zhang, You Peng, Hiroyuki Kato, Wei Wang, and Chuan Xiao (Eds.). Springer, 84–98. https://doi.org/10.1007/978-3-030-61133-0_7

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Russell Impagliazzo and Ramamohan Paturi. 2001. On the Complexity of k-SAT. *J. Comput. Syst. Sci.* 62, 2 (2001), 367–375. https://doi.org/10.1006/jcss.2000.1727

Somesh Jha and Thomas W. Reps. 2004. Model checking SPKI/SDSI. *J. Comput. Secur.* 12, 3-4 (2004), 317–353. http://content.iospress.com/articles/journal-of-computer-security/jcs209

Tracy Kimbrel and Rakesh K. Sinha. 1993. A Probabilistic Algorithm for Verifying Matrix Products Using $O(n^2)$ Time and $\log_2 n + O(1)$ Random Bits. *Inf. Process. Lett.* 45, 2 (1993), 107–110. https://doi.org/10.1016/0020-0190(93)90224-W

Ivan Korec and Jirí Wiedermann. 2014. Deterministic Verification of Integer Matrix Multiplication in Quadratic Time. In *SOFSEM 2014: Theory and Practice of Computer Science - 40th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 26-29, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8327)*, Viliam Geffert, Bart Preneel, Branislav Rovan, Julius Stuller, and A Min Tjoa (Eds.). Springer, 375–382. https://doi.org/10.1007/978-3-319-04298-5_33

Marvin Künnemann. 2018. On Nondeterministic Derandomization of Freivalds' Algorithm: Consequences, Avenues and Algorithmic Progress. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland (LIPIcs, Vol. 112)*, Yossi Azar, Hannah Bast, and Grzegorz Herman (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 56:1–56:16. https://doi.org/10.4230/LIPIcs.ESA.2018.56

Bernard Lang. 1994. Recognition can Be Harder Than Parsing. *Comput. Intell.* 10 (1994), 486–494. https://doi.org/10.1111/j.1467-8640.1994.tb00011.x

Lillian Lee. 2002. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM* 49, 1 (2002), 1–15.

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

Anders Alnor Mathiasen and Andreas Pavlogiannis. 2021. The fine-grained and parallel complexity of Andersen's pointer analysis. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–29. https://doi.org/10.1145/3434315

Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. 2011. Certifying algorithms. *Comput. Sci. Rev.* 5, 2 (2011), 119–161. https://doi.org/10.1016/j.cosrev.2010.09.009

David Melski and Thomas Reps. 2000. Interconvertibility of a class of set constraints and context-free-language reachability. *Theor. Comput. Sci.* 248(1-2) (2000), 29–98.

Radford Neal. 1989. The computational complexity of taxonomic inference. (1989). Unpublished manuscript. Available at http://www.cs.toronto.edu/~radford/ftp/taxc.pdf.

G.C. Necula. 1997. Proof carrying code. In *POPL 97: Principles of Programming Languages*. ACM, 106–119.

Laurent Pierre. 1992. Rational Indexes of Generators of the Cone of Context-Free Languages. *Theor. Comput. Sci.* 95, 2 (1992), 279–305. https://doi.org/10.1016/0304-3975(92)90269-L

Aaron Potechin and Jeffrey O. Shallit. 2020. Lengths of words accepted by nondeterministic finite automata. *Inf. Process. Lett.* 162 (2020), 105993. https://doi.org/10.1016/j.ipl.2020.105993

Thomas W. Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, Ron K. Cytron and Peter Lee (Eds.). ACM Press, 49–61. https://doi.org/10.1145/199448.199462

Wojciech Rytter. 1981. A Hardest Language Recognized by Two-Way Nondeterministic Pushdown Automata. *Inf. Process. Lett.* 13, 4/5 (1981), 145–146. https://doi.org/10.1016/0020-0190(81)90045-4

Wojciech Rytter. 1985. Fast Recognition of Pushdown Automaton and Context-free Languages. *Information and Control* 67, 1-3 (1985), 12–22.

Wojciech Rytter. 1987. 100 exercises in the theory of automata and formal languages. http://wrap.warwick.ac.uk/60795/ Research report RR-99, University of Warwick, Department of Computer Science, available at http://wrap.warwick.ac.

uk/60795/.

Stefan Schwoon. 2002. *Model checking pushdown systems.* Ph.D. Dissertation. Technical University Munich, Germany. http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/schwoon.html

Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II. 1965. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, USA, October 6-8, 1965*. IEEE Computer Society, 179–190. https://doi.org/10.1109/FOCS.1965.11

Joseph Swernofsky and Michael Wehar. 2015. On the Complexity of Intersecting Regular, Context-Free, and Tree Languages. 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, 414–426. https://doi.org/10.1007/978-3-662-47666-6_33

Roei Tell. 2019. Proving that *prBPP=prP* is as hard as proving that "almost *NP*" is not contained in *P*/poly. *Inf. Process. Lett.* 152 (2019). https://doi.org/10.1016/j.ipl.2019.105841

Leslie G. Valiant. 1975. General Context-Free Recognition in Less than Cubic Time. *J. Comput. Syst. Sci.* 10, 2 (1975), 308–315.

Virginia Vassilevska Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, Howard J. Karloff and Toniann Pitassi (Eds.). ACM, 887–898. https://doi.org/10.1145/2213977.2214056

Virginia Vassilevska Williams. 2018. On some fine-grained questions in algorithms and complexity. In *International Congress of Mathematicians (ICM'18)*. Available at https://eta.impa.br/dl/194.pdf and https://people.csail.mit.edu/virgi/eccentri.pdf.

Virginia Vassilevska Williams and R. Ryan Williams. 2018. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM* 65, 5 (2018), 27:1–27:38.

Mikhail Vyalyi. 2019. Personal communication. (2019).

Mikhail N. Vyalyi. 2011. On regular realizability problems. *Probl. Inf. Transm.* 47, 4 (2011), 342–352. https://doi.org/10.1134/S003294601104003X

Mikhail N. Vyalyi and Alexander A. Rubtsov. 2015. On regular realizability problems for context-free languages. *Probl. Inf. Transm.* 51, 4 (2015), 349–360. https://doi.org/10.1134/S0032946015040043

Michael Wehar. 2014. Hardness Results for Intersection Non-Emptiness. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8573)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer, 354–362. https://doi.org/10.1007/978-3-662-43951-7_30

Mihalis Yannakakis. 1990. Graph-Theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*. ACM Press, 230–242.