Dynamic Complexity of the Dyck Reachability

Patricia Bouyer(✉) and Vincent Jugé(✉)

LSV, CNRS & ENS Cachan, Univ. Paris-Saclay, Cachan, France
{bouyer,juge}@lsv.fr

Abstract. Dynamic complexity is concerned with updating the output of a problem when the input is slightly changed. We study the dynamic complexity of Dyck reachability problems in directed and undirected graphs, where updates may add or delete edges. We show a strong dichotomy between such problems, based on the size of the Dyck alphabet. Some of them are P-complete (under a strong notion of reduction) while the others lie either in DynFO or in NL.

1 Introduction

Dynamic problems and dynamic complexity. In this paper, we focus on the dynamic complexity of some reachability problems. Standard complexity theory aims at developing algorithms that, given an input of some problem, compute an output as efficiently as possible. Its dynamic variant is focused on algorithms that are capable of efficiently updating the output after a small change of the input [11,16,17]. Such algorithms may rely on auxiliary data about the current instance of the problem, and update it when the instance is modified.

A well-studied dynamic complexity class is DynFO. An algorithm is in DynFO if the output and the auxiliary data can be updated by FO formulas after a small change of the input. Variants of DynFO include the class DynFO^+, which allows polynomial-time precomputations, and DynTC^0, in which updates of the auxiliary data are performed by TC^0 circuits.

Consider the problem of reachability in directed graphs, and update operations that consist in inserting or deleting edges (one at a time). It was recently proven that this problem belongs to the class DynFO [3], which had been conjectured for decades.

Furthermore, like static complexity classes, dynamic complexity classes come with natural notions of reduction. The class DynFO is closed under bounded expansion first-order reductions (hereafter called bfo reductions), which are specific L reductions (L is for logarithmic space). A bfo reduction from a problem to another one is a first-order mapping from instances of the first problem to instances of the latter one, such that performing an update operation on the instance of the first problem amounts to performing a bounded number of update operations on the instance of the latter problem. Similarly, the class DynFO^+ is closed under bounded expansion first-order reductions with polynomial-time precomputation (hereafter called bfo^+ reductions).

This work is supported by EU under ERC EQualIS (FP7-308087).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 265–280, 2017.
DOI: 10.1007/978-3-662-54458-7_16

Reachability problems and language theory. Dyck reachability problems lie at the interface between two areas. On the one hand, language theory is concerned with handling descriptions of languages, that is sets of words, with respect to various questions: Is a language empty, finite or infinite? What about the intersection or the union of two languages? Does a language contain a given word? Among the best known and most simple classes of languages are regular and context-free languages. On the other hand, reachability problems deal with the existence of paths in graphs, and include questions such as: Does there exist a path between two given vertices? How long must be such paths?

Dyck reachability problems are focused on the existence of paths in labeled graphs, whose labels belong to a given *Dyck* language. Dyck languages are languages of well-parenthesized words and, roughly speaking, are the most simple context-free languages that are not regular. The Dyck reachability problem in labeled directed acyclic graphs was proven to be in DynFO [17], when considering two types of update operations on labeled graphs, which are insertion and deletion of edges. Whether this result extends to all labeled directed graphs was then an open question.

Our contributions. We study this open question, and we distinguish the Dyck reachability problem in two different ways. Is the labeled graph directed or undirected? How many symbols does the Dyck alphabet contain?

We prove that there exists a strong dichotomy between the dynamic complexity of such problems, based on the size of the Dyck alphabet. In the case of a unary Dyck alphabet, the Dyck reachability problem lies in NL (nondeterministic logarithmic space), and even lies in DynFO in the case of undirected graphs; this contrasts with the case of binary Dyck alphabets, where we prove that the Dyck reachability problem is P-complete under bfo^+ reductions. Furthermore, it is widely believed [16] that no P-complete problems under bfo^+ reductions lie in classes such as DynFO or the slightly broader class DynFO^+.

Related works. From its very inception 20 years ago, dynamic complexity has been a framework of study for several variants of reachability problems and language theory problems. The class DynFO was shown to contain reachability problems in directed acyclic graphs [5], undirected graphs [16] and, most recently, in all directed graphs [3]; regular and Dyck languages [16], then all context-free languages [6]; Dyck reachability in directed acyclic graphs [17].

At the same time, finding natural problems that are NL- or P-complete (under L reductions) and belong to low dynamic complexity classes such as DynFO, DynFO^+ or DynTC^0 is an ongoing challenge. All known P-complete problems lying in DynFO rely on highly redundant inputs, hence may be seen as artificial [16]. Hence, a notion of *non-redundant projection* [14] was introduced. Non-redundant projections are a special kind of P reductions, which contains, in particular, bfo and bfo^+ reductions.

Hence, for every static complexity class $\mathcal{C}$, we define *non-redundant* $\mathcal{C}$-complete problems as those problems that are $\mathcal{C}$-complete both under L reductions and under non-redundant projections. Most canonical P-complete problems

are non-redundant, hence non-redundancy may be seen as a prerequisite for being a "natural" problem.

A breakthrough was the proof that the Dyck reachability problem in acyclic directed graphs, which is a non-redundant LogCFL-complete problem, belongs to DynFO [17]. It was then proved in [15] that the reachability problem in labeled acyclic graphs, where path labels are constrained to belong to a given context-free grammar (and not only to a Dyck language), is in DynFO. We prove here that the results of [15] are unlikely to extend to all labeled graphs, or even to undirected graphs, even in the simple case of two-letter Dyck languages. This also allows us to answer negatively a question of Weber and Schwentick, who asked in [17] whether "the Dyck reachability problem might be a non-redundant P-complete problem that allows efficient updates."

Complete proofs can be found in research report [2].

2 Definitions

2.1 Dyck Reachability Problems

A labeled directed graph is a triple $G = (V, L, E)$ where V is a finite set of vertices, L is a finite set of labels and $E \subseteq V \times L \times V$ is a finite set of edges. The graph G is said to be *unlabeled* if L is a singleton set; in that case, we may directly represent G as a pair (V, E) where $E \subseteq V \times V$. The graph G is also said to be *undirected* if the relation E is symmetric, i.e. if, for every edge (v, θ, w) in E, the triple (w, θ, v) also belongs to E.

A path in the graph G is a finite sequence of edges $\pi = (v_1, \theta_1, w_1) \cdot (v_2, \theta_2, w_2) \cdot \ldots \cdot (v_k, \theta_k, w_k)$ such that $v_{i+1} = w_i$ for all $i \in \{1, \ldots, k-1\}$. The vertex v_1 is called the *source* of π, and w_k is called the *sink* of π. We also denote by $\lambda(\pi)$ the word $\theta_1 \cdot \ldots \cdot \theta_k$, which is called the *label* of π.

Assume that the label set L is of the form $L = \{\ell_1, \ldots, \ell_n\} \uplus \{\overline{\ell}_1, \ldots, \overline{\ell}_n\}$ for some integer $n \geq 1$. The *Dyck language* (also called *semi-Dyck language* in [7]) associated with L is the context-free language $\mathbf{D}_n$ built over the grammar: $S \to \varepsilon \mid \ell_1 \cdot S \cdot \overline{\ell}_1 \cdot S \mid \ldots \mid \ell_n \cdot S \cdot \overline{\ell}_n \cdot S$, where ε is the empty word. The set $\{\ell_1, \ldots, \ell_n\}$ is said to be the *Dyck alphabet* of that language.

The *n-letter Dyck reachability problem* asks whether, given two vertices s and t of G, there exists a path in G, with source s and sink t, and whose label belongs to the Dyck language $\mathbf{D}_n$ (the actual value of the label set L does not matter, as long as its elements can be partitioned in n ordered pairs). The *n-letter undirected* Dyck reachability problem is the restriction of that problem to the case where the underlying graph G is constrained to be undirected.

2.2 Dynamic Complexity

In this paper, we study the dynamic complexity of Dyck reachability problems. To that end, we first introduce briefly the formalisms of descriptive and dynamic complexity here, and refer to [10,13,16] for more details.

Descriptive complexity aims at characterizing positive instances of a problem using logical formulas. The input is then described as a logical structure described by a set of k-ary predicates (the *vocabulary*) over its universe. For example, a graph can be described by a binary predicate representing its edges, with the set of vertices (usually identified with $\{1, \dots, n\}$ for some n) as the universe. The problem of deciding whether some state has at most one outgoing edge can be described by the first-order formula $\exists x. \forall y. \forall z. (E(x, y) \wedge E(x, z)) \Rightarrow (y = z)$. The class FO contains all problems that can be characterized by such first-order formulas. This class corresponds to the circuit-complexity class AC^0 (under adequate uniformity assumptions) [1].

Dynamic complexity aims at developing algorithms that can efficiently update the output of a problem when the input is slightly changed, for example reachability of one vertex from another one in a graph. We would like our algorithm to take advantage of previous computations in order to very quickly decide the existence of a path in the modified graph.

Formally, a decision problem S is a subset of the set of τ-structures $\mathsf{Struct}(\tau)$ built on a vocabulary τ. In order to turn S into a dynamic problem DynS, we need to define a finite set of allowed updates. For instance, we might use a 2-ary operator $\mathsf{ins}(x, y)$ that would insert an edge between nodes x and y. For a universe of size n, the set of update operations forms a finite alphabet, denoted by Σ_n. A finite word in Σ_n^* then corresponds to a structure obtained by applying a sequence of update operations of Σ_n to the empty structure $\mathcal{I}_n$ over the vocabulary τ. The language DynS_n is defined as the set of those words in Σ_n^* that correspond to structures of S, and DynS is the union (over all n) of all such languages.

A dynamic machine is a uniform family $(M_n)_{n \in \mathbb{N}}$ of deterministic finite automata $M_n = \langle Q_n, \Sigma_n, \delta_n, s_n, F_n \rangle$ over an update alphabet Σ_n, with an update transition function δ_n. Every state is a polynomial-size auxiliary data structure over some vocabulary τ^{aux}, which contains the vocabulary τ. Such a machine solves a dynamic problem if $\mathsf{DynS}_n = \mathcal{L}(M_n)$ for all n. It is in the dynamic complexity class $\mathcal{C}'$-$\mathsf{Dyn}\mathcal{C}$ (or simply $\mathsf{Dyn}\mathcal{C}$ if $\mathcal{C} = \mathcal{C}'$) if the update transition function and membership in the accepting set can be computed in $\mathcal{C}$, while the initial state can be computed in $\mathcal{C}'$. In other words, solving the initial instance of the problem and computing initial auxiliary data structure can be done in $\mathcal{C}'$, and after any update of the input (specified by some letter of Σ_n), further calculations to solve the problem and update the auxiliary data on that new instance are restricted to the class $\mathcal{C}$. Of course, for a dynamic complexity class $\mathcal{C}'$-$\mathsf{Dyn}\mathcal{C}$ to have some interest, the class $\mathcal{C}$ should be easier than the static complexity class of the original problem.

In this paper, we only consider the case where $\mathcal{C} = \mathsf{FO}$, and where $\mathcal{C}' = \mathsf{FO}$ or $\mathcal{C}' = \mathsf{P}$, meaning that first-order formulas will be used to describe how predicates are updated along transitions, and that we may make use of polynomial-time precomputations. As a convention, we will denote the class P-DynFO by DynFO^+, and we recall that the simple notation DynFO is for FO-DynFO.

2.3 Dynamic Reductions

Dynamic complexity comes with the notion of dynamic reductions [16]. Let $\mathcal{C}$ be a complexity class. A (static) $\mathcal{C}$ reduction from a decision problem $\mathcal{P}$ to another decision problem $\mathcal{Q}$ is a mapping in $\mathcal{C}$ from the instances of $\mathcal{P}$ to the instances of $\mathcal{Q}$ that associates every positive instance of $\mathcal{P}$ with a positive instance of $\mathcal{Q}$, and every negative instance of $\mathcal{P}$ with a negative instance of $\mathcal{Q}$. Standard P-completeness results use L reductions [7].

A *dynamic reduction* from a dynamic problem $\mathcal{P}$ (with vocabulary τ_1) to another dynamic problem $\mathcal{Q}$ (with vocabulary τ_2) is a mapping from $\mathsf{Struct}(\tau_1)$ to $\mathsf{Struct}(\tau_2)$ such that:

- every positive (respectively, negative) instance of $\mathcal{P}$ is mapped to a positive (respectively, negative) instance of $\mathcal{Q}$;
- every update on an instance i_1 of $\mathcal{P}$ results in a *well-behaved* sequence of updates on the instance i_2 of $\mathcal{Q}$ to which i_1 is mapped.

Dynamic reductions have therefore several parameters: the complexity class to which the mapping belongs, and the sequences of updates that are allowed.

The dynamic classes DynFO and DynFO^+ are respectively closed under bounded expansion first-order (bfo for short) and bounded expansion first-order with polynomial-time precomputation (bfo^+ for short) reductions [16]. A dynamic reduction μ from $\mathcal{P}$ to $\mathcal{Q}$ is bfo^+ if it is a FO reduction and if every update on an instance i_1 of $\mathcal{P}$ results in a bounded sequence of FO updates on its image $\mu(i_1)$. If, furthermore, the empty structure $\mathcal{I}_1$ is mapped to a structure $\mu(\mathcal{I}_1)$ that can be obtained by applying a bounded sequence of FO updates on the empty structure $\mathcal{I}_2$, then we say that μ is bfo.

Note that dynamic reductions can be applied to the class P (which coincides with the class DynP, under the assumption that updates are one-bit input changes). So, being P-hard for bfo^+ reductions is arguably stronger than being P-hard for L reductions. Furthermore, it is known that the classes of bfo and of bfo^+ reductions are closed under composition and that the circuit value problem is a P-complete problem for bfo^+ reductions [16]. Hence, every P problem to which the circuit value problem is bfo^+-reducible is also P-complete problem for bfo^+ reductions.

2.4 Main Result

We are now in a position to formally present our main result.

Theorem 1. *The* 1*-letter Dyck reachability problem is in* NL*, and the* 1*-letter undirected Dyck reachability problem is in* $\mathsf{NL} \cap \mathsf{DynFO}$*. Furthermore, for all integers* $n \geq 2$*, both the* n*-letter Dyck reachability problem and the* n*-letter undirected Dyck reachability problem are* P*-complete for* bfo^+ *reductions.*

Remark 1. Note that $\mathsf{NL} \cap \mathsf{DynFO}$ is not known to be strictly included in NL. Nevertheless, the case of undirected graph appears to be "easier" than the case of directed graphs in the 1-letter case. Hence, the P-hardness of both cases for alphabets with at least two letters appears rather unexpected.

3 One-Letter (Undirected) Dyck Reachability Problems

We prove here the first part of Theorem 1, that is we assume $n = 1$. We first observe that the 1-letter Dyck reachability problem is equivalent to a standard reachability problem in one-counter automata (without zero-tests), which is known to belong to NL [4,8]. The 1-letter undirected Dyck reachability problem is a restriction of the 1-letter directed Dyck reachability problem, hence it is in NL as well. Furthermore, we make the following claim.

Proposition 1. *Let s and t be two distinct vertices of an undirected labeled graph $G = (V, E, L)$, with $L = \{\ell_1, \overline{\ell}_1\}$. There exists a Dyck path from s to t in G if and only if:*

- *the set $\{x \in V \mid (s, \ell_1, x) \in E\}$ is non-empty;*
- *the set $\{y \in V \mid (t, \overline{\ell}_1, y) \in E\}$ is non-empty;*
- *there exists a path of even length from s to t in G.*

Proof. First, if there exists a Dyck path $\pi = (v_i, \lambda_i, v_{i+1})_{0 \leq i < k}$ with $s = v_0$ and $t = v_k$, then $\lambda_0 = \ell_1$, $\lambda_{k-1} = \overline{\ell}_1$, and the sets $\{0 \leq i < k \mid \lambda_i = \ell_1\}$ and $\{0 \leq i < k \mid \lambda_i = \overline{\ell}_1\}$ have the same cardinality, which proves that k is an even number.

Conversely, assume that $s \neq t$ and that the three conditions of Proposition 1 hold. Let $\pi = (v_i, \lambda_i, v_{i+1})_{0 \leq i < 2k}$ be a path of length $2k$ from s to t in G, for some integer $k \geq 1$. Let κ be the cardinality of the set $\{0 \leq i < 2k \mid \lambda_i = \ell_1\}$ and let $\overline{\kappa}$ be the cardinality of the set $\{0 \leq i < 2k \mid \lambda_i = \overline{\ell}_1\}$. Since $\kappa + \overline{\kappa} = 2k$, we have $\overline{\kappa} - \kappa = 2(k - \kappa)$.

Furthermore, consider vertices $x, y \in V$ such that (s, ℓ_1, x) and $(t, \overline{\ell}_1, y)$ belong to E. Since the graph is undirected, there exist also edges (x, ℓ_1, s) and $(y, \overline{\ell}_1, t)$. Let ρ_1 be the length-2 circuit $(s, \ell_1, x) \cdot (x, \ell_1, s)$, and let ρ_2 be the length-2 circuit $(t, \overline{\ell}_1, y) \cdot (y, \overline{\ell}_1, t)$. One checks easily that the path $\rho_1^k \cdot \pi \cdot \rho_2^\kappa$ is a Dyck path in G, where ρ_1^k is the concatenation of k occurrences of ρ_1, and ρ_2^κ is the concatenation of κ occurrences of ρ_2. □

Hence, checking whether there exists a Dyck path from s to t in G amounts to checking whether $s = t$ or, if $s \neq t$, whether the sets $\{x \in V \mid (s, \ell_1, x) \in E\}$ and $\{y \in V \mid (t, \overline{\ell}_1, y) \in E\}$ are non-empty, and whether there exists a path of even length from s to t in G. The first statements can be checked directly in FO, and the latter one can be checked in DynFO, as proved below. This completes the proof of the first part of Theorem 1 in the case $n = 1$.

Lemma 1. *Checking whether there exists a path of even length from s to t in G is feasible in* DynFO.

Proof. Let Γ be the graph $(G \times \{0, 1\}, E')$, where $E' = \{((x, 0), \ell, (y, 1)) \mid (x, \ell, y) \in G\} \cup \{((x, 1), \ell, (y, 0)) \mid (x, \ell, y) \in G\}$. The graph Γ consists in two copies of G, and edges of G translate into edges between these two copies. Since Γ is undirected, the reachability problem in Γ is in DynFO [3,16]. Furthermore, there exists a path of even length from s to t in G if and only if there exists

a path from $(s,0)$ to $(t,0)$ in Γ. Since Γ is FO-definable in terms of G, and since adding/deleting one edge in G amounts to adding/deleting two edges in Γ, Lemma 1 follows. □

Remark 2. Note that this proof heavily relies on the property that the graph is undirected. In fact, the problem of computing distances in directed graphs, whose membership in DynFO or DynFO^+ is a long-standing open question [3,9], is bfo^+-reducible to the 1-letter Dyck reachability problem (over directed graphs). The reduction is as follows.

Given an unlabeled directed graph $G = (V, E)$, equip each edge with a label ℓ_1, and add self-loops (with the label ℓ_1) around each vertex in V. Then, for all vertices $v \in V$, add n vertices $(v,1), \ldots, (v,n)$, where $n = |V|$, and add edges with the label $\overline{\ell}_1$ from v to $(v,1)$ and from (v,i) to $(v,i+1)$, for all i. It comes at once that the distance (in the original graph G) from a vertex s to a vertex t is k if and only if there exists a Dyck path (in the extended, labeled graph) from s to (t,k) but not to $(t,k-1)$.

Furthermore, the proof of [9] showing that distances in graphs can be computed in DynTC^0 does not extend to the 1-letter Dyck reachability, whose precise dynamic complexity remains therefore unknown.

4 *n*-letter Dyck Reachability Problem

We prove now that, for all integers $n \geq 2$, the n-letter Dyck reachability problem is P-complete for bfo^+ reductions.

We first introduce two auxiliary problems.

1. Let $G = (V, E)$ be an unlabeled directed graph, let $(V_\wedge, V_\vee)$ be a partition of V, and let s and t be two marked vertices of G. The *alternating reachability problem* asks whether s belongs to the *alternating coaccessible set* of t, i.e. the smallest subset X of V such that all of $\{t\}$, $\{x \in V_\vee \mid \exists y \in X \text{ s.t. } (x,y) \in E\}$ and $\{x \in V_\wedge \mid \forall y \in V, (x,y) \in E \Rightarrow y \in X\}$ are subsets of X.
   Note that this problem could be alternatively and equivalently defined using the notion of winning state in a two-player turn-based zero-sum reachability game. However, we choose the above definition using a fixed point to avoid defining the notion of winning strategies.
2. Let $G = (V, E, L)$ be a labeled directed graph with set of labels $L = V \cup \{\overline{v} \mid v \in V\}$, and let s and t be two marked vertices of G. A *near-Dyck word* is an element of the set $\mathbf{D'}$ built over the grammar: $S \to \varepsilon \mid v \cdot S \cdot \overline{v} \cdot S$ (for all $v \in V$). The *near-Dyck reachability problem* asks whether there exists a path π in G, with source s, sink t, and whose label belongs to $\mathbf{D'}$.
   Note that the near-Dyck reachability problem may be viewed as a generalisation of the n-letter Dyck reachability problem: it involves a grammar with $|V|$ rules and not only n, i.e. the size of the grammar is not constant anymore.

While it is well-known that the alternating reachability problem is P-hard for standard logarithmic-space reductions, it is also the case that it is P-hard

for bfo^+ reductions [16]. Hence, we show in the two next subsections that there exists a bfo^+ reduction from the alternating reachability problem to the near-Dyck reachability problem, and that there exists a bfo^+ reduction from that latter problem to the 2-letter Dyck reachability problem. It will follow that the 2-letter (and, therefore, the n-letter) Dyck reachability problem is P-hard for bfo^+ reductions.

On the other hand, it is known that the n-letter Dyck reachability problem belongs to P (see [7, Sect. A.7.9]).

4.1 From the Near-Dyck Reachability Problem to the Dyck Reachability Problem

Let $G = (V, E, L)$ be a labeled directed graph with set of labels $L = V \cup \overline{V}$ (where $\overline{V} = \{\overline{v} \mid v \in V\}$), and let s and t be two marked vertices of G.

We fix the new alphabet $\mathcal{L} = \{0, 1, \overline{0}, \overline{1}\}$. Then, we consider an integer n and an injective *coding* function $\mathsf{cod} : (V \cup \overline{V}) \mapsto \mathcal{L}^n$ such that $\mathsf{cod}(V) \subseteq \{0,1\}^n$, $\mathsf{cod}(\overline{V}) \subseteq \{\overline{0}, \overline{1}\}^n$. We further assume the following consistency requirement about cod: for all $v \in V$ and all $i \in \{1, \dots, n\}$, we have $\mathsf{cod}(\overline{v})_i = \overline{\mathsf{cod}(v)_{n+1-i}}$, where we denote by w_i the i^{th} letter of the word $w \in \mathcal{L}^n$.

Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ be the labeled directed graph defined by:

- $\mathcal{V} = V \cup \big(V \times (V \cup \overline{V}) \times \{0, 1, \dots, n\}\big)$;
- $\mathcal{L} = \{0, 1, \overline{0}, \overline{1}\}$;
- $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, where

$$\mathcal{E}_1 = \{x \xrightarrow{0} (x, v, 0) \mid x, v \in V\} \cup \{x \xrightarrow{\overline{0}} (x, \overline{v}, 0) \mid x \in V, \overline{v} \in \overline{V}\} \cup \{(x, v, i) \xrightarrow{\mathsf{cod}(v)_{i+1}} (x, v, i+1) \mid x \in V, v \in V \cup \overline{V}, 0 \leq i \leq n-1\} \text{ and}$$

$$\mathcal{E}_2 = \{(x, v, n) \xrightarrow{0} y \mid x \xrightarrow{v} y \in E\} \cup \{(x, \overline{v}, n) \xrightarrow{\overline{0}} y \mid x \xrightarrow{\overline{v}} y \in E\},$$

and in which we mark the vertices s and t.

Each sequence of transitions $x \xrightarrow{0} (x, v, 0) \xrightarrow{v_1} \dots \xrightarrow{v_n} (x, v, n)$ prepares the encoding of some edge leaving x with label v. If there is some edge $x \xrightarrow{v} y$ in the original graph, then only one edge $(x, v, n) \xrightarrow{0} y$ needs to be added: this is the role of the edges in $\mathcal{E}_2$. We use a similar encoding for edges labeled by $\overline{v}$.

Proposition 2. *There exists a near-Dyck path from s to t in G if and only if there exists a Dyck path from s to t in $\mathcal{G}$.*

Proof. First, for every pair $(u, v) \in \mathcal{V}^2$, there exists at most one edge in $\mathcal{E}$ with source u and sink v. Henceforth, we omit representing labels of edges and of paths in $\mathcal{G}$.

We further define two mappings φ and ψ. The mapping φ identifies every label $\lambda \in L$ with a word $\varphi(\lambda) \in \mathcal{L}^*$, as follows:

$$\varphi(v) = 0 \cdot \mathsf{cod}(v) \cdot 0 \text{ for all } v \in V, \text{ and } \varphi(\overline{v}) = \overline{0} \cdot \mathsf{cod}(\overline{v}) \cdot \overline{0} \text{ for all } \overline{v} \in \overline{V}.$$

This mapping extends immediately to a morphism from L^* to $\mathcal{L}^*$ that maps every near-Dyck word $w \in \mathbf{D}'$ to a Dyck word $\varphi(w) \in \mathbf{D}$. The mapping ψ identifies every edge $e \in E$ with a path $\psi(e)$ in $\mathcal{G}$, as follows:

$$\psi(x \xrightarrow{v} y) = (x \to (x, v, 0) \to \ldots \to (x, v, n) \to y) \text{ for all } v \in V \cup \overline{V}.$$

This mapping extends immediately to a morphism that maps every path in G to a path in $\mathcal{G}$. The relation $\lambda(\psi(e)) = \varphi(\lambda(e))$ holds for all edges $e \in E$, and therefore extends to all paths π in G. Hence, a path π in G is near-Dyck if and only if the path $\psi(\pi)$ in $\mathcal{G}$ is Dyck.

In addition, let us call *nominal* paths in $\mathcal{G}$ the paths that belong to the set $\{\psi(e) \mid e \in E\}$, and *generic* paths in $\mathcal{G}$ the concatenations of nominal paths. Nominal paths are the minimal paths whose source and sink both belong to the subset V of $\mathcal{V}$. Hence, every path π from s to t in $\mathcal{G}$ is generic, thus π is the image by ψ of some path $\psi^{-1}(\pi)$ from s to t in G. □

The graph $\mathcal{G}$ is FO-definable as a function of G and of the coding function cod, and adding/deleting an edge in E amounts to adding/deleting exactly one edge in $\mathcal{E}_2$. Since the function cod can be precomputed in P, and due to Proposition 2, the near-Dyck reachability problem is therefore bfo^+-reducible to the Dyck reachability problem.

4.2 From the Alternating Reachability Problem to the Near-Dyck Reachability Problem

Let $G = (V, E)$ be an unlabeled directed graph, let $(V_\wedge, V_\vee)$ be a partition of V, let s and t be two marked vertices of G.

Let us number the vertices of G from 0 to $n-1$, i.e. set $V = \{v_0, \ldots, v_{n-1}\}$. Then, let $\mathbf{G}$ be the context-free grammar with set of non-terminal symbols V and initial symbol s, without terminal symbol, and that consists in three kinds of rules:

- a termination rule $t \to \varepsilon$;
- rules $v \to w$ for all vertices $v \in V_\vee$ and $w \in V$ such that $(v, w) \in E$;
- rules $v \to w_0 \cdot w_1 \cdots w_{n-1}$ for all vertices $v \in V_\wedge$, where $w_i = v_i$ if $(v, v_i) \in E$ and $w_i = t$ otherwise.

The following result is straightforward.

Proposition 3. *Let X be the alternating coaccessible set of t. The vertex s belongs to X if and only if the language generated by $\mathbf{G}$ is non-empty.*

Inspired by the translation of context-free grammars into pushdown automata (by simulating leftmost derivations, see for instance [12, Theorem 6.13]), we build below a labeled graph whose labels correspond to push and pop moves of such a pushdown automaton, so that near-Dyck paths in the new graph will correspond to the empty-stack accepting runs of the pushdown automaton.

Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ be the labeled directed graph defined by:

- $\mathcal{V} = \{\circ, \bullet\} \cup V \cup \big(V_\wedge \times \{1, \dots, n-1\}\big)$, where $\circ$ and $\bullet$ are two fresh vertex symbols;
- $\mathcal{L} = V \cup \{\overline{v} \mid v \in V\}$;
- $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, where

$$\mathcal{E}_1 = \{\circ \xrightarrow{s} \bullet\} \cup \{\bullet \xrightarrow{\overline{x}} x \mid x \in V\} \cup \{\bullet \xrightarrow{\overline{t}} \bullet\} \text{ and}$$

$$\mathcal{E}_2 = \{x \xrightarrow{y} \bullet \mid x \in V_\vee \text{ and } (x,y) \in E\} \cup$$
$$\{(x, n-1-i) \xrightarrow{v_i} (x, n-i) \mid x \in V_\wedge, 0 \le i \le n-1 \text{ and } (x, v_i) \in E\} \cup$$
$$\{(x, n-1-i) \xrightarrow{t} (x, n-i) \mid x \in V_\wedge, 0 \le i \le n-1 \text{ and } (x, v_i) \notin E\},$$

where we use $(v, 0)$ as a placeholder for v and (v, n) as a placeholder for $\bullet$ (for all vertices $v \in V_\wedge$). Then, let us mark vertices $\circ$ and $\bullet$.

The construction is illustrated in Fig. 1, in which the graph G is associated with the grammar $\mathbf{G}$ whose initial symbol is v_0 and whose rules are $v_3 \to \varepsilon \mid v_1 \to v_2 \mid v_1 \to v_4 \mid v_2 \to v_3 \mid v_3 \to v_1 \mid v_0 \to v_3 v_1 v_3 v_3 v_4 \mid v_4 \to v_3 v_1 v_3 v_3 v_3$.

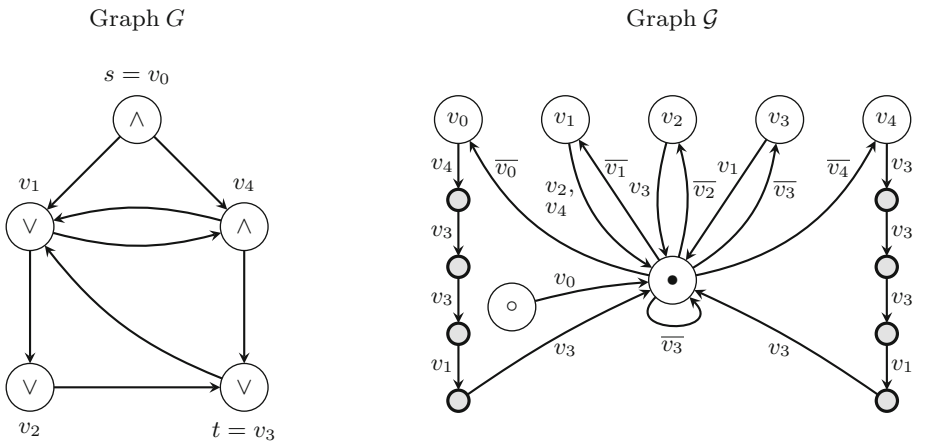

Fig. 1. Graphs G and $\mathcal{G}$

Proposition 4. *There exists a near-Dyck path from $\circ$ to $\bullet$ in $\mathcal{G}$ if and only if the language of the context-free grammar $\mathbf{G}$ is non-empty.*

Proof. Consider the pushdown automaton $\mathcal{A} = (Q, \Sigma, \delta, \iota, F)$, with set of states $Q = \mathcal{V}$, input alphabet $\Sigma = \emptyset$, initial state $\iota = \circ$, set of final states $F = \{\bullet\}$, whose transition function δ contains only the following ε-transitions:

$$\delta : q \xrightarrow{\text{push}(v)} q' \text{ for all } q, q' \in \mathcal{V} \text{ and } v \in V \text{ s.t. } (q, v, q') \in \mathcal{E}$$
$$q \xrightarrow{\text{pop}(v)} q' \text{ for all } q, q' \in \mathcal{V} \text{ and } v \in V \text{ s.t. } (q, \overline{v}, q') \in \mathcal{E},$$

and whose accepting runs are those that start in ι with an empty tape and end in a final state (i.e. in $\bullet$) with an empty tape.

It is straightforward that $\mathcal{A}$ accepts the language of $\mathbf{G}$, for instance by observing that it follows the construction of [12, Theorem 6.13]. Furthermore, near-Dyck paths from $\circ$ to $\bullet$ in $\mathcal{G}$ can be identified with stack operations of accepting executions of $\mathcal{A}$. Proposition 4 follows. □

The graph $\mathcal{G}$ is FO-definable as a function of G, s, t and of the mapping $i \mapsto v_i$. Moreover, adding/deleting an edge e in E amounts to adding/deleting exactly either one or two edges in $\mathcal{E}_2$. Since the mapping $i \mapsto v_i$ can be precomputed in P, and due to Propositions 3 and 4, the alternating reachability problem is therefore bfo^+-reducible to the near-Dyck reachability problem.

5 n-letter Undirected Dyck Reachability Problem

We proceed by proving that there exists a bfo^+ reduction from the 2-letter Dyck reachability problem (in directed graphs) to the 2-letter undirected reachability problem.

Let $G = (V, E, L)$ be a directed labeled graph, with $L = \{\ell_1, \ell_2, \overline{\ell}_1, \overline{\ell}_2\}$, and let s and t be two marked nodes of G. In addition, let $\mathcal{L} = \{0, 1, \overline{0}, \overline{1}\}$ be another set of labels.

The main difficulty, when working in an undirected graph, is that lots of cycles are created, generating lots of stuttering in the words labeling the paths. It is therefore hard to really control where a path goes just by looking at its label. In particular, the recipe used in Sect. 4.1 to reduce the near-Dyck reachability problem to the 2-letter Dyck reachability problem cannot be used now, and it is not clear whether simple alternative reductions from the near-Dyck undirected reachability problem to the 2-letter undirected Dyck reachability problem exist. Below, we prove directly the P-hardness of the 2-letter undirected Dyck reachability. In order to do so, we rely on a rather intricate encoding, where each part plays an important role.

We denote by $\varphi : L^* \mapsto \mathcal{L}^*$ the homomorphism of monoids defined by:

$$\varphi(\ell_1) = 0 \cdot \boxed{\overline{0} \cdot 1} \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot \boxed{\overline{1} \cdot 0} \qquad \varphi(\overline{\ell}_1) = \boxed{\overline{0} \cdot 1} \cdot \overline{1} \cdot \overline{1} \cdot \overline{1} \cdot \overline{1} \cdot \overline{0} \cdot \overline{0} \cdot \overline{1} \cdot \boxed{\overline{1} \cdot 0} \cdot \overline{0}$$

$$\varphi(\ell_2) = 0 \cdot \boxed{\overline{0} \cdot 1} \cdot 0 \cdot 0 \cdot 1 \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot \boxed{\overline{1} \cdot 0} \qquad \varphi(\overline{\ell}_2) = \boxed{\overline{0} \cdot 1} \cdot \overline{1} \cdot \overline{0} \cdot \overline{0} \cdot \overline{1} \cdot \overline{1} \cdot \overline{0} \cdot \overline{0} \cdot \boxed{\overline{1} \cdot 0} \cdot \overline{0}$$

Observe that the words $\varphi(\overline{\ell}_1)$ and $\varphi(\overline{\ell}_2)$ are formal inverses of the words $\varphi(\ell_1)$ and $\varphi(\ell_2)$: in particular, both the words $\varphi(\ell_1) \cdot \varphi(\overline{\ell}_1)$ and $\varphi(\ell_2) \cdot \varphi(\overline{\ell}_2)$ are Dyck words.

In gray boxes are locks: *along a Dyck path*, once a lock has been traveled through, we cannot go back earlier in the encoding, since this would create a factor $1 \cdot \overline{0}$ or $0 \cdot \overline{1}$, which is not a factor of any Dyck word. We therefore say that a path is *doomed* if it crosses a lock backwards, thereby having a factor $1 \cdot \overline{0}$ or $0 \cdot \overline{1}$. By preventing Dyck paths from having doomed subpaths, locks will allow us to recover partially the directed character of G.

Finally, for every word $w \in \mathcal{L}^*$, we denote by w_i the i^{th} letter of w. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ be the undirected labeled graph defined by:

- $\mathcal{V} = V \cup (V \times L \times V \times \{1, \dots, 11\})$;
- $\mathcal{E} = \mathcal{E}_{\mathsf{init}} \cup \mathcal{E}_{\mathsf{mid}} \cup \mathcal{E}_{\mathsf{end}}$, where

$$\begin{aligned}
\mathcal{E}_{\mathsf{init}} &= \{x \xleftrightarrow{\varphi(\lambda)_1} (x, \lambda, y, 1) \mid (x, \lambda, y) \in E\} \\
\mathcal{E}_{\mathsf{mid}} &= \{(x, \lambda, y, i-1) \xleftrightarrow{\varphi(\lambda)_i} (x, \lambda, y, i) \mid (x, \lambda, y) \in E, 1 \leq i \leq 11\} \\
\mathcal{E}_{\mathsf{end}} &= \{(x, \lambda, y, 11) \xleftrightarrow{\varphi(\lambda)_{12}} y \mid (x, \lambda, y) \in E\},
\end{aligned}$$

and in which we mark the vertices s and t.

Like in Sects. 4.1 and 4.2, we observe an equivalence between the two kinds of Dyck reachability problems in the graphs G and $\mathcal{G}$, which we prove formally in the rest of the section.

Proposition 5. *There exists a Dyck path from s to t in G if and only if there exists a Dyck path from s to t in $\mathcal{G}$.*

A first *completeness* result towards proving Proposition 5 comes quickly.

Lemma 2. *Let ρ be a Dyck path from s to t in G. There exists a Dyck path from s to t in $\mathcal{G}$.*

Proof. First, for every pair $(u, v) \in \mathcal{V}^2$ there exists at most one undirected edge between u and v in $\mathcal{E}$. Henceforth, we may omit representing labels of edges and of paths in $\mathcal{G}$.

Now, let us denote by ψ the mapping that identifies every edge $e \in E$ with the path $\psi(e) = (x \to (x, \theta, y, 1) \to \dots \to (x, \theta, y, 11) \to y)$ in $\mathcal{G}$, where $e = (x, \theta, y)$. Observe that ψ extends immediately to a morphism that maps every path in G to a path in $\mathcal{G}$. The relation $\lambda(\psi(e)) = \varphi(\lambda(e))$ holds for all edges $e \in E$, and therefore extends to all paths in G. Hence, a path ρ in G is Dyck if and only if the path $\psi(\rho)$ in $\mathcal{G}$ is Dyck. □

However, unlike in Sect. 4.1, there may exist Dyck paths in $\mathcal{G}$ that are not of the form $\psi(\rho)$, as shown by the examples of the two Dyck cycles γ_1 and γ_2 displayed in Fig. 2. Consequently, we cannot use directly the morphism ψ to associate every Dyck path in $\mathcal{G}$ with a Dyck path in G.

We overcome this problem as follows. Let $\mathcal{Q}$ be the set of all factors of all Dyck words with letters in $\mathcal{L}$ (called *approximate Dyck words*), and let $\mathcal{P}$ the set of all paths π in $\mathcal{G}$ such that $\lambda(\pi) \in \mathcal{Q}$ (called *approximate Dyck paths*). Moreover, for every set S of paths, we denote by $\lambda(S)$ the set of labels of paths in S, i.e. $\lambda(S) = \{\lambda(\pi) \mid \pi \in S\}$. It comes at once that $\lambda(\mathcal{P}) \subseteq \mathcal{Q}$, that $\mathcal{Q}$ is factor closed, and that none of the words $1 \cdot \overline{0}$ nor $0 \cdot \overline{1}$ belongs to $\mathcal{Q}$.

We further say that a path in $\mathcal{G}$ is *nominal* if its source and sink belong to V, while its intermediate vertices belong to $\mathcal{V} \setminus V$. For all edges $(x, \lambda, y) \in E$, we denote by $\mathcal{P}_{x,\lambda,y}$ the set of nominal paths $\pi \in \mathcal{P}$ such that π has source x,

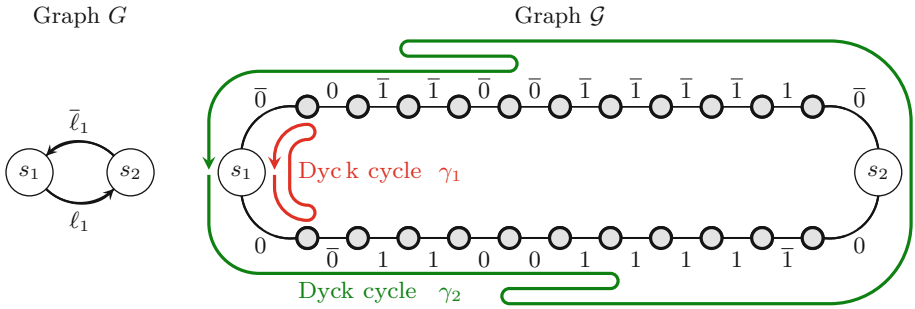

Fig. 2. Graphs G and $\mathcal{G}$, and Dyck cycle in $\mathcal{G}$

sink y, and such that its internal vertices are exactly the elements of the set $\{(x, \lambda, y, i) \mid 1 \leq i \leq 11\}$. For all vertices $x \in V$, we also denote by $\mathcal{P}_x$ the set of nominal paths $\pi \in \mathcal{P}$ such that π has source and sink x, and whose edges are all labeled with 0 or $\overline{0}$. These two classes of paths capture the entire family of nominal paths that belong to $\mathcal{P}$, as shown by the following result.

Lemma 3. *Let $\pi \in \mathcal{P}$ be a nominal path in $\mathcal{G}$. Either there exists an edge $(x, \lambda, y) \in E$ such that $\pi \in \mathcal{P}_{x,\lambda,y}$ or there exists a vertex $x \in V$ such that $\pi \in \mathcal{P}_x$. Moreover, the sets $\mathcal{P}_{x,\lambda,y}$ and $\mathcal{P}_x$ are pairwise disjoint.*

Proof. We first assume that some edge e in π is labeled by 1 or $\overline{1}$. By construction, there exists a unique edge $(x, \lambda, y) \in E$ and a unique pair of integers $i, j \in \{1, \ldots, 11\}$ such that $e = (x, \lambda, y, i) \to (x, \lambda, y, j)$, with $j = i \pm 1$. Since π is nominal, its internal vertices belong to the set $\{(x, \lambda, y, i) \mid 1 \leq i \leq 11\}$, and its source and sink belong to $\{x, y\}$. Then, since π belongs to $\mathcal{P}$, it does not contain doomed paths, hence its source must be x and its sink must be y.

Then, assume that no edge in π is labeled by 1 or $\overline{1}$. Since π is nominal, there exists a unique edge $(x, \lambda, y) \in E$ such that the internal vertices of π belong to the set $\{(x, \lambda, y, i) \mid 1 \leq i \leq 11\}$, and its source and sink belong to $\{x, y\}$. If x is the source of π, then π can never reach the vertex $(x, \lambda, y, 3)$, hence x is the sink of π; if y is the source of π, then π can never reach the vertex $(x, \lambda, y, 9)$, hence y is the sink of π.

Observing that every path in every set $\mathcal{P}_{x,\lambda,y}$ contains an edge labeled by 1 or $\overline{1}$ completes the proof. □

Going further, we associate with every path $\rho = (v_1, \lambda_1, w_1) \cdot \ldots \cdot (v_k, \lambda_k, w_k)$ in G the set $\overline{\mathcal{P}}_\rho$ of paths in $\mathcal{G}$ defined by:

$$\overline{\mathcal{P}}_\rho = \mathcal{P}^*_{v_1} \cdot \mathcal{P}_{v_1,\lambda_1,w_1} \cdot \mathcal{P}^*_{v_2} \cdot \mathcal{P}_{v_2,\lambda_2,w_2} \cdot \ldots \cdot \mathcal{P}^*_{v_k} \cdot \mathcal{P}_{v_k,\lambda_k,w_k} \cdot \mathcal{P}^*_{w_k}.$$

Observe that, unlike the sets $\mathcal{P}_x$ and $\mathcal{P}_{x,\lambda,y}$, the sets $\overline{\mathcal{P}}_\rho$ may contain paths that are not nominal and/or not approximate Dyck paths.

Conversely, however, it comes immediately that every Dyck path π in $\mathcal{G}$ belongs to one unique set $\overline{\mathcal{P}}_\rho$, where ρ is the *nominal ancestor* of π defined below.

Definition 1. *Let π be a Dyck path in $\mathcal{G}$ from s to t. There exists a unique sequence of vertices $v_0, \dots, v_k$, a unique partial function $f_\pi : \{1, \dots, k\} \mapsto L$, whose domain is denoted by $\mathsf{dom}(f_\pi)$, and a unique sequence of nominal paths $\pi_1, \dots, \pi_k$ such that:*

- *$v_0 = s$ and $v_k = t$;*
- *for all $i \in \mathsf{dom}(f_\pi)$, the edge $(v_{i-1}, f_\pi(i), v_i)$ belongs to E, and $\pi_i \in \mathcal{P}_{v_{i-1}, f_\pi(i), v_i}$;*
- *for all $i \in \{1, \dots, k\} \setminus \mathsf{dom}(f_\pi)$, we have $v_{i-1} = v_i$, and $\pi_i \in \mathcal{P}_{v_i}$;*
- *$\pi = \pi_1 \cdot \ldots \cdot \pi_k$.*

We call nominal vertex sequence *of π sequence $v_0, \dots, v_k$,* nominal label mapping *of π the mapping f_π,* nominal decomposition *of π the sequence $\pi_1, \dots, \pi_k$, and* nominal ancestor *of π the path $(v_{i-1}, f_\pi(i), v_i)_{i \in \mathsf{dom}(f_\pi)}$.*

Associating every Dyck path in $\mathcal{G}$ with a unique path in G is a first step towards proving the soundness of the construction. Further steps depend on the following, additional properties of the encoding.

Every Dyck path traveling through the word $\varphi(\ell_1)$, may go back and forth arbitrarily, except at locks, which it may cross only once. Consider such a possible journey through the word $\varphi(\ell_1)$, and observe the word w obtained during that journey. This word is made of blocks that consist, alternatively, of letters 0 and $\overline{0}$, and of letters 1 and $\overline{1}$. Such a word, *even if we first reduce it* (by deleting recursively the words $0 \cdot \overline{0}$ and $1 \cdot \overline{1}$), will always satisfy the following properties:

1. there exists at least four such (non-empty) blocks;
2. the last block consists of letters 0 only;
3. the last two blocks are of odd length, and every other block is of even length.

To illustrate the above analysis, consider the direct journey through $\varphi(\ell_1)$, where we have identified the blocks: $(0 \cdot \overline{0}) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1 \cdot 1 \cdot \overline{1}) \cdot (0)$. If that word is reduced, there remain four non-empty blocks: $(1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1) \cdot (0)$.

Another example is the journey first followed by the path γ_2 (see Fig. 2) from the vertex s_1 to the vertex s_2, and which gives us more blocks: $(0 \cdot \overline{0}) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1 \cdot 1 \cdot \overline{1}) \cdot (0)$. If that word is reduced, there remain six non-empty blocks: $(1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1) \cdot (0)$. One checks easily on these examples that all the words obtained satisfy the properties 1–3.

Properties 1–2 hold for the word $\varphi(\ell_2)$, while property 3 should be replaced by:

3'. the first block of letters 1 and $\overline{1}$ and the last block of letters 0 are of odd length, and every other block is of even length.

This distinction between encodings of the two letters will allow identifying a path that encodes ℓ_1 or ℓ_2, even when there is backtracking between the two locks.

Now, we denote by $\mathcal{Q}_{\mathsf{init}}$ the set of all prefixes of all Dyck words with letters in $\mathcal{L}$. Observe that, for all words $\rho_1, \rho_2 \in \mathcal{L}^*$, the three words $\rho_1 \cdot \rho_2$, $\rho_1 \cdot 0 \cdot \overline{0} \cdot \rho_2$ and $\rho_1 \cdot 1 \cdot \overline{1} \cdot \rho_2$ are either all Dyck or all non-Dyck.

Then, for every word $w \in \mathcal{L}^*$, we call *reduced word* of w the word $\mathsf{red}(w)$ obtained from w by deleting recursively the 2-letter words $0 \cdot \overline{0}$ or $1 \cdot \overline{1}$. Alternatively, if we consider w as an element of the free group generated by ℓ_1 and ℓ_2 (with inverses $\overline{\ell}_1$ and $\overline{\ell}_2$), then $\mathsf{red}(w)$ is the reduced word representing w. We just proved that w is Dyck if and only if $\mathsf{red}(w)$ is Dyck. Moreover, it comes immediately that $\mathcal{Q}_{\mathsf{init}}$ is in fact the set of all words w such that $\mathsf{red}(w)$ has only letters 0 and 1.

The above remarks and notions lead to the following results, whose proofs are then technical yet simple, and therefore omitted here.

Lemma 4. *Let ρ be a path in G. If ρ is not an approximate Dyck path, then $\lambda(\overline{\mathcal{P}}_\rho) \cap \mathcal{Q} = \emptyset$, and if $\lambda(\rho)$ is not a prefix of a Dyck word, then $\lambda(\overline{\mathcal{P}}_\rho) \cap \mathcal{Q}_{\mathsf{init}} = \emptyset$.*

Lemma 5. *Let π be a Dyck path from s to t in $\mathcal{G}$, let ρ be the nominal ancestor of π, and let $\lambda(\rho) \in L^*$ be the label of ρ. In addition, let $\mu : L^* \mapsto \mathbb{Z}$ be the morphism of monoids defined by $\mu(\ell_1) = \mu(\ell_2) = 1$ and $\mu(\overline{\ell}_1) = \mu(\overline{\ell}_2) = -1$. Then, we have $\mu(\lambda(\rho)) = 0$.*

A consequence of Lemmas 4 and 5 is the *correctness* of the construction, which is therefore valid.

Proof (Proposition 5). First, if there exists a Dyck path from s to t in G, then Lemma 2 already states that there also exists a Dyck path from s to t in $\mathcal{G}$. Hence, we look at the converse implication.

Let π be a Dyck path from s to t in $\mathcal{G}$, and let ρ be the nominal ancestor of π. Let $\lambda(\rho)$ be the label of ρ and let Λ be the reduction of $\lambda(\rho)$. Lemma 4 proves that $\lambda(\rho)$ is a prefix of a Dyck word, hence that Λ has only letters ℓ_1 and ℓ_2. Since Lemma 5 also proves that $\mu(\lambda(\rho)) = \mu(\Lambda) = 0$, it follows that Λ is the empty word, i.e. that $\lambda(\rho)$ is a Dyck word. □

We complete the proof of Theorem 1 as follows. Observe that the graph $\mathcal{G}$ is FO-definable as a function of G. Furthermore, adding/deleting an edge in E amounts to adding/deleting exactly twelve edges in $\mathcal{E}$. Due to Proposition 5, the 2-letter Dyck reachability problem is therefore bfo-reducible to the 2-letter undirected Dyck reachability problem.

On the other hand, as a restriction of the n-letter Dyck reachability problem, the n-letter undirected Dyck reachability problem is clearly in P, which completes the proof of Theorem 1.

References

1. Mix Barrington, D.A., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comput. Syst. Sci. **41**(3), 274–306 (1990)
2. Bouyer, P., Jugé, V.: Dynamic complexity of the Dyck reachability. Research Report arXiv/1610.07499, Computing Research Repository, October 2016
3. Datta, S., Kulkarni, R., Mukherjee, A., Schwentick, T., Zeume, T.: Reachability is in DynFO. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 159–170. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_13

4. Demri, S., Gascon, R.: The effects of bounding syntactic resources on Presburger LTL. J. Logic Comput. **19**(6), 1541–1575 (2009)
5. Dong, G., Jianwen, S.: Incremental and decremental evaluation of transitive closure by first-order queries. Inf. Comput. **120**(1), 101–106 (1995)
6. Gelade, W., Marquardt, M., Schwentick, T.: The dynamic complexity of formal languages. ACM Trans. Comput. Logic (TOCL) **13**(3), 19 (2012)
7. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press, New York (1995)
8. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_25
9. Hesse, W.: The dynamic complexity of transitive closure is in DynTC^0. Theoret. Comput. Sci. **296**(3), 473–485 (2003)
10. Hesse, W.: Dynamic computational complexity. Ph.D. thesis, Department of Computer Science, University of Massachusetts at Amherst, USA, September 2003
11. Hesse, W., Immerman, N.: Complete problems for dynamic complexity classes. In: Proceedings of the 17th Annual Symposium on Logic in Computer Science (LICS 2002), pp. 313–322. IEEE Comp. Soc. Press, July 2002
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Boston (2006)
13. Immerman, N.: Descriptive Complexity. Graduate Texts in Computer Science. Springer, New York (1999)
14. Miltersen, P.B., Subramanian, S., Vitter, J.S., Tamassia, R.: Complexity models for incremental computation. Theoret. Comput. Sci. **130**(1), 203–236 (1994)
15. Muñoz, P., Vortmeier, N., Zeume, T.: Dynamic graph queries. In: Proceedings of the 19th International Conference on Database Theory, ICDT 2016. Leibniz International Proceedings in Informatics, vol. 48, pp. 14:1–14:18. Leibniz-Zentrum für Informatik (2016)
16. Patnaik, S., Immerman, N.: Dyn-FO: a parallel, dynamic complexity class. J. Comput. Syst. Sci. **55**(2), 199–209 (1997)
17. Weber, V., Schwentick, T.: Dynamic complexity theory revisited. Theory Comput. Syst. **40**(4), 355–377 (2007)