The Decidability and Complexity of Interleaved Bidirected Dyck Reachability

ADAM HUSTED KJELSTRØM, Aarhus University, Denmark

ANDREAS PAVLOGIANNIS, Aarhus University, Denmark

Dyck reachability is the standard formulation of a large domain of static analyses, as it achieves the sweet spot between precision and efficiency, and has thus been studied extensively. *Interleaved* Dyck reachability (denoted $\mathcal{D}_k \odot \mathcal{D}_k$) uses two Dyck languages for increased precision (e.g., context and field sensitivity) but is well-known to be undecidable. As many static analyses yield a certain type of *bidirected* graphs, they give rise to *interleaved bidirected Dyck reachability* problems. Although these problems have seen numerous applications, their decidability and complexity has largely remained open. In a recent work, Li *et al.* made the first steps in this direction, showing that (i) $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability (i.e., when both Dyck languages are over a single parenthesis and act as counters) is computable in $O(n^7)$ time, while (ii) $\mathcal{D}_k \odot \mathcal{D}_k$ reachability is NP-hard. However, despite this recent progress, most natural questions about this intricate problem are open.

In this work we address the decidability and complexity of all variants of interleaved bidirected Dyck reachability. First, we show that $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability can be computed in $O(n^3 \cdot \alpha(n))$ time, significantly improving over the existing $O(n^7)$ bound. Second, we show that $\mathcal{D}_k \odot \mathcal{D}_1$ reachability (i.e., when one language acts as a counter) is decidable, in contrast to the non-bidirected case where decidability is open. We further consider $\mathcal{D}_k \odot \mathcal{D}_1$ reachability where the counter remains linearly bounded. Our third result shows that this bounded variant can be solved in $O(n^2 \cdot \alpha(n))$ time, while our fourth result shows that the problem has a (conditional) quadratic lower bound, and thus our upper bound is essentially optimal. Fifth, we show that full $\mathcal{D}_k \odot \mathcal{D}_k$ reachability is undecidable. This improves the recent NP-hardness lower-bound, and shows that the problem is equivalent to the non-bidirected case. Our experiments on standard benchmarks show that the new algorithms are very fast in practice, offering many orders-of-magnitude speedups over previous methods.

CCS Concepts: • **Software and its engineering → Software verification and validation**; • **Theory of computation** → *Theory and algorithms for application domains*; *Program analysis*.

Additional Key Words and Phrases: static analysis, CFL/Dyck reachability, bidirected graphs, complexity

ACM Reference Format:

Adam Husted Kjelstrøm and Andreas Pavlogiannis. 2022. The Decidability and Complexity of Interleaved Bidirected Dyck Reachability. *Proc. ACM Program. Lang.* 6, POPL, Article 12 (January 2022), 26 pages. https://doi.org/10.1145/3498673

Authors' addresses: Adam Husted Kjelstrøm, Aarhus University, Aabogade 34, Aarhus, 8200, Denmark, au640702@post.au.dk; Andreas Pavlogiannis, Aarhus University, Aabogade 34, Aarhus, 8200, Denmark, pavlogiannis@cs.au.dk.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2022 Copyright held by the owner/author(s).
2475-1421/2022/1-ART12
https://doi.org/10.1145/3498673

```
1  ...
2  void setX(Point p, int v){
3    p.x = v;
4  }
5  int getX(Point r){
6    return r.x;
7  }
8  ...
```

```
9  ...
10 int a,b;
11 Point q;
12 setX(q, a);
13 b=getX(q);
14 ...
```

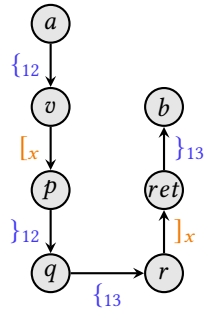

Fig. 1. *(Left):* A program on which to perform context-sensitive and field-sensitive alias analysis. *(Right):* An interleaved Dyck graph where the curly braces (blue) model context sensitivity and the square brackets (orange) model field sensitivity. The path $a \rightsquigarrow b$ produces two interleaved strings, $\{_{12}\}_{12}\{_{13}\}_{13}$ and $[_x]_x$. As both strings are well-balanced, the path is a valid witness and thus b may alias a.

1 INTRODUCTION

Static analyses are one of the most common approaches to program analysis. They offer tunable approximations to program behavior, by representing the set of possible program executions in the semantics of a simpler model (e.g., a graph, or a set of constraints) that is amenable to efficient analysis. Although the model typically over-approximates program behavior, and may thus contain unrealizable executions, it can be used to make sound conclusions of correctness, and even raise warnings of potential erroneous behavior. In essence, this approach casts the semantic question of program correctness to an algorithmic question about the model (e.g., "determine the existence of a path in a graph", or "find the least fixpoint to a system of constraints").

Dyck reachability. The standard formalism of a plethora of static analyses is via language graph reachability [Reps 1997]. Informally the nodes of a graph represent various program segments (e.g., variables), while edges capture relationships between those segments, such as program flow or data dependencies. In order to refine such relationships, the edges are annotated with letters of some alphabet. This formalism reduces the analysis question to a reachability question between nodes in the graph, as witnessed by paths whose labels along the edges produce a string that belongs to a language $\mathcal{L}$. Although $\mathcal{L}$ varies per application, it is almost always a form a visibly pushdown language [Alur and Madhusudan 2004], yielding the problem commonly known as *CFL/Dyck reachability*.

The Dyck reachability problem (denoted $\mathcal{D}_k$) has applications to a very wide range of static analyses, such as interprocedural data-flow analysis [Reps et al. 1995], slicing [Reps et al. 1994], shape analysis [Reps 1995], impact analysis [Arnold 1996], type-based flow analysis [Rehof and Fähndrich 2001], taint analysis [Huang et al. 2015], data-dependence analysis [Tang et al. 2017], alias/points-to analysis [Lhoták and Hendren 2006; Xu et al. 2009; Zheng and Rugina 2008], and many others. In practice, widely-used large-scale analysis tools, such as Wala [Wal 2003] and Soot [Bodden 2012], equip Dyck-reachability techniques to perform the analysis. In all such cases, the balanced-parenthesis property of Dyck languages is ideal for expressing sensitivity of the analysis in matching calling contexts, matching field accesses of composite objects, matching pointer references and dereferences etc. This sensitivity improves precision in the obtained results and drastically reduces false positives.

Interleaved Dyck reachability and variants. When two types of sensitivity are in place simultaneously, the underlying graph problem is lifted to *interleaved* Dyck reachability over two Dyck languages (denoted $\mathcal{D}_k \odot \mathcal{D}_k$). Intuitively, each language acts independently over its own alphabet to ensure sensitivity in one aspect (e.g., context vs field sensitivity). This time, a path is accepted as a reachability witness iff the produced string wrt to each alphabet belongs to the respective language (see Figure 1 for an illustration). Unfortunately, interleaved Dyck reachability is well-known to be undecidable [Reps 2000]. Nevertheless, the importance of the problem has lead to the development of various approximations that often work well in practice. We refer to Section 7 for relevant literature in this area.

When a Dyck language is over a single parenthesis, the underlying stack acts as a counter. This gives rise to variants of interleaved Dyck reachability, depending on whether one ($\mathcal{D}_k \odot \mathcal{D}_1$) or both ($\mathcal{D}_1 \odot \mathcal{D}_1$) languages are over a single parenthesis. Although these are less expressive models, they have rich modeling power and computational appeal. For a $\mathcal{D}_k \odot \mathcal{D}_1$ example, $\mathcal{D}_1$ can express that a queue is dequeued as many times as it is enqueued, and in the right order, while $\mathcal{D}_k$ is used as before to make the analysis context sensitive. Similarly, $\mathcal{D}_1 \odot \mathcal{D}_1$ can be used to identify an execution in which the program properly uses two interleaved but independent reentrant locks.

Bidirected graphs. A Dyck graph is bidirected if every edge is present in both directions, and the two mirrored edges have complementary labels. That is, $a \to b$ opens a parenthesis iff $b \to a$ closes the same parenthesis. Bidirectedness turns reachability into an equivalence relation, much like the case of plain undirected graphs. This happens because every path produces the same string when traversed backwards, and thus either both directions witness reachability or none does. From a semantics perspective, bidirectedness is a standard approach to handle mutable heap data [Lu and Xue 2019; Sridharan and Bodík 2006; Xu et al. 2009; Zhang and Su 2017] – though it can sometimes be relaxed for read-only accesses [Milanova 2020], and the de-facto formulation of demand-driven points-to analyses [Shang et al. 2012; Sridharan et al. 2005; Vedurada and Nandivada 2019; Yan et al. 2011; Zheng and Rugina 2008]. Bidirectedness is also used for CFL-reachability formulations of pointer analysis [Reps 1997], and has also been used for simplifying the input graph [Li et al. 2020]. Thus in all these cases, the analysis yields the problem of *interleaved bidirected Dyck reachability*.

Open questions. Given the many applications of interleaved bidirected Dyck reachability, it is surprising that very little has been known about its decidability and complexity. This gap is even more pronounced if contrasted to the rich literature on the complexity of Dyck/CFL reachability (see Section 7). The recent work of [Li et al. 2021] makes an important step in this direction, by proving an upper bound of $O(n^7)$ for $\mathcal{D}_1 \odot \mathcal{D}_1$, and also showing that the $\mathcal{D}_k \odot \mathcal{D}_k$ case is at least NP-hard. However, most of the fundamental questions remain unanswered, such as the following.

(1) Although the $O(n^7)$ bound is polynomial, it remains prohibitively large. Is there a faster algorithm, e.g., one that operates in quadratic/cubic time that is common in static analyses?
(2) What is the decidability and complexity for $\mathcal{D}_k \odot \mathcal{D}_1$? As $\mathcal{D}_k \odot \mathcal{D}_1$ is more expressive than $\mathcal{D}_1 \odot \mathcal{D}_1$, it leads to better precision in static analyses, and thus it is very well motivated. Moreover, the non-bidirected case is also known as pushdown vector addition systems, where the decidability of reachability has been open (see Section 7). Is bidirectedness sufficient to show decidability?
(3) Given the undecidability of $\mathcal{D}_k \odot \mathcal{D}_k$ on general graphs, does bidirectedness make the problem decidable?

This work delivers several new results on the decidability and complexity of all variants of interleaved bidirected Dyck reachability, and paints the rich landscape of the problem.

Table 1. Summary of our results for interleaved bidirected Dyck reachability.

	$\mathcal{D}_1 \odot \mathcal{D}_1$	$\mathcal{D}_k \odot \mathcal{D}_1$	$\mathcal{D}_k \odot \mathcal{D}_1$ (bounded counter)	$\mathcal{D}_k \odot \mathcal{D}_k$
Upper Bound	$O(n^3 \cdot \alpha(n))$	Decidable	$O(n^2 \cdot \alpha(n))$	-
Lower Bound	-	-	OV-hard	Undecidable

Our contributions. Here we state the main results of this paper, and put them in context with regards to existing literature (see Table 1 for a summary). We consider as input a bidirected interleaved Dyck graph G with n nodes.

Bidirected $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability. We start with the case of bidirected $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability. The non-bidirected case falls into the class of vector addition systems with states. In two dimensions, the problem is known to be in NL (and thus in PTime) [Englert et al. 2016]. The recent work of [Li et al. 2021] established an $O(n^7)$ bound for the bidirected case. As our first theorem shows, the problem admits a must faster solution, namely, in essentially cubic time, which is a common complexity bound in static analyses.

Theorem 1.1. *Bidirected $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability can be computed in $O(n^3 \cdot \alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function.*

We obtain Theorem 1.1 by proving a boundedness property on paths witnessing reachability. In particular, we show that wlog, along every such path *both counters* remain quadratically bounded. This strengthens the shallow-path property of [Li et al. 2021] which states that at any time *one of the two counters* is bounded (though the bound is linear).

We next turn our attention to more expressive variants of interleaved Dyck reachability.

Bidirected $\mathcal{D}_k \odot \mathcal{D}_1$ reachability. Here we address bidirected $\mathcal{D}_k \odot \mathcal{D}_1$ reachability. The non-bidirected case is also known as pushdown vector addition systems in one dimension [Leroux et al. 2015b], for which the decidability of reachability is open. Here we first show that bidirectedness suffices to make the problem decidable.

Theorem 1.2. *Bidirected $\mathcal{D}_k \odot \mathcal{D}_1$ reachability is decidable.*

We next focus on the problem under a form of witness bounding, which is a standard technique for efficient static analyses (e.g., in terms of context bounding [Chatterjee et al. 2017; Shivers 1991], or field-limiting [Deutsch 1994]). In all these cases, the respective bound is not guaranteed a-priori, but the analysis guarantees soundness up to that bound, which is a very useful property. Here, we consider the case where reachability is witnessed by paths on which the counter stays linearly bounded, while the stack has no restrictions. We establish the following theorem.

Theorem 1.3. *Bidirected $\mathcal{D}_k \odot \mathcal{D}_1$ reachability with $O(n)$-bounded counters can be computed in $O(n^2 \cdot \alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function.*

Hence, restricting to a linear bound on the counter makes the problem efficiently solvable. The next natural question is whether this restrictive setting admits even faster algorithms, e.g., can the problem be solved in $O(n)$ time? We answer this question in negative.

Theorem 1.4. *For any fixed $\epsilon > 0$, bidirected $\mathcal{D}_k \odot \mathcal{D}_1$ reachability with $O(n)$-bounded counters cannot be solved in $O(n^{2-\epsilon})$ time under* OV.

Orthogonal Vectors (OV) is a well-studied problem with a long-standing quadratic worst-case upper bound. The corresponding hypothesis states that there is no sub-quadratic algorithm for the problem [Williams 2019]. It is also known that the strong exponential time hypothesis (SETH) implies the Orthogonal Vectors hypothesis [Williams 2005]. Thus, under Theorem 1.4, the upper-bound of Theorem 1.3 is (nearly) optimal.

Bidirected $\mathcal{D}_k \odot \mathcal{D}_k$ reachability. Finally, we turn our attention to the general case of bidirected $\mathcal{D}_k \odot \mathcal{D}_k$ reachability. The non-bidirected case is well-known to be undecidable [Reps 2000], while it was recently shown [Li et al. 2021] that the bidirected case is NP-hard, leaving its decidability open. The following theorem resolves this open question.

Theorem 1.5. *Bidirected $\mathcal{D}_k \odot \mathcal{D}_k$ reachability is undecidable.*

In terms of practical implications, Theorem 1.5 establishes the undecidability of popular pointer and alias analyses as those in [Shang et al. 2012; Sridharan et al. 2005; Vedurada and Nandivada 2019; Yan et al. 2011; Zheng and Rugina 2008].

Experimental results. We have implemented our algorithms for $\mathcal{D}_1 \odot \mathcal{D}_1$ and $\mathcal{D}_k \odot \mathcal{D}_1$ reachability (Theorem 1.1 and Theorem 1.3, respectively), and have evaluated them on standard benchmarks on a conventional laptop. Each algorithm handles the whole benchmark set in less than 5 minutes. On the other hand, the previous algorithm of [Li et al. 2021] for $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability was reported to handle the same benchmark set in more than 3 days when run on a bigger machine. Thus, besides the theoretical improvements, our new algorithms confer a significant practical speedup, and are the first ones ready to be used in practice.

In the following sections we develop relevant notation and present details of the above theorems. To improve readability, in the main paper we present algorithms, examples, and proofs of all theorems. To highlight the main steps of the proofs, we also present all intermediate lemmas; many lemma proofs, however, are relegated to a technical report [Kjelstrøm and Pavlogiannis 2021]

2 PRELIMINARIES

General notation. Given a natural number $k \in \mathbb{N}$, we denote by $[k]$ the set $\{1, \ldots, k\}$. Given a sequence of elements $\sigma = e_1, \ldots, e_\ell$, we denote by $|\sigma| = \ell$ the length of σ, and denote by $\mathrm{Pref}(\sigma)$ the set of prefixes of σ, i.e., $\mathrm{Pref}(\sigma) = \{e_1, \ldots, e_{\ell'} : \ell' \in [\ell]\}$. Given two sequences σ_1, σ_2, we denote by $\sigma_1 \circ \sigma_2$ their concatenation. Finally, we lift concatenation to sets of sequences, i.e., for two sets of sequences L_1, L_2, we have $L_1 \circ L_2 = \{\sigma_1 \circ \sigma_2 : \sigma_i \in L_i \text{ for each } i \in [2]\}$.

Dyck Languages. Given a natural number $k \in \mathbb{N}$, a Dyck alphabet $\Sigma = \{\alpha_i, \overline{\alpha}_i\}_{i \in [k]}$ is a finite *alphabet* of k parenthesis symbols, where $\mathrm{Open}(\Sigma) = \{\alpha_i\}_{i \in k}$ and $\mathrm{Close}(\Sigma) = \{\overline{\alpha}_i\}_{i \in k}$ are the sets of open parenthesis symbols and close parenthesis symbols of Σ, respectively[1]. We denote by $\mathcal{D}(\Sigma)$ the Dyck language over Σ, defined as the language of strings generated by the following context-free grammar $\mathcal{G}$:

$$\mathcal{S} \to \mathcal{S}\,\mathcal{S} \mid \mathcal{A}_1\,\overline{\mathcal{A}}_1 \mid \ldots \mid \mathcal{A}_k\,\overline{\mathcal{A}}_k \mid \epsilon\,; \qquad \mathcal{A}_i \to \alpha_i\,\mathcal{S}\,; \qquad \overline{\mathcal{A}}_i \to \overline{\alpha}_i$$

[1]For more clarity in our presentation, we use Greek letters such as α, β to represent opening parenthesis symbols, and $\overline{\alpha}, \overline{\beta}$ to represent their matching closing parentheses.

Given a string σ and a non-terminal symbol X of $\mathcal{G}$, we write $X \vdash s$ to denote that X produces s. The Dyck language over Σ is then defined as $\mathcal{D}(\Sigma) = \{\sigma \in \Sigma^* \colon \mathcal{S} \vdash \sigma\}$. For example, $\alpha_1\alpha_2\overline{\alpha}_2\alpha_3\overline{\alpha}_3\overline{\alpha}_1 \in \mathcal{D}(\Sigma)$, but $\alpha_1\alpha_2\alpha_3\overline{\alpha}_3\overline{\alpha}_1\overline{\alpha}_2 \notin \mathcal{D}(\Sigma)$. We typically ignore the alphabet Σ and write $\mathcal{D}_k$ for the Dyck language over some implicit alphabet with k different parenthesis symbols. Finally, given a string $\sigma = \gamma_1 \dots \gamma_m \in \Sigma^*$, we let $\overline{\sigma} = \overline{\gamma}_1 \dots \overline{\gamma}_m \in \Sigma^*$, where $\overline{\gamma}_i$ is a closing parenthesis symbol if γ_i is an opening parenthesis symbol, and vice versa. For example, if $\sigma = \alpha_1\alpha_2\overline{\alpha}_2\alpha_3\overline{\alpha}_3\overline{\alpha}_1$, then $\overline{\sigma} = \overline{\alpha}_1\overline{\alpha}_2\alpha_2\overline{\alpha}_3\alpha_3\alpha_1$.

Interleaved Dyck languages. Given natural numbers $k_1, k_2 \in \mathbb{N}$, consider two alphabets $\Sigma_1 = \{\alpha_i, \overline{\alpha}_i\}_{i=1}^{k_1}$ and $\Sigma_2 = \{\beta_i, \overline{\beta}_i\}_{i=1}^{k_2}$ with $\Sigma_1 \cap \Sigma_2 = \emptyset$, i.e., the two alphabets are disjoint. Given some word $w \in (\Sigma_1 \cup \Sigma_2)^*$, we denote by $w \downharpoonright \Sigma$ the projection of w on the alphabet $\Sigma \in \{\Sigma_1, \Sigma_2\}$. The interleaved Dyck language over the alphabet pair (Σ_1, Σ_2) is defined as

$$\mathcal{D}(\Sigma_1) \odot \mathcal{D}(\Sigma_2) = \{\sigma \in (\Sigma_1 \cup \Sigma_2)^* \colon \sigma \downharpoonright \Sigma_1 \in \mathcal{D}(\Sigma_1) \text{ and } \sigma \downharpoonright \Sigma_2 \in \mathcal{D}(\Sigma_2)\}$$

For example, we have $\alpha_1\beta_1\overline{\alpha}_1\overline{\beta}_1 \in \mathcal{D}(\Sigma_1) \odot \mathcal{D}(\Sigma_2)$. Similarly as above, we typically ignore the alphabets Σ_1 and Σ_2, and write $\mathcal{D}_{k_1} \odot \mathcal{D}_{k_2}$ for the interleaved Dyck language over two implicit alphabets of sizes k_1 and k_2, with the understanding that the alphabets are disjoint.

Graphs and language reachability. We consider labeled directed graphs $G = (V, E, \Sigma)$ where V is the set of nodes, Σ is an alphabet, and $E \subseteq V \times V \times (\Sigma \cup \{\epsilon\})$ is a set of edges (partially) labeled with letters from Σ. Given an edge $(u, v, \alpha) \in E$, we denote by $\lambda(u, v, \alpha) = \alpha$ the label of the edge, and often write $u \xrightarrow{\alpha} v$ to denote the existence of such an edge. A path $P = e_1, \dots, e_\ell$ is a sequence of edges. The label of P is $\lambda(P) = \lambda(e_1) \dots \lambda(e_\ell)$, i.e., it is the concatenation of the labels of the edges along P. We often write $P \colon u \rightsquigarrow v$ to denote a path from node u to node v. Naturally, we say that v is reachable from u if such a path exists. Given some language $\mathcal{L} \subseteq \Sigma^*$, we say that v is $\mathcal{L}$-reachable from u if there exists a path $P \colon u \rightsquigarrow v$ such that $\lambda(P) \in \mathcal{L}$, in which case we write $P \colon u \overset{\mathcal{L}}{\rightsquigarrow} v$.

Notation on paths and cycles. A (simple) cycle is a path $C = e_{i_1}, e_{i_{i+1}}, \dots, e_{i_j}$ such that the left endpoint of e_{i_1} coincides with the right endpoint of e_{i_j} and no other node repeats in C. A cyclic path is a path $P \colon u \rightsquigarrow u$ (though P may also repeat nodes other than u, and thus not be a cycle). Given a cyclic sub-path P' of P, we denote by $P \setminus P'$ the path we obtain after removing P' from P.

Dyck graphs. A labeled graph $G = (V, E, \Sigma)$ is a Dyck graph if Σ is a Dyck alphabet. Dyck reachability in G is defined with respect to the corresponding Dyck language $\mathcal{D}(\Sigma)$. A Dyck path $P \colon u \rightsquigarrow v$ is a path such that $\lambda(P) \in \mathcal{D}(\Sigma)$, i.e., P witnesses the reachability of v from u wrt the Dyck language $\mathcal{D}_k$.

Interleaved Dyck graphs. An interleaved Dyck graph is a Dyck graph $G = (V, E, \Sigma)$ where Σ is implicitly partitioned into two disjoint Dyck alphabets $\Sigma = \Sigma_1 \uplus \Sigma_2$. Interleaved Dyck reachability in G is defined wrt the respective interleaved Dyck language $\mathcal{D}(\Sigma_1) \odot \mathcal{D}(\Sigma_2)$. That is, a path $P \colon u \rightsquigarrow v$ witnesses the reachability of v from u in G iff $\lambda(P) \downharpoonright \Sigma_i \in \mathcal{D}(\Sigma_i)$ for each $i \in [2]$. Given some $i \in [2]$, we will write $\mathrm{Stk}_i(P)$ to denote content of the i-th stack at the end of P, with the natural interpretation that open parenthesis symbols push on the stack and close parenthesis symbols pop from the stack. Moreover we use $\mathrm{SH}_i(P)$ and $\mathrm{MaxSH}_i(P)$ to denote the stack height and maximum stack height of P wrt alphabet Σ_i. Formally,

$$\mathrm{SH}_i(P) = |\lambda(P) \downharpoonright \mathrm{Open}(\Sigma_i)| - |\lambda(P) \downharpoonright \mathrm{Close}(\Sigma_i)| \qquad \text{and} \qquad \mathrm{MaxSH}_i(P) = \max_{P' \in \mathrm{Pref}(P)} \mathrm{SH}_i(P')$$

Fig. 2. *(Left):* An interleaved Dyck graph. We have $u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$, via the path $u \xrightarrow{\alpha_1} u \xrightarrow{\beta_1} x \xrightarrow{\beta_2} x \xrightarrow{\alpha_2} y \xrightarrow{\overline{\beta}_2} x \xrightarrow{\overline{\alpha}_2} v \xrightarrow{\overline{\alpha}_1} y \xrightarrow{\overline{\beta}_1} v$. *(Right):* A bidirected Dyck graph. We omit the ϵ label from the edges.

Figure 2 shows an interleaved Dyck graph. We will often project G to some alphabet Σ_i, for some $i \in [2]$, in which case we obtain a (non-interleaved) Dyck graph. That is, the projection $G \downharpoonright \Sigma_i$ is identical to G where every edge label of Σ_{3-i} is replaced by ϵ.

Special cases. We obtain two special cases of interleaved Dyck graphs when one, or both Dyck alphabets are binary (i.e., consisting of one opening-parenthesis symbol and the corresponding closing-parenthesis symbol). When a Dyck alphabet is binary, the corresponding stack behaves as a counter. To make this case explicit, if $|Alphabet_i| = 2$ for some $i \in [2]$, we refer to the stack as a counter, and given a path P, we write $\mathrm{Cnt}_i = \mathrm{SH}_i(P)$ and $\mathrm{MaxCnt}_i(P) = \mathrm{MaxSH}_i(P)$.

Bidirected graphs. Consider a labeled graph $G = (V, E, \Sigma)$ where Σ is a Dyck alphabet. We call G bidirected if it satisfies the following condition, where we take $\overline{\epsilon} = \epsilon$.

$$\forall u, v \in V, \alpha \in \{\epsilon\} \cup \Sigma \colon (u, v, \alpha) \in E \text{ iff } (v, u, \overline{\alpha}) \in E$$

Bidirected graphs can be seen as a natural lifting of undirected graphs to the labeled setting where reachability is expressed wrt a Dyck language. Indeed, for every path $P \colon u \rightsquigarrow v$, the reverse path $\overline{P}$ satisfies $\lambda(\overline{P}) = \lambda(P)$. Thus, Dyck reachability is an equivalence relation on bidirected graphs, much like plain reachability on undirected graphs. Observe that the same holds when Σ is the disjoint union of two Dyck alphabets, and G is an interleaved Dyck graph. Figure 2 shows a bidirected Dyck graph. As we mostly deal with bidirected graphs in this paper, we will define them and depict them with every edge appearing in only one direction, with the inverse direction and label taken implicitly.

Irreducible paths. We now introduce the notion of irreducible paths, which is helpful in analyzing bidirected graphs. A path $P \colon u \rightsquigarrow v$ is *reducible*, if there exist $j_1 < j_2$ such that (i) $P[j_1 \colon j_2]$ is a cyclic sub-path of P, and (ii) $\mathrm{Stk}_i(P[\colon j_1]) = \mathrm{Stk}_i(P[\colon j_2])$ for each $i \in [2]$. In this case, we can simplify P by the path $P' = P \setminus P[j_1 \colon j_2]$, as $P' \colon u \rightsquigarrow v$ is a valid path and each stack has the same contents at the end of P' as at the end of P. Finally, we call P *irreducible* if it is not reducible. Without loss of generality, throughout this work we consider that paths witnessing reachability are irreducible paths.

Interleaved Dyck reachability problems. In this work we study interleaved Dyck reachability problems on bidirected Dyck graphs G, over a corresponding interleaved Dyck language $\mathcal{D}_{k_1} \odot \mathcal{D}_{k_2}$. As standard in the literature, we take that k_1, k_2 are constant and independent of G. We will distinguish cases when k_1 and k_2 are unrestricted, as well as special cases when $k_1 = 1$ and k_2 is

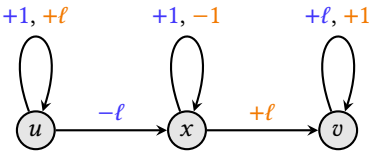

Fig. 3. A simple graph where every $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ path has both counters reaching $\Omega(n^2)$. Edges indicate paths with the respective effect on the counters. In every such path each counter reaches value ℓ^2, where we take $\ell = (n-3)/4 = \Theta(n)$.

unrestricted, and when $k_1 = k_2 = 1$. In the unrestricted case, (i.e., when $k_1, k_2 > 1$), for ease of presentation we assume that $k_1 = k_2 = k$, and write $\mathcal{D}_k \odot \mathcal{D}_k$.

Remark 1 (Sparsity of G)**.** *Without loss of generality, we assume that G is sparse. i.e., $|E| = O(|V|)$. Indeed, if any node x has two outgoing edges with the same closing label $x \xrightarrow{\overline{\alpha}} y$ and $x \xrightarrow{\overline{\alpha}} z$, then $z \xrightarrow{\alpha} x \xrightarrow{\overline{\alpha}} y$ and hence y and z are reachable from each other and thus can be merged. By applying this merging process repeatedly, we arrive at a graph that is sparse, as every node has a bounded number of outgoing edges. The total time of this process is nearly linear [Chatterjee et al. 2018].*

3 A FAST ALGORITHM FOR $\mathcal{D}_1 \odot \mathcal{D}_1$ REACHABILITY

In this section we deal with $\mathcal{D}_1 \odot \mathcal{D}_1$, i.e., when both Dyck languages are over a unary alphabet.

Shallow paths. The key insight behind the algorithm of [Li et al. 2021] is reachable nodes exhibit a "shallow-paths" property, which states that wlog, along every witness path one of the two counters is linearly bounded. We restate the property here.

Lemma 3.1 (Shallow Paths [Li et al. 2021]). *For any nodes $u, v \in V$ such that v is $\mathcal{D}_1 \odot \mathcal{D}_1$-reachable from u, there exists a path $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ such that for every prefix $P' \in \mathrm{Pref}(P)$, we have $\mathrm{Cnt}_i(P') \leq 6 \cdot n$ for some $i \in [2]$.*

We call any path P that satisfies the conditions of Lemma 3.1 a *shallow path*. The shallow-path property was exploited in [Li et al. 2021] to compute $\mathcal{D}_1 \odot \mathcal{D}_1$-reachability in $O(n^7)$ time. The key insight behind our faster algorithm is a stronger property that we call *bounded paths*.

Bounded paths. The bounded-paths property states that wlog, each counter is *always* bounded by $O(n^2)$ along any witness path. Formally, we have the following lemma.

Lemma 3.2 (Bounded Paths). *For any nodes $u, v \in V$ such that v is $\mathcal{D}_1 \odot \mathcal{D}_1$-reachable from u, there exists a witness path $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ such that $\mathrm{MaxCnt}_i(P) \leq 18 \cdot n^2 + 6 \cdot n$ for each $i \in [2]$, where n is the number of nodes in G.*

We remark that the $O(n^2)$ bound of Lemma 3.2 is tight: Figure 3 shows a simple example where in every $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ path, each counter reaches $\Theta(n^2)$. In the following, we first present an efficient algorithm for $D_1 \odot D_1$ based on Lemma 3.2, and then prove the lemma.

3.1 Algorithm for $\mathcal{D}_1 \odot \mathcal{D}_1$ Reachability

The bounded-paths property implies a straightforward algorithm to solve $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability. Let c be the counter bound of the bounded-paths lemma. We flatten the graph on one counter, by associating each node with all possible values of that counter up to this bound, while replacing all edge labels of that counter with 0 (i.e., no effect). This transformation reduces bidirected $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability to bidirected $\mathcal{D}_1$ reachability, which is solved in almost linear-time on bidirected graphs [Chatterjee et al. 2018]. See Algorithm 1 for details.

Algorithm 1: Algo-$\mathcal{D}_1 \odot \mathcal{D}_1$

Input: A bidirected $\mathcal{D}_1 \odot \mathcal{D}_1$ graph $G = (V, E, \Sigma = \Sigma_1 \uplus \Sigma_2)$ of n nodes
Output: All-pairs $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability in G

1 Let $c \leftarrow 18 \cdot n^2 + 6 \cdot n$
2 Construct the bidirected $\mathcal{D}_1$ graph $G' = (V', E', \Sigma')$ as follows
  (1) The node set is $V' = V \times (\{0\} \cup [c])$
  (2) The edge set is such that, for every edge $(u, v, \gamma) \in E$ and $j \in \{0\} \cup [c]$, we have an edge $((u, j_1), (u, j_2), \delta) \in E'$, as follows
3 (a) If $\gamma \in \Sigma_1 \cup \{\epsilon\}$, then $j_2 = j_1$ and $\delta = \gamma$
  (b) If $\gamma \in \mathrm{Open}(\Sigma_2)$ and $j_1 < c$, then $j_2 = j_1 + 1$ and $\delta = \epsilon$
  (c) If $\gamma \in \mathrm{Close}(\Sigma_2)$ and $j_1 > 0$, then $j_2 = j_1 - 1$ and $\delta = \epsilon$
4 Solve all-pairs $\mathcal{D}_k$ reachability on G' using [Chatterjee et al. 2018]
5 **return** *that* $u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ *in* G *iff* $(u, 0) \overset{\mathcal{D}_1}{\rightsquigarrow} (v, 0)$ *in* G'

Proof of Theorem 1.1. We start with the correctness. For any path $P\colon u \rightsquigarrow v$ in G with $\mathrm{MaxCnt}_1(P) \leq c$, there exists a path $P'\colon (u, 0) \rightsquigarrow (v, \ell)$ in G', such that $\ell = \mathrm{Cnt}_1(P)$ and $\mathrm{Cnt}_2(P) = \mathrm{Cnt}_2(P')$. By Lemma 3.2, if a node v is $\mathcal{D}_1 \odot \mathcal{D}_1$-reachable from a node u in G, then there exists a witness path P with $\mathrm{MaxCnt}_j(P) \leq c$ for each $j \in [2]$. Thus there exists a corresponding path $P'\colon (u, 0) \rightsquigarrow (v, 0)$ in G', which means that $(u, 0)$ and $(v, 0)$ are in the same reachability class of G'. The correctness hence follows from the correctness of the algorithm of [Chatterjee et al. 2018] for $\mathcal{D}_1$-reachability.

Regarding the complexity, by Lemma 3.2, the graph G' has $O(n \cdot c)$ nodes. By Remark 1, the graph G is sparse, which implies that G' is also sparse, i.e., G' has $O(n \cdot c)$ edges. By [Chatterjee et al. 2018], solving $\mathcal{D}_1$-reachability on G' requires $O(n \cdot c \cdot \alpha(n)) = O(n^3 \cdot \alpha(n))$ time. The desired result follows. □

3.2 Bounded Paths in $\mathcal{D}_1 \odot \mathcal{D}_1$ Reachability

We now turn our attention to the proof of the bounded paths lemma. This section is somewhat technical and can be be skipped at first, as later sections do not depend on it. We remark that our main goal is to show a quadratic upper-bound on each counter along a path that witnesses $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability. As such, our proof aims at readability, without necessarily establishing the smallest constant factor in this bound.

Counter indexes. Given a path P, an integer $i \in [2]$ and a natural number $c \in \mathbb{N}$, we define the *counter indexes* of counter i of P on counter value c as the set of indexes of P in which P attains value c on counter i. Formally, we have

$$\mathrm{CntInd}_i^c(P) = \{j \in \{0\} \cup [|P|] \colon \mathrm{Cnt}_i(P[:j]) = c\}$$

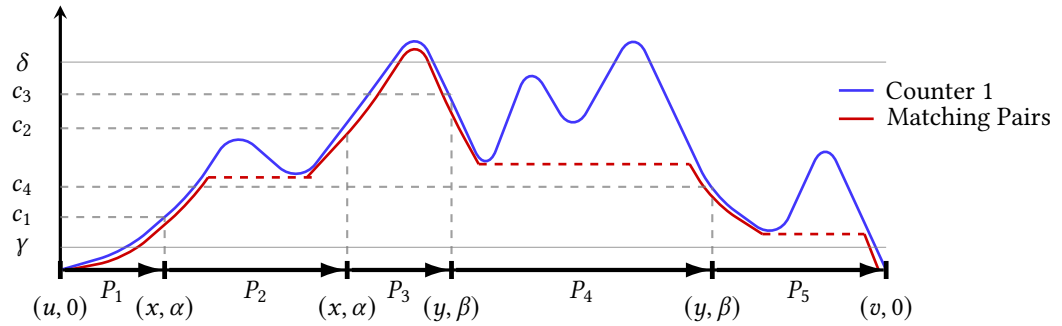

Fig. 4. Illustration of Lemma 3.3. The x-axis shows the decomposition of the path $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ into paths $P = P_1 \circ P_2 \circ P_3 \circ P_4 \circ P_5$. The pairs (p, q) show the node p and the value q of the second counter along various segments of P. The y-axis shows the value of the first counter along P (blue) as well as its value on the matching pairs of P (red).

To make our exposition simpler, we focus on the boundedness property of Lemma 3.2 on the first counter. The case of the second counter is completely symmetric, while it will become clear in the final step of the proof that both properties can be guaranteed simultaneously (i.e., both counters satisfying the bound of Lemma 3.2).

Matching pairs. Consider a path $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ in G, and let $i_{\max}$ be the first point where P attains its maximum value on the first counter. That is, let $i_{max} = \min(\mathrm{CntInd}_1^a(P))$, where $a = \mathrm{MaxCnt}_1(P)$. For every $c \in \{0\} \cup [\mathrm{MaxCnt}_1(P)]$, we define the indexes

$$l_c = \max(\{i\colon i \leq i_{\max} \text{ and } \mathrm{Cnt}_1(P[:i]) = c\}) \text{ and } r_c = \min(\{i\colon i \geq i_{\max} \text{ and } \mathrm{Cnt}_1(P[:i]) = c\})$$

i.e., l_c (resp., r_c) is the last index before $i_{\max}$ (resp., the first index after $i_{\max}$) in which the first counter of P has value c. Let x_c (resp., y_c) be the last node of $P[:l_c]$ (resp., $P[:r_c]$), and we call (x_c, y_c) a *matching pair*. The following lemma states that if a shallow path P reaches a large enough value on the first counter, then "on its way up" it must go through the same pair (p, q) twice, where p is a node and q is the value of the second counter, while the same holds "on its way down". It is a straightforward application of the pigeonhole principle (see Figure 4 for an illustration).

Lemma 3.3. *Let $\gamma = 12 \cdot n^2 + 6 \cdot n$ and $\delta = 18 \cdot n^2 + 6 \cdot n + 1$. Consider any shallow path $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ such that $\mathrm{MaxCnt}_1(P) \geq \delta$. Then there exist matching pairs $(x_{c_j}, y_{c_j})_{1 \leq j \leq 4}$ such that the following hold.*

(1) $\gamma \leq c_1 < c_2 \leq \delta$ and $\gamma \leq c_4 < c_3 \leq \delta$.
(2) $x_{c_1} = x_{c_2}$ and $y_{c_3} = y_{c_4}$.
(3) $\mathrm{Cnt}_2(P[:l_{c_1}]) = \mathrm{Cnt}_2(P[:l_{c_2}])$ and $\mathrm{Cnt}_2(P[:r_{c_3}]) = \mathrm{Cnt}_2(P[:r_{c_4}])$.

Path deflation. Using Lemma 3.3 we describe a process we call *path deflation*. Informally, it states that if the first counter exceeds δ in P, we can construct a path Q in which the second counter stays the same as in P, but the first counter reaches the global maximum of P one less time in Q than in P (hence the path has been deflated). In particular, we construct Q as

$$Q = P_1 \circ P_3 \circ P_4 \circ \overline{P}_3 \circ P_2 \circ P_3 \circ P_5$$

where the various sub-paths P_i are defined based on Lemma 3.3 and are also illustrated in Figure 4. In more detail, the first counter reaches its global maximum in P for the first time while traversing the sub-path P_3, while P_2 starts and ends in the same node x with the second counter having the same value α, and P_4 starts and ends in the same node y with the second counter having the same value β. Thus, we can skip P_2 and instead reach and traverse P_4, without affecting the second counter. Skipping P_2 avoids increasing the first counter, while traversing P_4 decreases the first counter. Traversing $\overline{P}_3$ (i.e., traversing P_3 backwards) brings us back to node x with the second counter having the same value α. At this point we can traverse P_2 and P_3 and then proceed with P_5, i.e., proceed as in P but skip the already traversed sub-path P_4. This rearrangement has the effect that none of the traversals of P_3 in Q reaches the global maximum of P. Indeed, in the first traversal the counter has decreased by $c_2 - c_1 > 0$; in the second traversal it has decreased by $c_2 - c_1 + c_3 - c_4 > 0$; and in the third traversal it has decreased by a $c_3 - c_4 > 0$. Moreover, as $c_1, c_4 \geq \gamma$ while $c_2 - c_1 < \gamma/2$ and $c_3 - c_4 < \gamma/2$, while the first counter has decreased, it remains non-negative in these traversals, and thus Q is a valid path. The following lemma states this property formally.

Lemma 3.4. *Let $\delta = 18 \cdot n^2 + 6 \cdot n + 1$. Assume that there exists a shallow path $P \colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ such that $\mathrm{MaxCnt}_1(P) \geq \delta$. Then there exists a shallow path $Q \colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ such that the following hold.*

(1) $\mathrm{MaxCnt}_2(Q) = \mathrm{MaxCnt}_2(P)$.
(2) $|\mathrm{CntInd}_1^c(Q)| < |\mathrm{CntInd}_1^c(P)|$, *where* $c = \mathrm{MaxCnt}_1(P)$.

Note that if the path Q of Lemma 3.4 also has $\mathrm{MaxCnt}_1(Q) \geq \delta$, we can apply the lemma recursively on Q. This recursive process is guaranteed to terminate as the new path that comes out of the lemma either has a smaller global maximum on the first counter, or has the same global maximum but this maximum is attained one time less. Hence we have a finite number of applications of the lemma. Given this observation, we are now ready to conclude the proof of the bounded-paths lemma.

Proof of Lemma 3.2. Consider that there is a path $P \colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$. Due to Lemma 3.1, we may assume wlog that P is a shallow path. If $\mathrm{MaxCnt}_1(P) \leq 18 \cdot n^2 + 6 \cdot n$, we proceed with the second counter. Otherwise, we apply repeatedly Lemma 3.4 until we arrive at a path Q' with $\mathrm{MaxCnt}_2(Q') = \mathrm{MaxCnt}_2(P)$ and $\mathrm{MaxCnt}_1(Q') \leq 18 \cdot n^2 + 6 \cdot n$. Note that Q' remains a shallow path throughout this process. Finally, we follow the same process for Q' instead of P, and with the two counters swapped. In the end, we arrive at a path Q with $\mathrm{MaxCnt}_i(Q) \leq 18 \cdot n^2 + 6 \cdot n$. The desired result follows. □

4 UPPER AND LOWER BOUNDS FOR $\mathcal{D}_k \odot \mathcal{D}_1$ REACHABILITY

In this section we address $\mathcal{D}_k \odot \mathcal{D}_1$ reachability, i.e., where one of the two stacks behaves as a counter. In Section 4.1 we prove the decidability of the problem (Theorem 1.2) In Section 4.2 we turn our attention to the counter-bounded version of the problem, and prove Theorem 1.3 and Theorem 1.4.

4.1 Decidability of $\mathcal{D}_k \odot \mathcal{D}_1$ Reachability

Consider an interleaved bidirected Dyck graph $G = (V, E, \Sigma = \Sigma_k \uplus \Sigma_1)$ and two nodes $u, v \in V$. We show that, deciding whether there is a path $P \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} v$ is decidable. Our decidability proof makes use of a relaxed notion of reachability in vector addition systems, known as coverability.

Coverability. Consider two nodes $u, v \in V$ and a non-negative integer $c \in \mathbb{N}$. The node u *covers* (v, c) if there is a valid path $P\colon u \rightsquigarrow v$ with $\mathrm{Stk}(P) = \epsilon$ and $\mathrm{Cnt}(P) \geq c$. In other words, we can reach v from u via a path that is balanced wrt the stack, but the counter might be positive (instead of zero). The coverability problem is to decide whether u covers (v, c), and is known to be decidable even on non-bidirected interleaved Dyck graphs (phrased in the language of pushdown vector addition systems [Leroux et al. 2015b]). We start with a straightforward lemma.

Lemma 4.1. *If* $u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ *then* u *covers* $(v, 0)$ *and* v *covers* $(u, 0)$.

Intuitive description. Lemma 4.1 states a necessary condition for the reachability of v from u. Hence, as a first step, in order to decide reachability, we may verify that u covers $(v, 0)$ and v covers $(u, 0)$ using the procedure of [Leroux et al. 2015b]. Next, we can similarly check if u covers $(v, 1)$ and v covers $(u, 1)$. Clearly, if u does not cover $(v, 1)$, since u covers $(v, 0)$, we have that v is reachable from u and we are done. We arrive at a similar conclusion if v does not cover $(u, 1)$. But what if u covers $(v, 1)$ and v covers $(u, 1)$? The key insight is that, using the paths $P_u\colon u \rightsquigarrow v$ and $P_v\colon v \rightsquigarrow u$ that witness the corresponding coverability relationships, we can derive a bound on the length of the shortest path $L\colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ that may witness reachability. Algorithm 2 presents this algorithm.

Algorithm 2: Algo-$\mathcal{D}_k \odot \mathcal{D}_1$

Input: An interleaved bidirected Dyck graph $G = (V, E, \Sigma = \Sigma_2 \uplus \Sigma_1)$, two nodes $u, v \in V$
Output: True iff $u \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} v$

```
if u does not cover (v, 0) or v does not cover (u, 0) then
    return False
if u does not cover (v, 1) or v does not cover (u, 1) then
    return True
Let P_u ← a witness path for the coverability of (v, 1) from u
Let P_v ← a witness path for the coverability of (u, 1) from v
Let ζ ← max(MaxSH(P_u), MaxSH(P_v))
Let δ ← max(ζ, 2 · n^2)
foreach path L: u ⇝ v with |P| ≤ n^3 · k^{3·δ} do
    if L witnesses u ⇝^{D_k ⊙ D_k} v then
        return True
end
return False
```

The key idea behind the bounded length of L stems from the fact that, by traversing the cyclic path $P_u \circ P_v$ repeatedly, we can increase the value of the counter arbitrarily without increasing the maximum stack height. Now, starting from u with a large enough counter and an empty stack, we can reach v with the same counter value and an empty stack, while the maximum stack height of this $u \rightsquigarrow v$ path is bounded. Now repeating the process symmetrically from v, we end up with a path $T\colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ in which the stack height remains bounded. The final step is to show that wlog, any path that has bounded stack height also has bounded length, thus arriving at the path L. As there are finitely many paths of bounded length, the decidability follows. We use the remaining of this section to develop this intuition precisely.

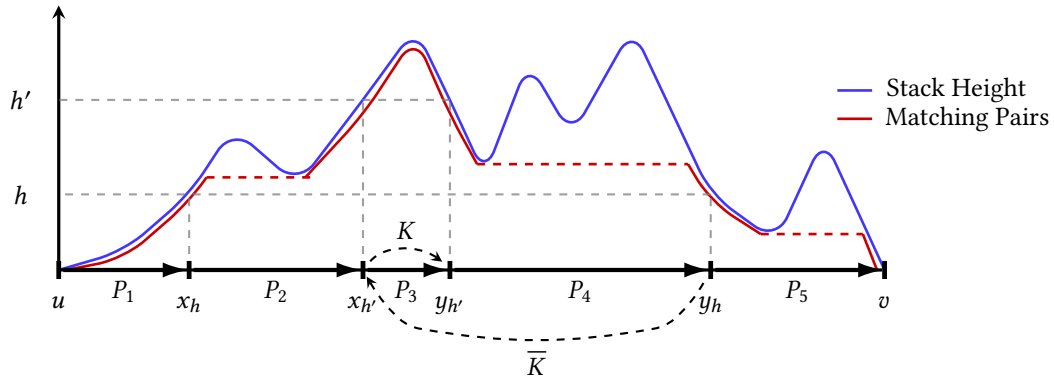

Fig. 5. Illustration of the path rearrangement behind Lemma 4.3.

Matching pairs. We revisit the notion of matching pairs from Section 3.2, and adapt it to our current setting where we have a stack instead of a counter. In particular, consider a path $P\colon u \overset{\mathcal{D}_k}{\rightsquigarrow} v$ in $G \downarrow \Sigma_k$, and let $i_{\max}$ be the first point where P attains its maximum stack height. For every $h \in [\mathrm{MaxSH}(P)]$, let

$$l_h = \max(\{i\colon i \le i_{\max} \text{ and } \mathrm{SH}(P[:i]) = h\}) \quad \text{and} \quad r_h = \min(\{i\colon i \ge i_{\max} \text{ and } \mathrm{SH}(P[:i]) = h\})$$

i.e., l_h (resp., r_h) is the last index before $i_{\max}$ (resp., the first index after $i_{\max}$) in which the stack of P has size h. Let x_h (resp., y_h) be the last node of $P[:l_h]$ (resp., $P[:r_h]$), and we call (x_h, y_h) a *matching pair*. We have the following straightforward lemma.

Lemma 4.2. *Consider a path $P\colon u \overset{\mathcal{D}_k}{\rightsquigarrow} v$ in $G \downarrow \Sigma_k$. If $\mathrm{MaxSH}(P) \ge n^2$, then P has two matching pairs (x_h, y_h) and $(x_{h'}, y_{h'})$ such that (i) $h < h' \le n^2$, (ii) $x_h = x_{h'}$, and (iii) $y_h = y_{h'}$.*

Based on Lemma 4.2 we prove the following lemma.

Lemma 4.3. *Consider a path $P\colon u \overset{\mathcal{D}_k}{\rightsquigarrow} v$ in $G \downarrow \Sigma_k$. There is a path $Q\colon u \overset{\mathcal{D}_k}{\rightsquigarrow} v$ in $G \downarrow \Sigma_k$ such that (i) $\mathrm{MaxSH}(Q) \le 2 \cdot n^2$, and (ii) $\mathrm{Cnt}(Q) = \mathrm{Cnt}(P)$.*

The key idea behind Lemma 4.3 is as follows. Assume that $\mathrm{MaxSH}(P) \ge 2 \cdot n^2$. We apply Lemma 4.2 to obtain two matching pairs (x_h, y_h) and $(x_{h'}, y_{h'})$. We decompose P as $P = P_1 \circ P_2 \circ P_3 \circ P_4 \circ P_5$ as follows (see Figure 5).

(1) P_1 is the prefix of P up to x_h.
(2) P_2 is the sub-path $x_h \rightsquigarrow x_{h'}$ of P.
(3) P_3 is the sub-path $x_{h'} \rightsquigarrow y_{h'}$ of P.
(4) P_4 is the sub-path $y_{h'} \rightsquigarrow y_h$ of P.
(5) P_5 is the suffix of P from x_h on.

Moreover, we let K be a shortest path $K\colon x_h \overset{\mathcal{D}_k}{\rightsquigarrow} y_h$, and it is known that $\mathrm{MaxSH}(K) \le n^2$ [Pierre 1992]. We rearrange P to obtain the path $R = P_1 \circ P_2 \circ K \circ P_4 \circ \overline{K} \circ P_3 \circ P_5$. Now R does not reach the maximum stack height that P reaches while traversing P_3, as, in the beginning of P_3, the stack height has decreased from h' to h. Moreover, all the way up to P_3, the stack height of R is (strictly) below $h' + n^2 \le 2 \cdot n^2$. Finally, since R traverses every edge of P exactly once, the two paths executed

in G have the same counter value at the end. Lemma 4.3 is obtained by repeated applications this process until we end up with a path Q as stated in the lemma.

Note that the above process only concerns the stack height. Indeed, if we consider the counter along Q, then Q is not necessarily a valid path as the counter may become negative because of the repeated rearrangements. Assume, however, that we also have a cyclic path $F_u \colon u \rightsquigarrow u$ that ends with an empty stack and increases the counter by a positive amount. Then we may prefix Q by iterating F_u a number of ℓ_u times until the counter becomes large enough to stay non-negative along Q. Now the corresponding path $F_u^{\ell_u} \circ Q$ is a valid path, but no longer witnesses the $\mathcal{D}_k \odot \mathcal{D}_1$ reachability of v from u, as the counter at the end of the path equals $\mathrm{Cnt}(F_u^{\ell_u}) > 0$. However, if we have a similar path $F_v \colon v \rightsquigarrow v$ from v, we can follow the process symmetrically on the side of v. In the end, we construct a path T that is a reachability witness as

$$T = F_u^{\ell_u} \circ Q \circ F_v^{\ell_v} \circ \overline{Q} \circ \overline{F}_u^{\ell_u} \circ Q \circ \overline{F}_v^{\ell_v}$$

The crucial observation is that each of the above sub-paths starts and ends with an empty stack. Hence, the maximum stack height of T is the maximum among the maximum stack heights of these sub-paths. We thus arrive at the following lemma.

Lemma 4.4. *Assume that there are cyclic paths $F_u \colon u \rightsquigarrow u$ and $F_v \colon v \rightsquigarrow v$ such that $\mathrm{Stk}(F_u) = \mathrm{Stk}(F_v) = \epsilon$, and $\mathrm{Cnt}(F_u), \mathrm{Cnt}(F_v) > 0$. Let $\zeta = \max(\mathrm{MaxSH}(F_u), \mathrm{MaxSH}(F_v))$. If $u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$, then there is a path $T \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ such that $\mathrm{MaxSH}(T) \leq \max(\zeta, 2 \cdot n^2)$.*

In particular, the bound ζ comes from the stack height of the circular paths F_u and F_v, while the bound $2 \cdot n^2$ comes from the stack height of Q, following Lemma 4.3. At this point, a natural question is whether a bound on the maximum stack height of reachability witnesses bounds the search space for a witness. The following lemma shows that such a bound implies a bound on the length of the witness, and thus answers this question in positive.

Lemma 4.5. *Assume that there is a path $P \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ with $\mathrm{MaxSH}(P) \leq \delta$, for some $\delta \in \mathbb{N}$. Then there is a path $Q \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ with $|Q| \leq n^3 \cdot k^{3 \cdot \delta}$.*

We finally have all the ingredients to prove Theorem 1.2.

Proof of Theorem 1.2. We argue about the correctness of Algorithm 2. Clearly, if u does not cover $(v, 0)$ or v does not cover $(u, 0)$, due to Lemma 4.1 the algorithm returns False correctly in Line 2. On the other hand, if u does not cover $(v, 1)$, then the path that witnesses the coverability of $(v, 0)$ from u also witnesses reachability. Similarly if v does not cover $(u, 1)$, and thus the algorithm returns True correctly in Line 4.

Now consider the case that u covers $(v, 1)$ and v covers $(u, 1)$, and let $P_u \colon u \rightsquigarrow v$ and $P_v \colon v \rightsquigarrow u$ be the corresponding witness paths. Then we have paths $F_u = P_u \circ P_v$ and $F_v = P_v \circ P_u$ with $\mathrm{Stk}(F_u) = \mathrm{Stk}(F_v) = \epsilon$ and $\mathrm{Cnt}(F_u), \mathrm{Cnt}(F_v) \geq 0$. By Lemma 4.4, if v is $\mathcal{D}_k \odot \mathcal{D}_1$-reachable from u then there is a path $T \colon u \rightsquigarrow v$ such that $\mathrm{MaxSH}(T) \leq \delta = \max(\zeta, 2 \cdot n^2)$. Finally, Lemma 4.5 applies, and thus there is a path $Q \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ with $|Q| \leq n^3 \cdot k^{3 \cdot \delta}$. Algorithm 2 iterates over all such paths Q in Line 9, and tests whether any of them witnesses the reachability of v from u, returning True if such a path is found, and False otherwise. The desired result follows. □

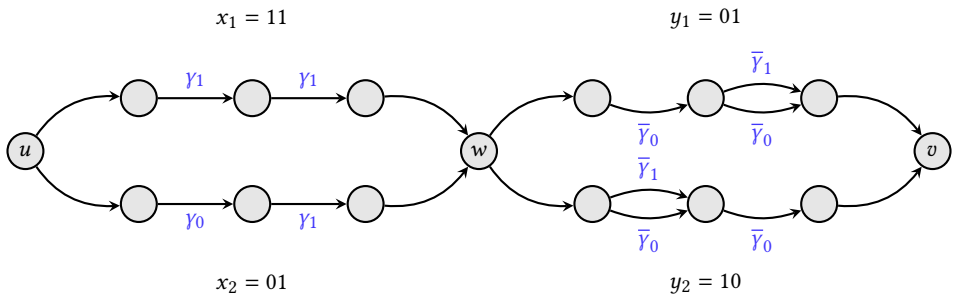

Fig. 6. A simple reduction from OV to Dyck reachability on non-bidirected graphs.

4.2 Bounded-Counter $\mathcal{D}_k \odot \mathcal{D}_1$ Reachability

In this section we focus on a bounded version of $\mathcal{D}_k \odot \mathcal{D}_1$ reachability. The goal is to determine all nodes u, v such that v is $\overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow}$ reachable from u via a path P such that $\text{MaxCnt}(P) = O(n)$. We first describe the algorithm for the quadratic upper bound (Theorem 1.3), and then prove that the bound is optimal (Theorem 1.4).

Upper bound. The algorithm for the upper bound follows the counter-flattening idea of Theorem 1.1. In particular, we use an algorithm identical to Algorithm 1, with the exception that the counter bound is $c = O(n)$. Note that the algorithm of [Chatterjee et al. 2018] solves bidirected $\mathcal{D}_k$ reachability, i.e., when $k \geq 2$ in general, and thus applies to this setting as well.

Proof of Theorem 1.3. The correctness follows straightforwardly. For the complexity, observe that the graph G' has $O(n \cdot c) = O(n^2)$ nodes. By Remark 1, the graph G is bounded-degree, which implies that G' is also bounded-degree, and hence G' has $O(n^2)$ edges. By [Chatterjee et al. 2018], solving $\mathcal{D}_k$-reachability on G' requires $O(n^2 \cdot \alpha(n))$ time. The desired result follows. □

The simple algorithm behind Theorem 1.3 leads to the natural question of whether a more involved algorithm can achieve a better bound, i.e., below quadratic. We next establish Theorem 1.4, showing that, in fact, no algorithm can bring the complexity below quadratic, under standard complexity-theoretic hypotheses. To show this, we establish a fine-grained reduction from the problem of orthogonal vectors.

Orthogonal vectors (OV). The input to OV is two sets X, Y, each containing n vectors in $\{0,1\}^D$, for some dimension $D = \omega(\log n)$. The task is to determine if there exists a pair $(x_i, y_j) \in X \times Y$ that is orthogonal, i.e., for each $\ell \in [D]$, we have $x_i[\ell] \cdot y_j[\ell] = 0$. The respective hypothesis states that the problem cannot be solved in time $O(n^{2-\epsilon})$, for any fixed $\epsilon > 0$ [Williams 2019].

Reduction. Given an instance (X, Y) of OV, we construct an interleaved and bidirected Dyck graph $G = (V, E, \Sigma = \Sigma_1 \uplus \{+1, -1\})$ such that for two distinguished nodes $u, v \in V$ we have a path $P\colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ with $\text{MaxCnt}(P) = O(n)$ iff there exists an orthogonal pair $(x_i, y_j) \in X \times Y$.

Intuition. Before presenting the construction we provide some intuition. To develop some insight, we first consider a simple reduction of OV to plain Dyck reachability (i.e., we don't have a counter) for non-bidirected graphs (see Figure 6). Starting from u, we have n parallel paths to an intermediate node q. Along the i-th such path, we push on the stack the encoding of the vector x_i using the

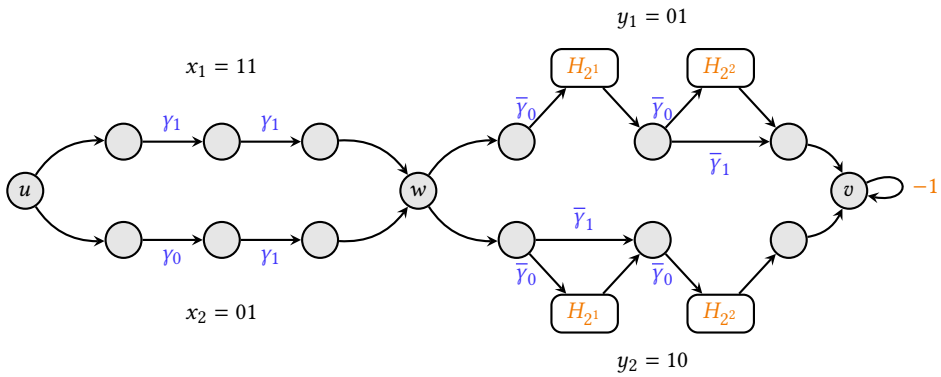

Fig. 7. Reduction from OV to $\mathcal{D}_k \odot \mathcal{D}_1$-reachability with linearly bounded counter. A path $P_x : u \rightsquigarrow w$ pushes on the stack an encoding of the vector $x \in X$ using symbols γ_0, γ_1. A path $P_y : w \rightsquigarrow v$ pops from the stack any vector that is orthogonal with y. When P_y pops a γ_0 in the ℓ-th coordinate, it increases the counter by 2^ℓ. In the end, the counter is emptied by self-looping on v.

symbols γ_0 and γ_1 in the straightforward way. A similar encoding for the vectors in Y suffices, where we match the contents of the stack by traversing a path that corresponds to a vector y_j. If $y_j[\ell] = 1$, then we can only move forward if $x_i[\ell] = 0$, thus we have a labeled transition that pops γ_0 from the stack. If $y_j[\ell] = 0$, then we expect either $x_i[\ell] = 0$ or $x_i[\ell] = 1$, thus we have two parallel labeled transitions, the first popping γ_0 and the second popping γ_1.

The above construction works for Dyck reachability on non-bidirected graphs, but fails for bidirected graphs. The issue is that, if $y_j[\ell] = 0$, then we can follow the edge that pops γ_1, and then follow the other edge (i.e., the one that pops γ_0) in inverse direction, which has the effect of pushing γ_0 on the stack. But now the stack encodes a vector x that is not part of X, which affects correctness. Note, however, that the above construction reduces OV to Dyck reachability, as opposed to interleaved Dyck reachability (i.e., it does not make use of the counter).

To alleviate the bidirectedness problem in interleaved Dyck reachability, we make use of the counter to force that certain edges are traversed only one way. Similarly as before, to encode $y_j[\ell]$, we have an edge popping γ_0. However, when $y_j[\ell] = 0$, in order to pop γ_1 we have to traverse a counter gadget that increases the counter by 2^ℓ. This implies that this path cannot be traversed backwards, as this would require to decrease the counter by 2^ℓ, which is a value that cannot have been accumulated so far (the current counter value can be at most $\sum_{l<\ell} 2^l = 2^\ell - 1$). In the end, we self-loop on v to reduce the counter to 0. See Figure 7 for an illustration.

Formal construction. We now present the formal construction.

(1) We have special nodes $\{u, w, v\} \cup \bigcup_{i\in[n],j\in[D+1]} \{x_i^\ell, y_i^\ell\}$, as well as n copies of the counter gadgets H_{2^ℓ}, for every $\ell \in [D]$.
(2) For every $i \in [n]$, we have edges $u \to x_i^1$, $x_i^{D+1} \to w$, $w \to y_i^{D+1}$ and $y_i^1 \to v$.
(3) For every $i \in [n]$ and $\ell \in [D]$, we have $x_i^\ell \xrightarrow{\gamma_j} x_i^{\ell+1}$, where $j = x_i[\ell]$.
(4) For every $i \in [n]$ and $\ell \in [D]$, we have $y_{i+1}^\ell \xrightarrow{\overline{\gamma}_0} y_i^\ell$, where $j = y_i[\ell]$. Moreover, if $y_i[\ell] = 0$, we have an edge $y_i^\ell \xrightarrow{\overline{\gamma}_1} p_i^\ell$ and an edge $q_i^\ell \to y_{i+1}$, where p_i^ℓ and q_i^ℓ are the entry and exit nodes, respectively, of the i-th copy of the counter gadget H_{2^ℓ}.

(5) Finally, we have a self-loop $v \xrightarrow{-1} v$.

Correctness. We now turn our attention to the correctness of the construction. It is clear that if there are x_i and y_j that are orthogonal, we have a path $P\colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ by traversing the corresponding nodes x_i^ℓ and y_j^ℓ, and end the path by looping on v. Now assume that there exists a (irreducible) path $P\colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$. Observe that the path goes from u to w by pushing on the stack the contents of a vector x_i. At this point the stack dictates how the path can continue from y_j^ℓ to $y_j^{\ell+1}$. Moreover, the value of the counter while the path is at y_j^ℓ is bounded by $\sum_{l<\ell} 2^l = 2^\ell - 1$. Hence, although the path can transition back from y_j^ℓ to $y_j^{\ell-1}$, it can only do so by pushing γ_0 on the stack. This has the effect that if the path ever returns to w, its stack will encode a vector (possibly not in X) that has at least 1 in the coordinates ℓ for which $x_i[\ell] = 1$. It follows that when P reaches v, it has traversed the nodes x_i^ℓ and y_j^ℓ, for some $i, j \in [n]$, such that x_i and y_j are orthogonal. We thus have the following lemma.

Lemma 4.6. *There is a path $P\colon u \overset{\mathcal{D}_k \odot \mathcal{D}_1}{\rightsquigarrow} v$ iff there is an orthogonal pair $(x_i, y_j) \in X \times Y$.*

We are now ready to conclude Theorem 1.4.

Proof of Theorem 1.4. Lemma 4.6 proves the correctness of the construction, so it remains to argue about the complexity. The number of nodes in G is $O(n \cdot D^2)$, as we have $O(n \cdot D)$ nodes x_i^ℓ and y_i^ℓ, while each counter gadget H_{2^ℓ} uses ℓ nodes. Since $\ell \leq D$, we have $O(n \cdot D^2)$ nodes in total. Finally, observe that G is sparse, and thus there are $O(n \cdot D^2)$ edges as well. The desired result follows. □

5 UNDECIDABILITY OF $D_k \odot D_k$ REACHABILITY

Finally, in this section we prove the undecidability of general $D_k \odot D_k$ reachability on bidirected graphs (Theorem 1.5). Our reduction is from $D_k \odot D_k$ reachability on non-bidirected graphs, which is known to be undecidable [Reps 2000].

Reduction. Consider a non-bidirected graph $G = (V, E, \Sigma)$ where $\Sigma = \Sigma_1 \uplus \Sigma_2$, and the problem of $D_k \odot D_k$ reachability wrt the alphabets Σ_1 and Σ_2 on two nodes u and v of G. We assume wlog that G is not a multi-graph, i.e., every pair of nodes has a unique edge between them. We construct a bidirected graph $G' = (V', E', \Sigma')$ where $V \subset V'$ and $\Sigma' = \Sigma'_1 \uplus \Sigma'_2$, and such that a node $t \in V'$ is $D_k \odot D_k$-reachable from node $s \in V'$, wrt the alphabets Σ'_1 and Σ'_2, iff v is reachable from u in G. To keep the exposition simple, the size of Σ' is proportional to the size of G. Standard constructions can turn Σ' to constant, though this comes at the expense of increasing the graph size by a logarithmic factor (see, e.g., [Chistikov et al. 2021]). Since we show undecidability, this increase is not a concern.

Intuitive description. We start with the intuition behind the reduction, while we refer to Figure 8 for illustrations on a small example. Given G and nodes u and v, we construct G' such that there is a path $P\colon u \rightsquigarrow v$ in G iff there exists a path $P'\colon s \rightsquigarrow t$ in G', as follows. Initially, the path P' "guesses" the path P edge-by-edge in reverse order, while it (i) pushes every guess in the first stack, and (ii) uses the second stack to simulate the behavior of the second stack along P in G (in reverse), mirrored under the parenthesis complementation operator $\overline{\gamma}$ (i.e., if the edge in G pushes γ, the simulating edge will pop γ). Note that complementing the letters as found in the reverse order of P, in fact simulates the second stack in the forward order of P.

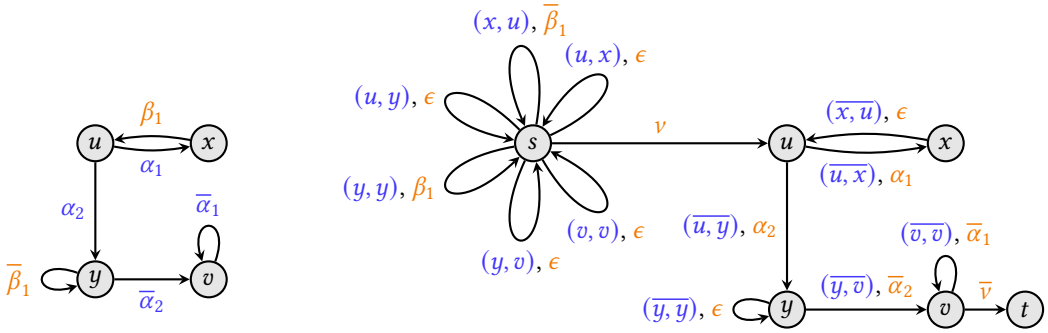

Fig. 8. *(Left):* An interleaved Dyck graph G with a reachability question on u, v. *(Right):* The bidirected interleaved Dyck graph G' constructed in our reduction, with a reachability question on s, t.

After this part is done, the second stack is empty, while the first stack contains the sequence of edges of P in top-down order (i.e., the last edge of P is at the top of the stack). At this point, P' will traverse the remaining of the graph G' and verify that the path P stored in first stack is a valid path of G, while using the second stack to simulate the effect of P on the first stack. As a preliminary step, P' is forced to push a special symbol ν on the second stack as it takes a transition from s to u. This is a simple trick to deal with bidirectedness, in the sense that, if in the future P' decides to come back to s and alter the guessed path, it can only do so by popping ν. This will imply that the second stack is empty and the first stack contains a suffix of the guessed P, and thus P' is reducible.

In order to verify the guess of P', the graph G' is constructed as follows. For every edge $x \xrightarrow{\gamma} y$ of G, we have a two-edged path $x \xrightarrow{(\overline{x,y}),\delta} y$ in G'. If $\gamma \in \Sigma_1$, i.e., the edge in G manipulates the first stack, δ is identical to γ but affects the second stack of G'. On the other hand, if $\gamma \in \Sigma_2$, then $\delta = \epsilon$. Intuitively, we can traverse the path $x \xrightarrow{(\overline{x,y}),\delta} y$ in G' iff the top-most edge guess in the first stack matches the edge $x \xrightarrow{\gamma} y$ of G. By traversing this path, if γ manipulates the first stack in G, we use $\delta = \gamma$ to manipulate the second stack in G'. On the other hand, if γ manipulates the second stack in G we ignore its effect ($\delta = \epsilon$), as the second stack has already been simulated in the first phase where we guessed the path P while self-looping on s.

The key idea in the above construction is that while self-looping over s, the path P' can only guess the edges of P in the direction they appear in G. Although P' can traverse edges backwards in G' (due to bidirectedness), this will generally result in pushing more edge symbols $e_1, \ldots, e_\ell$ on the first stack. Because of the letter ν, the only way to pop these symbols from the stack is for P' to reverse the sub-path that added these symbols, which makes P' reducible.

Construction. We now proceed with the detailed construction. For ease of presentation, we specify some parts of G' as paths, meaning unique sequences of labeled edges, without explicit reference to all the nodes and edges in the path. We begin with the alphabets

$$\Sigma'_1 = \bigcup_{x \xrightarrow{\gamma} y \in E} \{(x,y), (\overline{x,y})\} \qquad \Sigma'_2 = \Sigma_1 \cup \Sigma_2 \cup \{\nu, \overline{\nu}\}$$

Let $\Sigma' = \Sigma'_1 \uplus \Sigma'_2$. We have $V \subset V'$, while G' contains some additional nodes and edges, defined by the following paths. For every edge $x \xrightarrow{\gamma} y \in E$, we construct a cyclic path $s \xrightarrow{(x,y),\delta} s$, where $\delta = \overline{\gamma}$

if $\gamma \in \Sigma_2$ and $\delta = \epsilon$ otherwise. We also have an edge $s \xrightarrow{\nu} u$. For every edge $x \xrightarrow{\gamma} y$ of G, we have a path $x \xrightarrow{(\overline{x},\overline{y}),\delta} y$ in G', where $\delta = \gamma$ if $\gamma \in \Sigma_1$ and $\delta = \epsilon$ otherwise. Finally, we have an edge $v \xrightarrow{\overline{\nu}} t$.

Correctness. It remains to argue about the correctness of the above construction, i.e., we have $u \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} v$ in G iff $s \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} t$ in G'.

Proof of Theorem 1.5. We argue separately about completeness and soundness.

Completeness. Assume that there is a path $P \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} v$ in G. We construct a path $P' \colon s \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} t$ in G' as follows. Let $P = e_1, \ldots, e_m$ be the sequence of edges traversed by P, without their labels. For each $i \in [m]$, we take the self-loop path $s \xrightarrow{e_{m-i+1},\delta} s$. Afterwards, we traverse the edge $s \xrightarrow{\nu} u$, and then we repeatedly traverse the path $x \xrightarrow{(\overline{x},\overline{y}),\delta} y$, where (x, y) is the current top symbol on the first stack. Finally, we traverse the edge $v \xrightarrow{\overline{\nu}} t$.

Let P'_1 be the prefix of P' right before we traverse the edge $s \xrightarrow{\nu} u$. Observe that

$$\lambda(P'_1) \downharpoonright \Sigma'_2 = \overline{\lambda(\overline{P}) \downharpoonright \Sigma_2} = \lambda(P) \downharpoonright \Sigma_2$$

and hence P'_1 is valid on the second stack. From this point, It is straightforward to verify by induction that P' is a valid path with $\mathrm{Stk}_1(P') = \mathrm{Stk}_2(P') = \epsilon$, and hence $P' \colon s \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} t$.

Soundness. Assume that there is a path $P' \colon s \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} t$ in G'. Let P'_1 be the prefix of P' right before the last time that P' traverses the edge $s \xrightarrow{\nu} u$. Observe that $\mathrm{Stk}_2(P'_1) = \epsilon$. Indeed, at that point, the second stack cannot contain any symbol from Σ_2, as these symbols are not popped in the suffix of P' that succeeds P'_1. Moreover, the second stack cannot contain any symbol from Σ_1, as these symbols can only be pushed in the stack on top of ν, which prevents P'_1 to traverse the edge $s \xrightarrow{\nu} v$ in reverse.

We now argue that at the end of P'_1, the first stack encodes a path $P \colon u \overset{\mathcal{D}_k \odot \mathcal{D}_k}{\rightsquigarrow} v$ in G. Indeed, let P'_2 be the suffix of P' such that $P' = P'_1 \circ P'_2$. Since P' is an irreducible path, P'_2 is also irreducible. This implies that P'_2 never pushes an element on the first stack. Hence, the whole of P'_2 except the last edge $v \xrightarrow{\overline{\nu}} t$ matches the content of the first stack at the end of P'_1. This implies that $\mathrm{Stk}_1(P'_1)$ encodes a path $P \colon u \rightsquigarrow v$ in G such that $\lambda(P) \downharpoonright \Sigma_1 \in \mathcal{D}(\Sigma_1)$, i.e., the label of P produces a valid Dyck string wrt the first alphabet. Finally, since $\mathrm{Stk}_2(P'_1) = \epsilon$, it follows that $\lambda(P) \downharpoonright \Sigma_2 \in \mathcal{D}(\Sigma_2)$, and thus the label of P produces a valid Dyck string wrt the second alphabet as well. Hence P witnesses the $\mathcal{D}_k \odot \mathcal{D}_k$ reachability of v from u. □

6 EXPERIMENTS

In this section we report on the experimental evaluation of our algorithms for $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability (Theorem 1.1) and $\mathcal{D}_k \odot \mathcal{D}_1$ reachability with linearly-bounded counters (Theorem 1.3). We first describe some straightforward optimizations to the baseline algorithms (Section 6.1) and then present the experimental results (Section 6.2).

Fig. 9. *(Left):* A bidirected, interleaved Dyck graph G with a doubly-self-looped node x. *(Right):* The two connected components on which we perform $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability separately in order to infer whether x is reachable from any other node.

6.1 Experimental Algorithms

We describe three straightforward optimizations.

1. Under-approximating with $\mathcal{D}_k$. Our first optimization is based on the simple observation that $\mathcal{D}_k \odot \mathcal{D}_k$ reachability wrt two alphabets Σ_1 and Σ_2 can be under-approximated by performing $\mathcal{D}_k$ reachability on the union alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$. Thus, as a first step, we perform bidirected $\mathcal{D}_k$ reachability on the input graph, and reduce the graph by merging pairs that are $\mathcal{D}_k$ reachable.

2. Removing doubly-self-looped nodes. Our final optimization concerns only $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability, and is an effective procedure for removing doubly-self-looped nodes. Indeed, for any node x that has a self loop on each counter, we can make the following observations.

(1) Any witness path $P\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$ that goes through x implies also the existence of reachability paths $u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} x$ and $x \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} v$, by self-looping on x a sufficient number of times. Hence, we can focus on whether x reaches any other node, and if not, remove x.
(2) For any witness path $Q\colon u \overset{\mathcal{D}_1 \odot \mathcal{D}_1}{\rightsquigarrow} x$, wlog Q is a path that never exits x, i.e., Q has the form $Q = u \dots x \dots x$. Hence, in order to decide whether x is reachable from any other node, it suffices to compute $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability locally on each connected component that is connected to the rest of the graph only via x. In practice, we have found that when doubly self-looped nodes are present, they are articulation points that separate many small connected components.

Thus, performing the above process on x allows us to either (i) merge x with another node that reaches x and repeat, or (ii) remove x from the graph. In the first case we have reduced the size of the graph, while in the second case we proceed recursively on the small connected components that are created. Figure 9 illustrates this process on a small example.

3. Node trimming. Our second optimization is based on identifying simple motifs in the graph which guarantee that a node x (i) is not reachable from any other node, and (ii) if there is a path P that witnesses reachability and goes through x, there is a path Q that witnesses the same reachability without going through x. Applying this process repeatedly can prune away many such "isolated" nodes and thus reduce the effective size of the graph. Figure 10 illustrates two simple cases that allow successive trimming on nodes z and x.

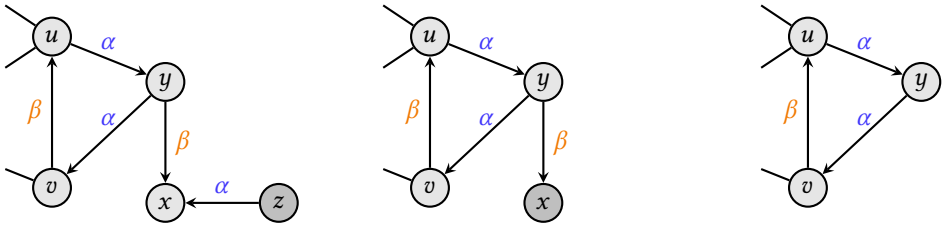

Fig. 10. *(Left):* A sub-graph of a bidirected, interleaved Dyck graph G. By construction, any witness path that goes through z must contain the sub-path xzx, which can be substituted by x. Moreover, no path starting in z can go beyond x, as going to y must pop a β. Thus z can safely be removed. *(Middle):* By construction, any witness path that goes through x must contain the sub-path yxy, which can be substituted for y. Moreover, any path leading to x will have a stack word of at least one β, meaning it is unreachable and can be safely removed. *(Right):* The trimmed sub-graph.

In our experiments, we found that all heuristics were applicable in all benchmarks. In particular, steps (1) and (2) were important for significantly reducing the input graph to small, while step (3) helped with running time and also with reducing the graph further in some cases.

6.2 Experimental Results

We are now ready to report on our experimental results.

Experimental setup. We have used the DaCapo benchmarks [Blackburn 2006] as in earlier works [Li et al. 2021; Zhang and Su 2017], for performing context-sensitive and field-sensitive alias analysis. Each benchmark provided one interleaved bidirected Dyck graph that models context and field sensitivity, and creates an instance of $\mathcal{D}_k \odot \mathcal{D}_k$ reachability. In all cases needed, we projected a language $\mathcal{D}_k$ to $\mathcal{D}_1$ by projecting the parenthesis alphabet to a unique parenthesis symbol. For $\mathcal{D}_k \odot \mathcal{D}_1$ reachability we used a counter bound of n.

Each of the above graphs was given as input to our algorithm. This is in small difference to the procedure of [Li et al. 2021], where the graph was first reduced using a recent fast simplification technique [Li et al. 2020]. Our algorithms were implemented in C++ 11 without any compiler optimizations, and were executed on a conventional laptop with a Core i7 CPU with 8 GB of memory running Ubuntu 20.10. The results are reported in Section 6.2.

Results on $\mathcal{D}_1 \odot \mathcal{D}_1$. We first discuss $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability. Recall that $\mathcal{D}_k$ reachability (on the union alphabet) serves as an under-approximation of $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability. The table shows that $\mathcal{D}_k$ reachability already discovers most of the connected components, with $\mathcal{D}_1 \odot \mathcal{D}_1$ refining this results with a few remaining components that are hard to detect. This observation is in alignment to the study of [Li et al. 2021], where it is reported that the number of new pairs discovered by $\mathcal{D}_1 \odot \mathcal{D}_1$ reachability is only about 1% more than those discovered by $\mathcal{D}_k$ reachability.

Regarding running time, we see that our algorithm handles each benchmark within seconds, while the total benchmark set is processed in less than 5 minutes. As a comparison, the recent method of [Li et al. 2021] reported more than 3 days (~ 74 hours) for the same benchmark set when run on a relatively big machine, and it does so when the input graphs are simplified by a preprocessing step [Li et al. 2020]. As a sanity check, we have verified that our algorithm also handles the simplified graphs within seconds, while our own implementation of the technique of [Li et al. 2021] does not finish on our machine after 5 hours even for small graphs.

Table 2. Experimental results on $\mathcal{D}_1 \odot \mathcal{D}_1$ and $\mathcal{D}_k \odot \mathcal{D}_1$ reachability. In each case, ID-CCs denotes the number of connected components wrt the interleaved Dyck language, while D-CCs denotes the number of connected components wrt the under-approximating Dyck language on the union alphabet.

Benchmark	n	$\mathcal{D}_1 \odot \mathcal{D}_1$			$\mathcal{D}_k \odot \mathcal{D}_1$		
		ID-CCs	**D-CCs**	**Time (s)**	**ID-CCs**	**D-CCs**	**Time (s)**
antlr	29831	26793	26825	40.5	27014	27015	9.1
bloat	36181	32693	32725	16.2	32946	32946	12.8
chart	67535	60787	60844	32.8	61158	61158	65.1
eclipse	30981	27812	27840	16.0	27999	28018	12.3
fop	61016	54671	54723	29.8	55012	55012	58.4
hsqldb	27494	24584	24610	15.5	24775	24776	8.8
jython	36162	31811	31845	26.3	32060	32062	21.5
luindex	28595	25610	25636	15.6	25809	25810	7.9
lusearch	29530	26417	26447	17.8	26655	26655	9.1
pmd	31333	28064	28093	18.2	28296	28297	9.9
xalan	27358	24498	24523	15.2	24689	24690	7.4

Results on $\mathcal{D}_k \odot \mathcal{D}_1$. We now turn our attention to $\mathcal{D}_k \odot \mathcal{D}_1$ reachability. Interestingly, we see that staying faithful to one Dyck language (not projecting $\mathcal{D}_k$ to $\mathcal{D}_1$) increases the number of connected components (i.e., there are fewer reachable pairs). In principle, this could be due to the theoretical incompleteness arising from the bound on the counter. However, we believe that our reports do not miss any reachable pairs, and thus the reduced reachability relationships (compared to $\mathcal{D}_1 \odot \mathcal{D}_1$) are true negatives and thus increase the analysis precision. This is further supported by our experience that all reported reachability relationships under $\mathcal{D}_k \odot \mathcal{D}_1$ were already formed with counter values much smaller than n. Moreover, we find that performing $\mathcal{D}_k$ reachability (on the union alphabet) provides an under-approximation that is often very close to (and sometimes matches) the final result.

Finally, regarding time, we again see that our algorithm handles each benchmark within seconds, and the whole benchmark set is again processed in less than 5 minutes. These times indicate that our algorithms are suitable for static analysis tools.

7 RELATED WORK

Complexity of Dyck reachability. The immense importance of Dyck reachability in static analyses has lead to a systematic study of its complexity in various settings. General Dyck reachability can be solved in $O(n^3)$ time [Yannakakis 1990] using an extension of the CYK algorithm to graphs. The bound is believed to be tight as the problem is 2NPDA-hard [Heintze and McAllester 1997], while the combinatorial cubic hardness persists even on constant-treewidth graphs [Chatterjee et al. 2018]. Sub-cubic algorithms do exist, but they only offer logarithmic speedups [Chaudhuri 2008]. When the underlying graph is a Recursive State Machine (RSM) with constant entries and exits, treewidth has been shown to lead to fast on-demand reachability queries [Chatterjee et al. 2020, 2015]. Despite the cubic hardness of the general problem, it is known to have sub-cubic certificates for both positive and negative instances [Chistikov et al. 2021]. Dyck reachability over a single parenthesis symbol (aka one-counter systems) has been shown to admit a sub-cubic bound [Bradford 2018]. The best current bound for this setting is $O(n^\omega \cdot \log^2 n)$ [Mathiasen and Pavlogiannis 2021], where ω is the matrix-multiplication exponent, while it is also known that the

problem has a (conditional) $\Omega(n^\omega)$ lower bound even for the single-pair question [Cetti Hansen et al. 2021].

The algorithmic benefit of bidirectedness was highlighted in [Yuan and Eugster 2009], where an $O(n \cdot \log n)$ algorithm was presented when the underlying graph is a bidirected tree. Later this bound was improved to $O(n)$ for trees, while the problem was shown to take $O(n^2)$ time (and $O(n \cdot \log n)$ expected time) on general bidirected graphs, thereby breaking below the cubic bound [Zhang et al. 2013]. This sequence of improvements ended with an $O(n \cdot \alpha(n))$ algorithm on general bidirected graphs (and $O(n)$ expected time), where $\alpha(n)$ is the inverse Ackermann function, and the bound was also shown to be tight [Chatterjee et al. 2018].

Interleaved Dyck reachability. The modeling power of interleaved Dyck reachability, as well as its undecidability, were illustrated in [Reps 2000]. As the problem is of vital importance to static analyses, various approximations have been developed. The most basic, but also widely used approach is to approximate one Dyck language with a regular language, e.g., by bounding the recursion depth [Lerch et al. 2015; Sridharan and Bodík 2006]. Other approximate techniques involve linear conjunctive language reachability [Zhang and Su 2017], synchronized pushdown systems [Späth et al. 2019], and CEGAR-style techniques [Ferles et al. 2021]. The hardness of the problem stems from the fact that the underlying pushdown automata is operating on multiple stacks. The problem also arises in distributed models of pushdown systems [Madhusudan and Parlato 2011], and has also been addressed with approximations based on parameterization [Kahlon 2008] and bounded-context switching [Qadeer and Rehof 2005].

Vector addition systems. When one or both Dyck languages are over one parenthesis symbol, interleaved Dyck reachability falls in the class of vector addition systems. In particular, $\mathcal{D}_1 \odot \mathcal{D}_1$ yields a unary vector addition system with states in two dimensions, while $\mathcal{D}_k \odot \mathcal{D}_1$ yields a unary pushdown vector addition system in one dimension. Reachability for the first case is NL-complete [Englert et al. 2016], while the decidability of reachability in the second case is open as for now (to our knowledge, see, e.g., [Schmitz and Zetzsche 2019, Table 1]). Nevertheless, the latter model enjoys some nice properties, such as decidability of coverability [Leroux et al. 2015b] and decidability of the boundedness of the reachable set [Leroux et al. 2015a]. Independently of our work, bidirectedness was recently studied in vector additions systems in [Ganardi et al. 2021].

8 CONCLUSION

In this work we have addressed the decidability and complexity of interleaved Dyck reachability on bidirected graphs, inspired by the recent work of [Li et al. 2021]. We have developed an efficient, nearly cubic-time algorithm for the $\mathcal{D}_1 \odot \mathcal{D}_1$ case, while we have shown that the $\mathcal{D}_k \odot \mathcal{D}_1$ is decidable. As a means to more tractable solutions, we have shown a nearly quadratic bound for $\mathcal{D}_k \odot \mathcal{D}_1$ when restricting witnesses to linearly-bounded counters, and we have shown that this quadratic bound is tight. Finally, we have shown that general $\mathcal{D}_k \odot \mathcal{D}_k$ reachability remains undecidable on bidirected graphs. Our results cover an important missing piece in the decidability and complexity landscape of static analyses modeled as Dyck/CFL reachability. Moreover, our experiments show that the new algorithms can handle big benchmarks within seconds, making them suitable for static analysis tools.

ACKNOWLEDGMENTS

We are grateful to Georg Zetzsche for comments on an earlier draft of the paper, and to the authors of [Li et al. 2021] for sharing their dataset and assisting us on its use. We also thank anonymous reviewers for their constructive feedback.

REFERENCES

2003. T. J. Watson Libraries for Analysis (WALA). https://github.com. (2003).

Rajeev Alur and P. Madhusudan. 2004. Visibly Pushdown Languages. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing (STOC '04)*. Association for Computing Machinery, New York, NY, USA, 202–211. https://doi.org/10.1145/1007352.1007390

Robert S. Arnold. 1996. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA.

Stephen M. et al. Blackburn. 2006. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. In *OOPSLA*.

Eric Bodden. 2012. Inter-procedural Data-flow Analysis with IFDS/IDE and Soot. In *SOAP*. ACM, New York, NY, USA.

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

Jakob Cetti Hansen, Adam Husted Kjelstrøm, and Andreas Pavlogiannis. 2021. Tight bounds for reachability problems on one-counter and pushdown systems. *Inform. Process. Lett.* 171 (2021), 106135. https://doi.org/10.1016/j.ipl.2021.106135

Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2018. Optimal Dyck Reachability for Data-Dependence and Alias Analysis. *Proc. ACM Program. Lang.* 2, POPL, Article Article 30 (Dec. 2018), 30 pages.

Krishnendu Chatterjee, Amir Kafshdar Goharshady, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. 2020. Optimal and Perfectly Parallel Algorithms for On-demand Data-Flow Analysis. In *Programming Languages and Systems*, Peter Müller (Ed.). Springer International Publishing, Cham, 112–140.

Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Andreas Pavlogiannis, and Prateesh Goyal. 2015. Faster Algorithms for Algebraic Path Properties in Recursive State Machines with Constant Treewidth. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. Association for Computing Machinery, New York, NY, USA, 97–109. https://doi.org/10.1145/2676726.2676979

Krishnendu Chatterjee, Bernhard Kragl, Samarth Mishra, and Andreas Pavlogiannis. 2017. *Faster Algorithms for Weighted Recursive State Machines*. Springer Berlin Heidelberg, Berlin, Heidelberg, 287–313. https://doi.org/10.1007/978-3-662-54434-1_11

Swarat Chaudhuri. 2008. Subcubic Algorithms for Recursive State Machines. *SIGPLAN Not.* 43, 1 (Jan. 2008), 159–169. https://doi.org/10.1145/1328897.1328460

Dmitry Chistikov, Rupak Majumdar, and Philipp Schepper. 2021. Subcubic Certificates for CFL Reachability. (2021). arXiv:cs.FL/2102.13095

Alain Deutsch. 1994. Interprocedural May-Alias Analysis for Pointers: Beyond k-Limiting. *SIGPLAN Not.* 29, 6 (June 1994), 230–241. https://doi.org/10.1145/773473.178263

Matthias Englert, Ranko Lazić, and Patrick Totzke. 2016. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*. Association for Computing Machinery, New York, NY, USA, 477–484. https://doi.org/10.1145/2933575.2933577

Kostas Ferles, Jon Stephens, and Isil Dillig. 2021. Verifying Correct Usage of Context-Free API Protocols. *Proc. ACM Program. Lang.* 5, POPL, Article 17 (Jan. 2021), 30 pages. https://doi.org/10.1145/3434298

Moses Ganardi, Rupak Majumdar, and Georg Zetzsche. 2021. The complexity of bidirected reachability in valence systems. (2021). arXiv:cs.FL/2110.03654

Nevin Heintze and David McAllester. 1997. On the Cubic Bottleneck in Subtyping and Flow Analysis. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS '97)*. IEEE Computer Society, Washington, DC, USA, 342–. http://dl.acm.org/citation.cfm?id=788019.788876

Wei Huang, Yao Dong, Ana Milanova, and Julian Dolby. 2015. Scalable and Precise Taint Analysis for Android. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. Association for Computing Machinery, New York, NY, USA, 106–117. https://doi.org/10.1145/2771783.2771803

Vineet Kahlon. 2008. Parameterization as Abstraction: A Tractable Approach to the Dataflow Analysis of Concurrent Programs. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*. IEEE Computer Society Press, 181–192.

Adam Husted Kjelstrøm and Andreas Pavlogiannis. 2021. The Decidability and Complexity of Interleaved Bidirected Dyck Reachability. (2021). arXiv:cs.PL/2111.05923 http://arxiv.org/abs/2111.05923

Johannes Lerch, Johannes Späth, Eric Bodden, and Mira Mezini. 2015. Access-Path Abstraction: Scaling Field-Sensitive Data-Flow Analysis with Unbounded Access Paths. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15)*. IEEE Press, 619–629. https://doi.org/10.1109/ASE.2015.9

Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. 2015a. On Boundedness Problems for Pushdown Vector Addition Systems. In *Reachability Problems*, Mikolai Bojanczyk, Slawomir Lasota, and Igor Potapov (Eds.). Springer International Publishing, Cham, 101–113.

Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. 2015b. On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension. In *Automata, Languages, and Programming*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 324–336.

Ondřej Lhoták and Laurie Hendren. 2006. Context-Sensitive Points-to Analysis: Is It Worth It?. In *Proceedings of the 15th International Conference on Compiler Construction (CC)*. 47–64.

Yuanbo Li, Qirun Zhang, and Thomas Reps. 2020. Fast Graph Simplification for Interleaved Dyck-Reachability. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 780–793. https://doi.org/10.1145/3385412.3386021

Yuanbo Li, Qirun Zhang, and Thomas Reps. 2021. On the Complexity of Bidirected Interleaved Dyck-Reachability. *Proc. ACM Program. Lang.* 5, POPL, Article 59 (Jan. 2021), 28 pages. https://doi.org/10.1145/3434340

Jingbo Lu and Jingling Xue. 2019. Precision-Preserving yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 148 (Oct. 2019), 29 pages. https://doi.org/10.1145/3360574

P. Madhusudan and Gennaro Parlato. 2011. The Tree Width of Auxiliary Storage. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*. Association for Computing Machinery, New York, NY, USA, 283–294. https://doi.org/10.1145/1926385.1926419

Anders Alnor Mathiasen and Andreas Pavlogiannis. 2021. The Fine-Grained and Parallel Complexity of Andersen's Pointer Analysis. *Proc. ACM Program. Lang.* 5, POPL, Article 34 (Jan. 2021), 29 pages. https://doi.org/10.1145/3434315

Ana Milanova. 2020. FlowCFL: Generalized Type-Based Reachability Analysis: Graph Reduction and Equivalence of CFL-Based and Type-Based Reachability. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 178 (Nov. 2020), 29 pages. https://doi.org/10.1145/3428246

Laurent Pierre. 1992. Rational indexes of generators of the cone of context-free languages. *Theoretical Computer Science* 95, 2 (1992), 279 – 305. https://doi.org/10.1016/0304-3975(92)90269-L

Shaz Qadeer and Jakob Rehof. 2005. Context-Bounded Model Checking of Concurrent Software. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*. Springer-Verlag, Berlin, Heidelberg, 93–107. https://doi.org/10.1007/978-3-540-31980-1_7

Jakob Rehof and Manuel Fähndrich. 2001. Type-base Flow Analysis: From Polymorphic Subtyping to CFL-reachability. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. 54–66.

Thomas Reps. 1995. Shape Analysis As a Generalized Path Problem. In *Proceedings of the 1995 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation (PEPM '95)*. ACM, 1–11.

Thomas Reps. 1997. Program Analysis via Graph Reachability. In *Proceedings of the 1997 International Symposium on Logic Programming (ILPS)*. 5–19.

Thomas Reps. 2000. Undecidability of Context-sensitive Data-dependence Analysis. *ACM Trans. Program. Lang. Syst.* 22, 1 (2000), 162–186.

Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *POPL*. ACM, New York, NY, USA.

Thomas Reps, Susan Horwitz, Mooly Sagiv, and Genevieve Rosay. 1994. Speeding Up Slicing. *SIGSOFT Softw. Eng. Notes* 19, 5 (1994), 11–20.

Sylvain Schmitz and Georg Zetzsche. 2019. Coverability Is Undecidable in One-Dimensional Pushdown Vector Addition Systems with Resets. In *Reachability Problems*, Emmanuel Filiot, Raphaël Jungers, and Igor Potapov (Eds.). Springer International Publishing, Cham, 193–201.

Lei Shang, Xinwei Xie, and Jingling Xue. 2012. On-demand Dynamic Summary-based Points-to Analysis. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO '12)*. ACM, 264–274.

Olin Grigsby Shivers. 1991. *Control-Flow Analysis of Higher-Order Languages of Taming Lambda*. Ph.D. Dissertation. USA. UMI Order No. GAX91-26964.

Johannes Späth, Karim Ali, and Eric Bodden. 2019. Context-, Flow-, and Field-Sensitive Data-Flow Analysis Using Synchronized Pushdown Systems. *Proc. ACM Program. Lang.* 3, POPL, Article 48 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290361

Manu Sridharan and Rastislav Bodík. 2006. Refinement-based Context-sensitive Points-to Analysis for Java. *SIGPLAN Not.* 41, 6 (2006), 387–400.

Manu Sridharan, Denis Gopan, Lexin Shan, and Rastislav Bodík. 2005. Demand-driven Points-to Analysis for Java. In *OOPSLA*.

Hao Tang, Di Wang, Yingfei Xiong, Lingming Zhang, Xiaoyin Wang, and Lu Zhang. 2017. Conditional Dyck-CFL Reachability Analysis for Complete and Efficient Library Summarization. In *Programming Languages and Systems*, Hongseok Yang (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 880–908.

Jyothi Vedurada and V. Krishna Nandivada. 2019. Batch Alias Analysis. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*. IEEE Press, 936–948. https://doi.org/10.1109/ASE.2019.00091

Ryan Williams. 2005. A New Algorithm for Optimal 2-Constraint Satisfaction and Its Implications. *Theor. Comput. Sci.* 348, 2 (Dec. 2005), 357–365. https://doi.org/10.1016/j.tcs.2005.09.023

Virginia Vassilevska Williams. 2019. *On some fine-grained questions in algorithms and complexity*. Technical Report.

Guoqing Xu, Atanas Rountev, and Manu Sridharan. 2009. Scaling CFL-Reachability-Based Points-To Analysis Using Context-Sensitive Must-Not-Alias Analysis. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming (Genoa)*. 98–122.

Dacong Yan, Guoqing Xu, and Atanas Rountev. 2011. Demand-driven Context-sensitive Alias Analysis for Java. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA)*. 155–165.

Mihalis Yannakakis. 1990. Graph-theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. 230–242.

Hao Yuan and Patrick Eugster. 2009. An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees. In *Proceedings of the 18th European Symposium on Programming Languages and Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009 (ESOP)*. 175–189.

Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. 2013. Fast Algorithms for Dyck-CFL-reachability with Applications to Alias Analysis *(PLDI)*. ACM.

Qirun Zhang and Zhendong Su. 2017. Context-Sensitive Data-Dependence Analysis via Linear Conjunctive Language Reachability. *SIGPLAN Not.* 52, 1 (Jan. 2017), 344–358. https://doi.org/10.1145/3093333.3009848

Xin Zheng and Radu Rugina. 2008. Demand-driven Alias Analysis for C. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '08)*. ACM, 197–208.