Parity games in pushdown systems with parameters

Mathieu Hilaire ✉
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, Gif-sur-Yvette, France

Abstract

We investigate the decidability and complexity of parity games over an extension of pushdown systems. More specifically, we consider parameters pushdown systems, in which parameters can be instantiated to some word over a stack alphabet, and compared against the current stack content. We determine the problem belongs to $(n+1)$-EXP in case the number of parameters n is fixed, and provide a nonelementary lower bound for the general case.

2012 ACM Subject Classification Theory of computation → Automata extensions

Keywords and phrases Parity Games, Computational Complexity, Pushdown systems

Digital Object Identifier 10.4230/LIPIcs...70

Funding *Mathieu Hilaire*: This work was partly done while the author was supported by the Agence Nationale de la Recherche grant no. ANR-17-CE40-0010.

Acknowledgements The author would like to thank Benedikt Bollig, Stefan Göller and Alain Finkel for helpful discussions and feedback.

1 Introduction

Background. Pushdown systems can be used to model the behavior of recursive programs. As a result, a variety of program analysis questions can be reduced to decision problems for games on pushdown systems. They have applications for instance in inter-procedural control-flow analysis of recursive programs [16, 31].

Two player games can be used in particular to represent the interaction of a program with some environment, with the first player representing the program while the second player represents the environment. A winning condition then expresses a property required to hold however the environment behaves. Finding a winning strategy for the first player thus provides some way to ensure that the desired property, as expressed by the winning condition, always holds [2].

Games on the graphs of pushdown systems have been extensively studied. It is known in particular from [35] that deciding the winner of a parity game on a graph of a pushdown system is an EXPTIME-complete problem. Several generalizations of the pushdown systems have been studied in depths, including tree-pushdown systems [21], ordered multi-pushdown systems [5, 3], annotated higher-order pushdown systems [28, 6], and strongly normed multi-pushdown systems [13].

Parametric pushdown systems Parametric pushdown systems provide a formalism to reason about recursive programs making use of parametric constraints. Recently, different variants of parametric pushdown systems have been introduced in the literature [22, 15, 18] however as parameterized asynchronous shared-memory systems. These systems consist of a leader pushdown automaton and arbitrarily many identical contributor pushdown systems, communicating via a shared memory in the form of a register which can take finitely many values. This variant have been shown to have applications to the dataflow analysis of concurrent programs [24]. We consider here a different approach to extending pushdown systems with parameters, by allowing them to test equality of the stack content against parameters. Perhaps a similar model consists in pushdown systems with transitions that are conditioned by regular conditions on their stack content, which can in particular test the

stack content against specific words. They can be used to ensure that some word over the stack alphabet appears in the configurations of a run, but only for a specific value (or, rather, specific regular languages). They have phrases trop longues been used to establish that CTL* model checking remains decidable when the formulas are allowed to include regular predicates on the stack content, and to obtain model checking algorithms for LTL and CTL* model checking for pushdown systems [17]. Pushdown systems with transitions that are conditioned by regular conditions on their stack content can be viewed as a special case of stack automata as seen in [23]. Instead of checking the stack content against regular conditions, or, in a more limited manner, against specified word values, we consider checks against unspecified word values that can be instantiated using parameters. This allows for example to ensure equality between two stack contents appearing in two distincts configurations in a run.

Related models. Extensions of pushdown systems with storage for later comparisons have been introduced for instance as register pushdown systems. Such automata possess registers in addition to their stack, and can keep data values in both. Register pushdown systems have been shown to have applications for malware detection and XML schema checking [32, 33]. In theory, since registers can be unfilled and later be filled again an unbounded number of times, the number of different data values a pushdown register systems can store in its registers in a run is unbounded. In [30] it was shown however that a register pushdown automaton can only really "remember" at most $3r$ data values, where r is the number of registers, i.e. for any run of a register pushdown automaton with r registers there exists an equivalent run with the same initial and final configurations, but in which every configuration contains register assignments drawn from only $3r$ elements. This leads to the question of how useful the ability to unfill and later refill registers really is compared to a model that would only specify valuations once.

Another closely related formalism is that of alternating tree automata with nested pebbles. The approach of extending tree automata with pebbles was used in [29] to show that the XML typechecking problem is decidable, and in [25] to recognize first-order logic on trees. In both cases however, the input trees considered are finite, while simulation of plays in parametric pushdown graphs requires the use of pebble alternating tree automata working on infinite trees since the stack of a pushdown system is in general unbounded.

The aim of the present paper is to study the decidability and complexity of deciding the winner of a parity game on a graph of a pushdown system with parameters.

Our contributions

Our main result consists in proving parametric pushdown reachability games and parametric pushdown parity games belong to $(n+1)$-EXP in case the number of parameters n is fixed, and providing a nonelementary lower bound for parametric pushdown reachability games in general.

For the nonelementary lower bound, we reduce the FO satisfiability problem on words, known to be nonelementary from [34], to the problem of deciding whether a player has a winning strategy for parametric pushdown reachability games.

For the upper bound, we start by replacing parameters by pebbles acting as registers, leading to a more general model, pebble pushdown systems. We then show that higher-order pushdown systems parity games can be used to solve parity games on the transition systems of pebble pushdown systems, using one additional stack level for each pebble. Since solving parity games on higher-order pushdown systems with level n stack is n-EXP-complete [8, 9], this provides an $(n+1)$-EXP upper bound for solving parametric parity games on parametric pushdown systems with n parameters.

Overview. In Section 2, we provide general notations and preliminary definitions. Section 3 will deal with the introduction of parametric pushdown games. Section 4 is Reduction to higher-order pushdown systems Section 5 at last shows a nonelementary lower bound for the problem.

2 Preliminaries

By $\mathbb{Z}$ we denote the *integers* and by $\mathbb{N} = \{0, 1, \ldots\}$ we denote the *naturals*. For every finite alphabet A, A^* is the set of finite words with letters in A, A^ω is the set of infinite words with letters in A, and $A^\infty = A^\omega \cup A^*$. We denote the empty word in A^* by ϵ.

For any two sets X and S, let X^S denote the set of all functions from S to X. For any set S let $\mathcal{P}(S) = \{X \mid X \subseteq S\}$ denote the power set of S. A *partial function* f from a set X to a set Y is a function defined on a subset C of X (possibly X itself) with output values in Y. We extend the domain of a partial function to the whole set by considering it as returning the bottom element $\perp$ when it is undefined.

A *graph* $G = (V, E)$ is a finite set of graph vertices V with a set of graph edges $E \subseteq V \times V$. For any set V, and a given infinite sequence $\pi = v_0 v_1 \ldots \in V^\omega$ of elements of V, we define the set $Inf(\pi)$ of elements occurring infinitely often in π as $Inf(\pi) = \{v \in V \mid \forall i, \exists j > i,\ v_j = v\}$.

A *game* is composed of an arena and a winning condition. We will first study arenas and then introduce common winning conditions that will be of interest to us.

An *arena* is a tuple $A = (S_0, S_1, \to)$ which is composed of two disjoint sets of configurations, S_0 and S_1, whose disjoint union $S_0 \cup S_1$ we denote by S, and a relation of transitions $\to \subseteq S \times S$. Note that $(S, \to)$, where $S = S_0 \uplus S_1$, forms a transition system, and, in particular, given a transition system, one needs only to provide a partition of the set of configurations S into two sets S_0 and S_1 to obtain an arena.

The games we are interested in are played by two players, called *player* 0 and *player* 1. We will often write player i to denote a general player for $i \in \{0, 1\}$, and we will call player $1 - i$ its *opponent*. A *play* in an arena $A = (S_0, S_1, \to)$ is a path in $(S, \to)$ that is *maximal* in the following sense: it is either infinite or finite and if it is finite, then the target configuration is a dead end. A *partial play* in an arena $A = (S_0, S_1, \to)$ is a prefix of a play in A. Essentially plays can be seen as this: the two players, player 0 and player 1, move along the labeled transition system, taking turns either infinitely often or until a dead end is reached.

A *strategy* for player $i \in \{0, 1\}$ is a function $\sigma_i : S^* S_i \to S$ such that $\sigma_i(ws)$ is a successor of s. A strategy is called *memoryless* if its output only depends on the final configuration of the sequence $s \in S_i$, i.e. if $\sigma_i(ws) = \sigma_i(w's)$ for all $w, w' \in S^*$. Thus, a memoryless strategy for player i can be written as a function $\sigma_i : S_i \to S$.

A partial play $\pi = s_0 \to s_1 \to \ldots \to s_n$ is *consistent* with a strategy σ_i if sequences of configurations that end in S_i along this play have all successors according to the strategy, i.e. if for all $j \in [0, n-1]$, for all element $v_j \in V_i$ in the sequence π, $v_{j+1} = \sigma_i(v_j)$. Given a configuration s, a strategy σ_0 for player 0 and a strategy σ_1 for player 1, the play starting with s that is consistent with both strategies is unique and is called the *resulting play* of σ_0 and σ_1 starting from s. It is denoted by $\pi(s, \sigma_0, \sigma_1)$.

Once we have provided players with an arena, it remains to define properly what the winning condition is in order to define a game. Given an arena $A = (S_0, S_1, \to)$, a *winning condition* Win is a subset of the set of maximal plays for A. A *game* is a pair $\mathcal{G} = (A, \text{Win})$ which is composed of an arena A and a winning condition Win. Although a winning condition is in general an infinite object we are interested in winning conditions that can finitely be described. A strategy σ_0 for player 0 from position $s \in S$ is a *winning strategy from* s for

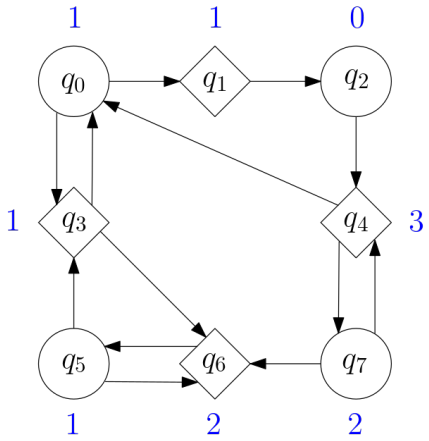

Figure 1 An example of an arena. Circles belong to player 0, whereas diamonds belong to player 1. Arrows represent transitions from a configuration to the next. The priority assigned by the priority mapping to each configuration is a number in $\{0, 1, 2, 3\}$, next to the configuration it is assigned to. Note that we do not have any dead end in our example.

player 0 if every maximal play starting from s and consistent with σ_0 is in Win. On the other hand, a strategy σ_1 for player 1 from position $s \in S$ is a *winning strategy from* s for player 1 if every maximal play starting from s and consistent with σ_1 is not in Win.

We first consider reachability games: the winning condition $\text{Win}_F(S)$ is given by a set of final configurations $F \subseteq S$, and $\text{Win}_F(S)$ is the set of maximal plays in S^*FS^∞. We simply write Win_F in case the set S is obvious from context. To indicate that the winning condition of a game is a reachability winning condition, we will speak of *reachability games*.

With regards to reachability games, we are going to concern ourselves with games over potentially infinitely large arenas, and we are interested in the following decision problem.

Reachability game

INPUT: A reachability game $\mathcal{G} = ((S_0, S_1, \to), \text{Win}_F)$, an initial configuration $s \in S_0 \cup S_1$.

QUESTION: Does player 0 have a winning strategy from s in $\mathcal{G}$?

Secondly, we consider min-parity games: the winning condition $\text{Win}_\Omega(S)$ is given by a *priority function* $\Omega : S \to [0, m]$. An infinite path $\pi = s_0 s_1 s_2 \dots$ is in $\text{Win}_\Omega(S)$ if the smallest priority appearing infinitely often in the sequence is even, i.e. if $\min(\{\Omega(s) |\ s \in \mathit{Inf}(\pi)\})$ is even. A finite play is in $\text{Win}_\Omega(S)$ if its target configuration is in S_1. We simply write Win_Ω in case the set $S = S_0 \uplus S_1$ is obvious from context. To indicate that the winning condition of a game is a parity winning condition, we will speak of *parity games*.

With regards to parity games, we are going to concern ourselves with games over potentially infinitely large arenas, and we are interested in the following decision problem.

Parity game

INPUT: A parity game $\mathcal{G} = ((S_0, S_1, \to), \text{Win}_\Omega)$, an initial configuration $s \in S_0 \cup S_1$.

QUESTION: Does player 0 have a winning strategy from s in $\mathcal{G}$?

▶ **Example 1.** Let us consider the arena presented in Figure 1. Configurations are partitioned into two sets: circles belonging to player 0 and diamonds belonging to player 1. The priorities

are $\{0,1,2,3\}$. Plays $\pi_1 = q_0q_1q_2(q_4q_7)^\omega$ and $\pi_2 = (q_0q_1q_2q_4)^\omega$ are both winning for player 0. Indeed, in π_1, the minimal priority encountered infinitely often is 2, whereas in π_2, it is 0. In both cases, it is an even number. On the other hand, the plays $\pi_3 = (q_5q_3q_6)^\omega$ and $\pi_4 = (q_5q_6)^\omega$ are both winning for player 1: the minimal priority encountered is odd, since it is 1 for both. One can take matters further: on this arena, player 0 has a winning strategy for the parity game from q_0, q_1, q_2, q_4 and q_7, but not from q_5, q_3 nor q_6, where player 1 has a winning strategy.

One important property of parity games is that of *memoryless determinacy*: for a parity game $\mathcal{G} = ((S_0, S_1, \to), \text{WIN}_\Omega)$ and an initial configuration $s \in S_0 \cup S_1$, one of the players has a memoryless winning strategy from s [37].

3 Parametric pushdown games

Parametric pushdown systems extend pushdown systems by allowing the stack to be compared against parameters that can be assigned values which are words over the stack alphabet. A parametric pushdown automaton is then a finite automaton extended with a finite set of parameters P and with a stack that can be manipulated by pushing or popping stack symbols and such that, moreover, the automaton can use the top of the stack, or check that the stack content corresponds to a parameter, to decide which transition to take next.

A *parametric pushdown system* (*PPDS* for short) is a tuple $\mathcal{Z} = (Q, \Gamma, P, R, q_{init}, \gamma_{init}, F)$, where Q is a non-empty finite *set of states*; Γ is a non-empty finite *stack alphabet*; P is a finite *set of parameters*, with $\Gamma \cap P = \varnothing$; $R \subseteq Q \times (\Gamma \uplus P) \times Q \times \mathsf{Op}(\Gamma)$ is finite *set of rules*; $q_{init} \in Q$ is an *initial state*; $\gamma_{init} \in \Gamma$ is an *initial stack symbol*; and $F \subseteq Q$ is a *set of final states*.

The *size* of $\mathcal{Z}$ is defined as $|\mathcal{Z}| = |Q| + |\Gamma| + |P| + |R|$. We also refer to $\mathcal{Z}$ as an n-parametric pushdown automaton if $|P| = n$. A *stack content* is a word from Γ^*. As before we write the top of the stack at the right of the word. By $\mathsf{Conf}(\mathcal{Z}) = Q \times \Gamma^*$ we denote the set of *configurations* of $\mathcal{Z}$. As usual we rather write $q(w)$ instead of (q, w). A *parameter valuation* is a function μ from P to Γ^*.

A parametric pushdown automaton $\mathcal{Z} = (Q, \Gamma, P, uR, q_{init}, \gamma_{init}, F)$ and a parameter valuation $\mu : P \to \Gamma^*$ induce the transition system $T^\mu_{\mathcal{Z}} = (\mathsf{Conf}(\mathcal{Z}), \to_{\mathcal{Z},\mu})$ where for all $q, q' \in Q$, for all $w, w' \in \Gamma^*$, and for all $a \in \Gamma$, $q(wa) \to_{\mathcal{Z},\mu} q'(w')$ if there exists a rule in R of the form (q, a, q', op) or (q, p, q', op) with $\mu(p) = wa$, such that either of the following holds

- $op = push^\gamma$ and $w' = wa\gamma$,
- $op = pop$ and $w' = w$, or
- $op = skip$ and $w' = wa$.

A *μ-run* from $q_0(a_0)$ to $q_n(a_n)$ in $\mathcal{Z}$ is a corresponding path in the transition system $T^\mu_{\mathcal{Z}}$ induced by $\mathcal{Z}$ and μ. As with pushdown systems, we say π is *accepting* if $q_0 = q_{init}$, $a_0 = \gamma_{init}$, and $q_n \in F$.

In the particular case where $P = \{p\}$ is a singleton for some parameter p and $\mu(p) = u \in \Gamma^*$, we prefer to write $q(w) \to_{\mathcal{Z},u} q'(w')$ to denote $q(w) \to_{\mathcal{Z},\mu} q'(w')$ and will call the μ-run an *u-run*. In case the automaton $\mathcal{Z}$ is obvious from context, we write $\to_\mu$ (resp. $\to_u$) instead of $\to_{\mathcal{Z},\mu}$ (resp. $\to_{\mathcal{Z},u}$). Since we are concerned with the graph of configurations and not the language recognized, we omit the input alphabet from the model and will use the name pushdown system rather than pushdown automaton.

3.1 PPDS games

Several model checking problems can be expressed as decision problems for games. For instance in the case of pushdown systems, the modal μ-calculus model checking problem is polynomially equivalent to solving the parity game problem [**?**]. We are interested in this article in parametric variants of reachability games and parity games.

One goal may be to characterize the set of all possible parameter valuations for which a player has a winning strategy in a reachability game or a parity game played on the arena induced by the resulting concrete automaton and some partition of its set of states. For determining such a set of parameter valuations, we incorporate the choice of the parameter valuation to the game. Each player then wants the parameter valuation to lead to an arena where it has a winning strategy.

In keeping with the spirit of alternation between players choosing successors in the arena, we consider here games where players alternate choosing values for the parameters before alternating choosing successor configurations in the game on the induced arena. For simplicity's sake, the values are assigned to the parameters one by one, alternating between values assigned by player 0 and by player 1.

A winning strategy for player 0 allows to synthesize a controller that restricts the environment and ensures that the property expressed by the winning condition always holds.

We introduce here arenas over the set of partial parameter valuations for a set of parameters $P = P_0 \uplus P_1$. The values are assigned to the parameters one by one, alternating between values assigned by player 0 (corresponding to parameters in P_0) and by player 1 (corresponding to parameters in P_1). Other options could have been considered: for instance, we could consider a fully existential approach (i.e. all parameters have their values assigned by player 0), or a fully universal approach (i.e. all parameters have their values assigned by player 1). Note however that the approach considered here encompasses both of the latter approaches: the fully existential approach consists in the case where $P_1 = \varnothing$ and the fully universal one, the case $P_0 = \varnothing$.

Assume an ordering of the parameters $P_0 \uplus P_1 = \{p_0, p_1, \dots, p_k\}$ for some $k \in \mathbb{N}$. Then given a alphabet Γ, a *parameter valuation arena* is a tuple $A_{P_0,P_1,\Gamma} = (S_0, S_1, \rightarrow_P)$ where

- S_0 is the set of all partial parameter valuations with images in Γ^* and domain $\{p_0, p_1, \dots, p_j\} \setminus \{p_j\}$ where $\mathsf{p}_j \in P_0$,
- S_1 is the set of all partial M-parameter valuations with images in Γ^* and domain $\{p_0, p_1, \dots, p_j\} \setminus \{p_j\}$ where $\mathsf{p}_j \in P_1$,
- $\mu \rightarrow_P \mu'$, if $\mu' \in (\Gamma^*)^{\{p_0,\dots,p_j\}}$ for some $j \in [0,k]$, $\mathrm{Dom}(\mu) = \mathrm{Dom}(\mu') \setminus \{p_j\}$ and $\mu'(p) = \mu(p)$ for $p \in \mathrm{Dom}(\mu)$.

Note that the arena has dead ends, and that these are the configurations that correspond to parameter valuations. Given a strategy σ_0 for player 0 and a strategy σ_1 for player 1, the last configuration of the resulting play $\pi(\mu_\perp, \sigma_0, \sigma_1)$ is called *the resulting parameter valuation* and is denoted by μ_{σ_0,σ_1}, where $\mu_\perp$ is the partial parameter valuation with domain $\varnothing$. We then use parametric valuation arenas as prefixes in parametric games: once the resulting parameter valuation is produced by the two players, it serves to instantiate the transition system for continuing the game. Indeed, where a concrete automaton induces a transition system, a parametric automaton induces one transition system for every possible parameter valuation. We use $T^\mu_{\mathcal{A}}$ to denote the transition system induced by the concrete automaton corresponding to a parametric automaton $\mathcal{A}$ with parameters P taking values in Γ^* and a parameter valuation $\mu : P \to \Gamma^*$.

More formally, for an n-parametric pushdown automaton $\mathcal{Z} = (Q, \Gamma, P, R, q_{init}, \gamma_{init}, F)$, given a partition of Q into Q_0 and Q_1, we naturally partition the configurations of $\mathcal{Z}$ into

$\mathsf{Conf}_{\mathcal{Z},0} = Q_0 \times \Gamma^*$ and $\mathsf{Conf}_{\mathcal{Z},1} = Q_1 \times \Gamma^*$. With these notations in mind we define the arena $A_{(\mathcal{Z},Q_0,Q_1,\mu)} = (\mathsf{Conf}_{\mathcal{Z},0}, \mathsf{Conf}_{\mathcal{Z},1}, \rightarrow_{\mathcal{Z},\mu})$ induced by $\mathcal{Z}$, the partition $Q = Q_0 \uplus Q_1$, and a parameter valuation $\mu : P \to \Gamma^*$. From a partition of P into P_0 and P_1, we thus define $A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1)}$ as the arena that contains both the configurations and transitions of the parameter valuation arena $A_{P_0,P_1,\Gamma}$, and that moreover contains the configurations and transitions of $A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1,\mu)}$ — albeit with configurations additionally including μ in the tuple to avoid confusion — for all parameter valuations $\mu : P \to \Gamma^*$. Moreover, for every configuration μ in A_{P_0,P_1,Γ^*}, we add a transition from μ towards $q_{init}(\gamma_{init}, \mu)$ in $A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1,\mu)}$.

We extend any priority function $\Omega : Q \to [0,m]$ into $\widehat{\Omega} : \mathsf{Conf}(\mathcal{Z}) \cup (\Gamma^* \cup \bot)^{[0,n]} \to [0,m]$, where we set $\widehat{\Omega}(q,w,\mu) = \Omega(q)$ for all $w \in \Gamma^*$ and for all $\mu \in (\Gamma^*)^P$, and $\widehat{\Omega}(\mu) = 0$ for all $\mu \in (\Gamma^* \cup \bot)^P$.

Parametric pushdown reachability game

INPUT: A parametric pushdown automaton $\mathcal{Z} = (Q, \Gamma, \{p_0, p_1, \ldots, p_k\}, R, q_{init}, \gamma_{init}, F)$, where $Q = Q_0 \uplus Q_1$, and $P = P_0 \uplus P_1$.

QUESTION: Does player 0 have a winning strategy from $\mu_\bot$ in the reachability game $\mathcal{G} = (A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1)}, \mathrm{WIN}_{F \times \Gamma^* \times (\Gamma^*)^P})$?

Parametric pushdown parity game

INPUT: A parametric pushdown automaton $\mathcal{Z} = (Q, \Gamma, \{p_0, p_1, \ldots, p_k\}, R, q_{init}, \gamma_{init}, F)$, where $Q = Q_0 \uplus Q_1$, and $P = P_0 \uplus P_1$, and a priority function $\Omega : Q \to [0,m]$.

QUESTION: Does player 0 have a winning strategy from $\mu_\bot$ in the parity game $\mathcal{G} = (A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1)}, \mathrm{WIN}_{\widehat{\Omega}})$?

We write n-parametric reachability parity game and n-parametric pushdown parity game if the number of parameters $n = |P|$ of $\mathcal{Z}$ is fixed by the problem. Remark that the parametric reachability problem for an automaton with parameters can be seen as solving the parametric reachability game where $P_1 = \varnothing$, and $Q_1 = \varnothing$.

4 Reduction to pebble pushdown systems parity games

A pebble automaton is a two-way finite state automaton that uses a fixed, finite number of pebbles that it can drop on, and lift from words, using them as markers. Pebble automata recognize regular languages only, provided the life times of the pebbles, i.e. the times between dropping a pebble and lifting it again, are properly nested [19, 14]. Automata with nested pebbles were also introduced for tree-walking automata. It is known that tree-walking automata do not recognize all regular tree languages [4]. Using pebbles is a remedy against getting lost along a tree, but the unrestricted use of pebbles leads to a class of tree languages much larger than the regular tree languages, in fact to all tree languages in $\mathsf{NSPACE}(\log n)$. Thus, in both pebble word automata and pebble tree-walking automata, the placement of the pebble follows a strict stack discipline. It is traditional hence to represent syntactically the dropping and lifting of pebbles by operations *lift* and *drop*; a *drop* simply records the current position with a fresh pebble (such a pebble should be available) and a *lift* pops the last dropped pebble if the current position corresponds to the one recorded by it.

In this section, we extend these ideas to pushdown systems. Instead of using pebbles as markings on their input, pebble pushdown systems have the ability to lift or drop pebbles on their universe of stack contents; a *drop* simply records the current stack content with a fresh pebble (such a pebble should be available) and a *lift* pops the last dropped pebble while

requiring that it was placed on the current node. One can think of a pebble as a register that can store a stack content for later comparisons.

We first define more formally our pebble pushdown automaton framework, and then provide a reduction from the problem of solving parametric pushdown parity games to the problem of solving pebble pushdown parity games.

4.1 Pebble pushdown systems

An *n-pebble pushdown automaton* is a tuple $\mathcal{I} = (Q, \Gamma, R, q_{init}, \gamma_{init}, F)$ where Q is a *finite set of states*; Γ is a *finite stack alphabet*; $R \subseteq Q \times \Gamma \times \{0, 1, \dots n\} \times \mathcal{P}(\{1, \dots, n\}) \times Q \times (\mathsf{Op}(\Gamma) \cup \{drop, lift\})$ is a *finite set of rules*, where the fourth element S of a rule r is a subset of $\mathcal{P}(\{1, \dots, i\})$ where i is the third element of r, and if the last element is *lift* then we additionally require $i \in S$; $q_{init} \in Q$ is an *initial state*; $\gamma_{init} \in \Gamma$ is an *initial stack symbol*; and $F \subseteq Q$ is a *set of final states*.

Recall that for a partial function $f : S \rightharpoonup X$, for notational purposes, we consider some element $\bot_X \notin X$ and associate f with the function returning the bottom element $\bot_X$ when f is undefined. Thus we write $(X \uplus \{\bot_X\})^S$ for the set of all partial functions from S to X, which we here abbreviate as $(X \uplus \{\bot\})^S$.

By $\mathsf{Conf}(\mathcal{I}) = Q \times \Gamma^* \times (\Gamma^* \uplus \{\bot\})^{\{1,\dots,n\}}$ we denote the set of *configurations* of $\mathcal{I}$. As expected, we rather write $q(w, \mu)$ instead of (q, w, μ). An *i-configuration* for $i > 0$ of $\mathcal{I}$ is a configuration $q(z, \mu)$ where $\mathrm{Dom}(\mu) = \{1, \dots, i\}$, while a 0-configuration is a configuration $q(z, \mu)$ where $\mathrm{Dom}(\mu) = \varnothing$.

The idea of a transition $(q, a, i, S, q', m) \in R$ is that, if the automaton $\mathcal{I}$ is in state q with pebbles $1, \dots, i$ dropped — or without pebble dropped if $i = 0$ — with top stack symbol a, and stack content which corresponds to the stack contents of the pebbles from S and only these pebbles, then $\mathcal{I}$ goes to state q' and makes modifications to the stack or the pebbles according to m. Note a pebble can be lifted only if the stack content which corresponds to the pebble is the same as the current stack content. This is enforced by syntactically requiring the last pebble dropped i is in the set S used for testing the presence of certain pebbles.

A *pebble set* of $\mathcal{I}$ is a set $U \subseteq \{1, \dots, n\}$. For a stack alphabet Γ, a *U-pebble assignment* is a function which maps each $j \in U$ to a word in Γ^*. The $\varnothing$-pebble assignment is denoted by $\mu_{init} : \{1, \dots, n\} \rightharpoonup \Gamma^*$ and is the totally undefined function.

An n-pebble pushdown automaton $\mathcal{I} = (Q, \Gamma, R, q_{init}, \gamma_{init}, F)$ induces a transition system $T_{\mathcal{I}} = (\mathsf{Conf}(\mathcal{I}), \to_{\mathcal{I}})$ such that for all $(q, a, i, S, q', m) \in R$, with $a \in \Gamma$, for all words $w \in \Gamma^*$, and for all $\{1, \dots, i\}$-pebble assignments μ such that $\mu(j) = wa$ for each $j \in S$ and $\mu(j) \neq wa$ for each $j \in \{1, \dots, i\} \setminus S$, the following holds

- if $m \in \mathsf{Op}(\Gamma)$, either
  - $m = push^\gamma$ and $(q, wa, \mu) \to_{\mathcal{I}} (q', wa\gamma, \mu)$,
  - $m = pop$ and $(q, wa, \mu) \to_{\mathcal{I}} (q', w, \mu)$, or
  - $m = skip$ and $(q, wa, \mu) \to_{\mathcal{I}} (q', wa, \mu)$,
- if $m = drop$, $(q, wa, \mu) \to_{\mathcal{I}} (q', wa, \mu')$, where μ' is the $\{1, \dots, i, i+1\}$-pebble assignment such that $\mu'(j) = \mu(j)$, for each $j \leq i$, and $\mu'(i+1) = wa$, and
- if $m = lift$, and $i \in S$, i.e. the last pebble dropped belong of the set of pebble we test the presence of, $(q, wa, \mu) \to_{\mathcal{I}} (q', wa, \mu')$, where μ' is the $\{1, \dots, i-1\}$-pebble assignment such that $\mu'(j) = \mu(j)$, for each $j < i$.

We are interested in games over pebble pushdown systems, mainly, parity games.

Again, given a transition system, one needs only to provide a partition of the set of configurations to obtain an arena. Given a partition of Q into Q_0 and Q_1, we partition the configurations of $T_{\mathcal{I}}$ into $\mathsf{Conf}_{\mathcal{I},0} = Q_0 \times \Gamma^* \times (\Gamma^* \uplus \{\bot\})^{\{1,\dots n\}}$ and $\mathsf{Conf}_{\mathcal{I},1} = Q_1 \times \Gamma^* \times (\Gamma^* \uplus \{\bot\})^{\{1,\dots n\}}$. We define the arena $A_{(\mathcal{I},Q_0,Q_1)} = (\mathsf{Conf}_{\mathcal{I},0}, \mathsf{Conf}_{\mathcal{I},1}, \rightarrow_{\mathcal{I}})$. As expected, given a priority function $\Omega : Q \to \{0,\dots m\}$, one naturally set the extension of Ω as $\overline{\Omega} : \mathsf{Conf}(\mathcal{I}) \to \{0,\dots m\}$ and $\overline{\Omega}(q,w,\mu) = \Omega(q)$ for all $w \in \Gamma^*$ and $\mu \in (\Gamma^* \uplus \{\bot\})^{\{1,\dots,n\}}$.

Concerning pebble pushdown systems, we are interested in the following decision problem.

n-PEBBLE PUSHDOWN PARITY GAME

INPUT: An n-pebble pushdown automaton $\mathcal{I} = (Q, \Gamma, R, q_{init}, \gamma_{init}, F)$, where $Q = Q_0 \uplus Q_1$, and a priority mapping $\Omega : Q \to \{0,\dots m\}$.

QUESTION: Does player 0 have a winning strategy from $q_{init}(\gamma_{init}, \mu_{init})$ for the parity game $\mathcal{G} = (A_{(\mathcal{I},Q_0,Q_1)}, \text{Win}_{\overline{\Omega}})$?

4.2 From parametric pushdown systems to pebble pushdown systems

We now provide a reduction from the problem of solving parametric pushdown parity games to the problem of solving pebble pushdown parity games.

▶ **Theorem 2.** n-PARAMETRIC PUSHDOWN PARITY GAME *is polynomial time reducible to* n-PEBBLE PUSHDOWN PARITY GAME*.*

Sketch. Let us fix some n-parametric pushdown automaton $\mathcal{Z} = (Q, \Gamma, P, q_{init}, \gamma_{init}, F)$, disjoint union $Q = Q_0 \uplus Q_1$ and $P = P_0 \uplus P_1$, and some priority function $\Omega : Q \times \Gamma^* \times (\Gamma^* \uplus \{\bot\})^R \to [0,m]$.

We construct an n-pebble pushdown automaton $\mathcal{I} = (Q', \Gamma, R', q'_{init}, \gamma_{init}, F')$, a disjoint union $Q' = Q'_0 \uplus Q'_1$ and a mapping $\Omega' : Q \times \Gamma'^* \times (\Gamma'^* \uplus \{\bot\})^{\{1,\dots,n\}} \to \{1,\dots,m\}$, such that player 0 has a winning strategy from $(q'_{init}, \gamma_{init}, \mu_{init})$ in the parity game $\mathcal{G}' = (A_{(\mathcal{I},Q'_0,Q'_1)}, \text{Win}_{\overline{\Omega'}})$ if and only if player 0 has a winning strategy from $\mu_\bot$ in the parity game $\mathcal{G} = (A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1)}, \text{Win}_{\widehat{\Omega}})$, where $A_{(\mathcal{Z},P_0,P_1,Q_0,Q_1)}$ and $\widehat{\Omega}$ correspond to the definitions on page 7.

The transition graph of the n-pebble pushdown automaton $\mathcal{I}$ will simulate the possible transitions graphs (one for every parameter assignment) of the pushdown automaton with n parameters by using pebbles to represent the parameters. The n-pebble pushdown automaton will first simulate the parameter valuation arena. Players take turns chosing particular stack contents on which to place pebbles. Once every parameter has been assigned with the dropping of a corresponding pebble, the pebble assignment μ is fixed.

Further configurations in the n-pebble pushdown automaton then consist of a word representing the status of the stack, a state, and a fixed pebble assignment. For a fixed pebble assignment there is then a one for one correspondance between configurations of the n-parametric pushdown automaton and these of the n-pebble pushdown automaton. The set of states of the n-pebble pushdown automaton apart from the initial gadget is the same as the set of states of the n-parametric pushdown automaton, and so are player 0 states, player 1 states and the priority mapping. ◀

5 Reduction to higher-order pushdown systems

Higher-order pushdown systems (HPDS for short) were introduced as a generalization of pushdown systems [1, 20, 28]. A stack of a pushdown automaton is seen as a *level* 1 *stack*.

A pushdown automaton of level 2 (or 2-HPDS) then works with a stack of level 1 stacks. In addition to the ability to push and to pop a symbol on the top-most level 1 stack, an 2-HPDS can copy or remove the entire topmost level 1 stack. The definition generalizes to any $n \geq 2$, and n-HPDS are similarly defined for all level n as systems working with a stack of level $(n-1)$ stacks.

We recall the definition from [9] which itself is taken from [26]. We then provide a reduction from the problem of solving pebble pushdown parity games to the problem of solving higher-order pushdown parity games.

5.1 Higher-order pushdown systems

A *level* 1 *stack* (or 1-*stack*) over an alphabet Γ is simply a stack over Γ, i.e. a word in Γ^*. A *level* n *stack* (or n-*stack*) over an alphabet Γ, for $n \geq 2$, is a non-empty sequence $\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle$ of $(n-1)$-stacks over Γ, for some $m \in \mathbb{N}$. The set of n-stacks over Γ is denoted by $\mathcal{S}_n(\Gamma)$, or simply $\mathcal{S}_n$ in case the set Γ is obvious from context. The set of all stacks over Γ is written $\mathcal{S}(\Gamma) = \bigcup_{n \in \mathbb{N}} \mathcal{S}_n(\Gamma)$. We define ϵ_1 as $\epsilon \in \Gamma^*$ and we inductively define $\epsilon_n = \langle \epsilon_{n-1} \rangle$ in $\mathcal{S}_n$ for all $n > 1$.

A *higher-order stack operation* is a partial function from $\mathcal{S}(\Gamma)$ to $\mathcal{S}(\Gamma)$ which preserves the level of the input (i.e. the image of an n-stack is an n-stack for all $n \in \mathbb{N}$). The *level* of an operation *op* is the smallest $n \in \mathbb{N}$ such that $\mathrm{Dom}(op) \cap \mathcal{S}_n \neq \emptyset$. The operations additionally respect the hierarchicality of higher-order stacks, i.e. in a level $n+1$ stack only the topmost level n stack can be accessed. An operation *op* of level n, applied to a level $n+1$ stack $\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle$ of length $m \in \mathbb{N}$, thus returns the output $\langle s_0 \rangle \langle s_1 \rangle \dots \langle op(s_m) \rangle$ if applicable. The definition for all levels of stacks greater than n follows the same pattern.

The following operations can be performed on a 1-stacks of length $m \in \mathbb{N}$.

$$\begin{aligned} push_1^\gamma(w_0 w_1 \dots w_m) &= w_0 w_1 \dots w_m \gamma \text{ for all } \gamma \in \Gamma, \\ pop_1(w_0 w_1 \dots w_{m-1} w_m) &= w_0 w_1 \dots w_{m-1}, \text{ if } m \geq 1 \\ top(w_0 w_1 \dots w_m) &= w_m. \end{aligned}$$

The operations added at level $n+1$ are the copy of the topmost n-stack and the removal of the topmost n-stack. More formally, if $\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle$ is a stack of level $n > 1$, the following operations are possible.

$$\begin{aligned} push_n(\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle) &= \langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle \langle s_m \rangle, \\ push_j(\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle) &= \langle s_0 \rangle, \langle s_1 \rangle \dots \langle push_j(s_m) \rangle, \text{ if } 2 \leq j < n \\ push_1^\gamma(\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle) &= \langle s_0 \rangle \langle s_1 \rangle \dots \langle push_1^\gamma(s_m) \rangle, \text{ for all } \gamma \in \Gamma, \\ pop_n(\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_{m-1} \rangle \langle s_m \rangle) &= \langle s_0 \rangle \langle s_1 \rangle \dots \langle s_{m-1} \rangle, \\ pop_j(\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_{m-1} \rangle \langle s_m \rangle) &= \langle s_0 \rangle \langle s_1 \rangle \dots \langle s_{m-1} \rangle \langle pop_j(s_m) \rangle, \text{ if } 1 \leq j < n \\ top(\langle s_0 \rangle \langle s_1 \rangle \dots \langle s_m \rangle) &= top(s_m). \end{aligned}$$

The operations pop_1 and top are undefined on a stack whose top 1-stack is empty.

Given a stack alphabet Γ and $n \in \mathbb{N}$, we denote by $\mathsf{Op}_n(\Gamma)$ the *base set of* n-*stack operations* as

- $push_j$ for all $2 \leq j \leq n$,
- $push_1^\gamma$ for all $\gamma \in \Gamma$,
- pop_j for all $1 \leq j \leq n$,

- *skip*, corresponding to the identity function of $\mathcal{S}(\Gamma)$.

A *higher-order pushdown systems of level n* (or n-HPDS for short) is a tuple $\mathcal{H} = (Q, \Gamma, R, q_{init}, \gamma_{init}, F)$, where

- Q is a non-empty finite *set of states*,
- Γ is a non-empty finite *stack alphabet*,
- $R \subseteq Q \times \Gamma \times Q \times \mathsf{Op}_n(\Gamma)$ is a finite *set of rules*,
- q_{init} is the *initial state*,
- $\gamma_{init} \in \Gamma$ is the *initial stack symbol*, and
- $F \subseteq Q$ is a *set of final states*.

By $\mathsf{Conf}(\mathcal{H}) = Q \times \mathcal{S}_n(\Gamma)$ we denote the set of *configurations* of an n-HPDS $\mathcal{H}$. As usual we abbreviate $(q, s) \in \mathsf{Conf}(\mathcal{H})$ as $q(s)$.

An n-HPDS $\mathcal{H} = (Q, \Gamma, R, q_{init}, F)$ induces the transition system $\mathcal{T}_{\mathcal{H}} = (Conf(\mathcal{H}), \rightarrow_{\mathcal{H}})$ where for all q, q' in Q, for all s, s' in $\mathcal{S}_n(\Gamma)$, $q(s) \rightarrow_{\mathcal{H}} q'(s')$ if there exists some rule $(q, \gamma, q', op) \in R$ such that $top(s) = \gamma$ and $s' = op(s)$.

Again we are interested in parity games. As expected, given an n-HPDS $\mathcal{H}$ and a partition of Q into Q_0 and Q_1, we partition the configurations of $T_{\mathcal{H}}$ into $\mathsf{Conf}_{\mathcal{H},0} = Q_0 \times \mathcal{S}_n(\Gamma)$ and $\mathsf{Conf}_{\mathcal{H},1} = Q_1 \times \mathcal{S}_n(\Gamma)$. With these notations in mind one can define the arena

$$A_{(\mathcal{H},Q_0,Q_1)} = (\mathsf{Conf}_{\mathcal{H},0}, \mathsf{Conf}_{\mathcal{H},1}, \rightarrow_{\mathcal{H}})$$

induced by an n-HPDS $\mathcal{H}$ and a partition of its set of states.

As expected, given a priority function $\Omega : Q \rightarrow [0, m]$, we naturally extend the function as follows, by setting $\Omega_{\mathcal{S}_n(\Gamma)} : \mathsf{Conf}(\mathcal{H}) \rightarrow [0, m]$ and $\Omega_{\mathcal{S}_n(\Gamma)}(q, s) = \Omega(q)$ for all $s \in \mathcal{S}_n(\Gamma)$.

Concerning higher-order pushdown systems, we are interested in the following decision problem.

n-HPDS PARITY GAME

INPUT: An n-HPDS $\mathcal{H} = (Q, \Gamma, R, q_{init}, \gamma_{init}, F)$, where $Q = Q_0 \uplus Q_1$, and a priority mapping $\Omega : Q \rightarrow \{0, \dots m\}$.

QUESTION: Does player 0 have a winning strategy from $q_{init}(push_1^{\gamma_{init}}(\epsilon_n))$ for the parity game $\mathcal{G} = (A_{(\mathcal{H},Q_0,Q_1)}, \text{WIN}_{\Omega_{\mathcal{S}_n(\Gamma)}})$?

It was shown in [8] that n-HPDS PARITY GAME can be solved in n-EXP. This also gives an n-EXP algorithm for the μ-calculus model checking over transitions systems induced by n-HPDS. In [9] the matching lower bound was showed, even in the case of reachability games, hence showing n-EXP-completeness of n-HPDS PARITY GAME.

▶ **Theorem 3.** *[8, 9] n-HPDS PARITY GAME is n-EXP-complete.*

In a 2-HPDS, the operation $push_2$ allows to "copy" the top level 1 stack. The current word is hereby stored away and left untouched until the next operation pop_2, while $push_1$ and pop_1 can be performed on the additional "copy" in the meantime. This behavior is similar to that of dropping and lifting a pebble. The main difference is that there is no operation to test that the "copy", after many updates, is again equal to the "original", i.e. there is no operation to syntactically test that the two topmost level 1 stacks are identical.

In [11, 36, 10] however Carayol and Wöhrle introduced a variant of n-HPDS by extending the pop_j operations for $2 \leq j \leq n$ with a built-in equality test. For $2 \leq j \leq n$ the new operation $pop_j^=$ has the same effect as pop_j, but can only be applied if the two top level j stacks coincide. In [10] it is seen as a symmetrical operation in comparison to $push_j$.

A *higher-order pushdown automaton with equality pop of level n* (n-HPDS$^{=}$ for short) is a higher-order pushdown automaton of level n where in $\mathsf{Op}_n(\Gamma)$ the operation pop_j is replaced by $push_j^{=}$ for $2 \leq j \leq n$. We denote this new set of operations by $\mathsf{Op}_n^{=}(\Gamma)$. More formally the new operations $push_j^{=}$ are defined as

$$pop_k^{=}(\langle s_0\rangle\langle s_1\rangle \dots \langle s_m\rangle\langle s_m\rangle) = \langle s_0\rangle\langle s_1\rangle \dots \langle s_m\rangle, \quad \text{and}$$
$$pop_j^{=}(\langle s_0\rangle\langle s_1\rangle \dots \langle s_m\rangle) = \langle s_0\rangle\langle s_1\rangle \dots \langle pop_j^{=}(s_m)\rangle, \text{ if } 2 \leq j < k.$$

Transition systems induced by higher-order pushdown systems with equality pop of level n are then defined as expected. Carayol and Wöhrle proved [36, 10] that the two models, namely higher-order pushdown systems with equality pop of level n and higher-order pushdown systems of level n, generate the same classes of transition systems.

▶ **Theorem 4.** *[36, 10] If $\mathcal{H}$ is an n-HPDS (resp. n-HPDS$^{=}$) then there exists an n-HPDS$^{=}$ (resp. n-HPDS) $\mathcal{H}'$ such that $\mathcal{H}$ and $\mathcal{H}'$ induce isomorphic transition systems.*

From n-HPDS to n-HPDS$^{=}$ (Proposition 3.12 in [36]) the proof relies on recreating a correct stack content to simulate higher-order *pop* operations by $pop^{=}$, essentially "guessing" the correct stack content to be able to apply $pop^{=}$. In the other direction (Proposition 3.18 in [36]), the author enrich the stack alphabet with new symbols stating which instructions have to be executed to recreate a previous stack content. It is proven that such en encoding is possible since there is for every stack s of level n a unique shortest sequence of instructions which creates s from ϵ_n.

Defined like n-HPDS PARITY GAME, we write n-HPDS$^{=}$ PARITY GAME in case the input is an n-HPDS$^{=}$ rather than an n-HPDS.

The algorithm from [8] actually provides an algorithmic solution to parity games on the graphs of the Caucal hierarchy. We skip a formal definition of a graph of level n in the Caucal hierarchy, and refer the reader to [12] for more details. The n-EXP upper bound on n-HPDS parity games follows from the fact that every transition system induced by an n-HPDS $\mathcal{H}$ is a graph of the Caucal hierarchy [8, 36], whose vertices are almost in one-to-one correspondence with the configurations of $\mathcal{H}$. In [36] the converse direction is proven, i.e. that every graph of level n of the Caucal hierarchy is generated by an n-HPDS. In [10] it is similarly shown that every transition system induced by an n-HPDS$^{=}$ $\mathcal{H}$ is a graph of the Caucal hierarchy. The algorithm from [8] hence lead to a solution for n-HPDS$^{=}$ parity games as well.

▶ **Theorem 5.** n-HPDS$^{=}$ PARITY GAME *is in* n-EXP.

5.2 From pebble pushdown systems to higher-order pushdown systems

We use HPDS$^{=}$ parity games to simulate pebble pushdown systems parity games. A similar approach was used in [10] to show that $(n+2)$-level stack systems could be used to simulate n-pebble alternating two-way word systems.

Let us start by discussing the simulation of the dropping and lifting of a pebble in the case of a pebble pushdown automaton $\mathcal{I}$ with only one pebble. As long as no pebble is dropped, a 2-HPDS$^{=}$ $\mathcal{H}$ can simulate the behavior of a pebble pushdown automaton $\mathcal{I}$ with a 2-stack containing a single 1-stack on which it performs the same operations $\mathcal{I}$ performs on its stack. When the automaton $\mathcal{I}$ is in configuration $q(w)$ and drops a pebble, instead of making a pop or a push, we need to store the information that the pebble has been dropped on w. To simulate the next configuration in such a case, we use $push_2$ to store the

information that the pebble has been dropped on the word w, leading to the new 2-stack $\langle w\rangle\langle w\rangle$. Configurations afterwards have 2-stacks of length two where the first component remains w until the pebble is lifted. Lifting the pebble can be done only at the position the pebble was dropped, by using $pop_2^=$. Checking that the node corresponds to the one where the pebble has been dropped simply makes use of the composition of $pop_2^=$ and $push_2$. Checking the absence of the pebble can be done by challenging the opponent to prove the presence of the pebble.

Now, let us detail the case when there is a second pebble, the construction for each additional pebble being highly similar, where every additional pebble after the first one would require the addition of another stack level. Similarly as before, as long as no pebble is dropped, the stack of $\mathcal{H}$ is a 3-stack containing a single 2-stack that contains a single 1-stack. To simulate configurations in the case one pebble have been dropped, the stack contains a single 2-stack of the form $\langle w_1\rangle\langle w\rangle$. In the case the two pebbles have been dropped, the stack is of the form $\langle\langle w_1\rangle\langle w_2\rangle\rangle\langle\langle w_1\rangle\langle w\rangle\rangle$.

Intuitively speaking, w_1 is, as before, the position where the first pebble is placed, and w_2 is where the second pebble is placed. Dropping pebbles like this again involves the cloning operation, only, each pebble operates at a different stack level: thus, dropping the first pebble will use $push_2$ and dropping the second pebble will use $push_3$. Knowing the number of pebbles dropped can be done by keeping track using the states of $\mathcal{H}$. Lifting or checking for the presence (or absence) of the pebbles functions on a similar basis as before. Checking the presence of the first pebble uses operation $pop_2^=$ followed by $push_2$, while checking the presence of the second pebble uses operation $pop_3^=$ followed by $push_3$. Lifting the first pebble uses $pop_2^=$, but check first that the second pebble has already been lifted. Lifting the second pebble simply uses $pop_3^=$.

By expanding this reasoning inductively, we conclude that given $n \in N$, and given an n-pebble pushdown automaton $\mathcal{I} = (Q, \Gamma, R, q_{init}, \gamma_{init}, F)$, where $Q = Q_0 \uplus Q_1$, one can compute an $(n+1)$-HPDS $\mathcal{H} = (Q', \Gamma, R', q'_{init}, \gamma'_{init}, F')$ where $Q' = Q'_0 \uplus Q'_1$, such that player 0 has a winning strategy from $q'_{init}(push_1^{\gamma'_{init}}(\epsilon_n))$ in $A_{(\mathcal{H},Q'_0,Q'_1)}$ if and only if player 0 has a winning strategy from $q(\gamma_{init}, \mu_{init})$ in $A_{(\mathcal{I},Q_0,Q_1)}$. Hence the following reduction.

▶ **Theorem 6.** *n-pebble pushdown parity game is polynomial time reducible to $(n+1)$-HPDS$^=$ parity game.*

The reduction implies decidability of pebble pushdown systems parity game. Furthermore, by Theorem 5, it implies the following complexity result.

▶ **Theorem 7.** *n-pebble pushdown systems parity game is in $(n+1)$-EXP.*

Finally, the following complexity result is due the above and Theorem 2.

▶ **Corollary 8.** *n-parametric pushdown systems parity game is in $(n+1)$-EXP.*

6 Lower bound

The aim of this section is to show a nonelementary lower bound for the problem of deciding whether player 0 has a winning strategy from some starting configuration in a parametric pushdown parity game.

We reduce the satisfiability problem for first-order logic on words to the problem of Parametric pushdown parity game. We know from [34] that satisfiability is nonelementary.

We define the tower function $T : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ by $T(0, r) = r$ and $T(h+1, r) = 2^{T(h,r)}$ for all $h \in \mathbb{N}, r \in \mathbb{R}$. Thus $T(h, r)$ is a tower of 2s of height h with an r sitting on top. Observe that

for all $n, h \in \mathbb{N}$ with $n \geq 1$, we have $T(h, log^{(h)}(n)) = n$. Here a problem is in ELEMENTARY if it can be solved in time $O(T(n, 0))$.

For a finite alphabet Σ, let $\tau(\Sigma)$ be the vocabulary consisting of a binary relational symbol $\leq$, and a unary relational symbol P_a for every $a \in \Sigma$. A word $w \in \Sigma^*$ can be seen as a $\tau(\Sigma)$-structure where the set of elements consists of a finite interval of $\mathbb{N}$. Elements are seen as positions in the word and for every element i, there exists precisely one $a \in \Sigma$ such that i is in the interpretation of P_a. Lastly $\leq$ is interpreted as the linear order of $\mathbb{N}$.

It is well-known that if we are interested in the complexity of first-order or monadic second-order model-checking on words, the alphabet can be assumed to be $\{0, 1\}$ without loss of generality. Thus the satisfiability problem is concerned with the following question.

FO SATISFIABILITY ON WORDS

INPUT: A $\mathsf{FO}[\tau(\{0,1\})]$-sentence ϕ.

QUESTION: Does there exists a word $w \in \{0,1\}^*$, such that $w \vDash \phi$?

The result that is previously known and motivates the upcoming reduction consists in the following.

▶ **Theorem 9** ([34]). *The* FO SATISFIABILITY ON WORDS *problem is not in* ELEMENTARY.

6.1 Reduction from FO SAT on words

The following theorem states a polynomial time reduction from FO SATISFIABILITY ON WORDS towards the problem of PARAMETRIC PUSHDOWN PARITY GAME.

▶ **Theorem 10.** FO SATISFIABILITY ON WORDS *is polynomial time reducible to* PARAMETRIC PUSHDOWN PARITY GAME.

Proof. We start with a $\mathsf{FO}[\tau(\{0,1\})]$ -formula ϕ. We construct a parametric pushdown system $\mathcal{Z}_\phi = (Q_\phi, W_\phi, \Delta_\phi)$, with a disjoint union $Q_\phi = Q_0 \uplus Q_1$, a priority mapping Ω_ϕ, and an initial state $q_I \in Q_\phi$ such that player 0 has a winning strategy from $(q_I, \$, f_I)$ in the parity game $\mathcal{G}_{\mathcal{Z}_\phi, Q_0, Q_1, \Omega_\phi}$ if and only if there exists a word $w \in \{0,1\}^*$ such that $w \vDash \phi$.

We assume without loss of generality that ϕ is written in the prenex normal form $\phi = \forall x_1 \exists x_2 \forall x_3 \ldots \exists x_k \ \psi$, where an even number k of quantifiers alternate between existential and universal ones, and ψ is quantifier-free and in disjunctive normal form.

Player 0 starts by chosing a valuation w for the first parameter (parameter p_0). The goal of this first parameter is to remember the word w, which corresponds to the word "guessed" by player 0. This parameter is meant in part to enforce that the stack remains a prefix of w, with each stack content of the play corresponding to a position in the word. To enforce this, we allow in $\mathcal{Z}$, after any decision, for a player to challenge the assumption that the last movement of the other player led to a position in the word. A player thus challenged only has the ability to add things onto the stack, and win if and only if able to find a way back to the value of parameter p_0. The gadgets for these intermediary potential challenges will be excluded from the illustrations to preserve clarity of representation. The other parameters are meant to correspond to the variables, with player 1 parameters corresponding to universal variables and player 0 parameters corresponding to existential variables.

Once players have chosen values for their respective variables, the parameter assignment μ is fixed. It is then time for player 0 to ensure that w satisfies ψ if the variables $x_1, \ldots, x_k$

are interpreted by $|\mu(p_1)|-1,\dots,|\mu(p_k)|-1$ respectively. This is the goal of the reachability game.

The first step of the game is to check that the values assigned to the parameters correspond indeed to possible variables, that is, that the values of the parameters are prefixes of w.

Recall ψ is quantifier-free and in disjunctive normal form, i.e.

$$\phi = \forall x_1 \exists x_2 \forall x_3 \dots \exists x_k \bigvee_{i=1}^{l} \left(\bigwedge_{j=1}^{h_i} \psi_{i,j}(x_1,\dots,x_k) \right).$$

Player 0, being the existential player, choses one of the conjunctive clause in ψ, essentially claiming to be able to prove it. Then player 1, the universal player, chooses one of the atomic formula in the clause to test. Testing the atomic formula is the purpose of a gadget in which player 0 has a winning strategy if and only if w satisfies the atomic formula if the variables $x_1,\dots,x_k$ are interpreted by $|f(1)|-1,\dots,|f(k)|-1$ respectively.

Now we need only to describe the gadgets $\mathcal{C}_{\psi_{i,j}}$. The gadget depends on the type of the atomic formula. For a formula checking the letter at the position of a variable x, player 0 goes to the position corresponding to x, then checks that the top of the stack x is the right letter. For one checking that $x = x'$ where x and x' are both variables, player 0 goes to the position corresponding to x and checks against the parameter corresponding to x' as well. If the formula is a negation, players exchange their roles. ◀

Then, the nonelementary lower bound follows from Theorem 9.

▶ **Theorem 11.** Parametric pushdown parity game *is not in* ELEMENTARY.

7 Conclusion

In this paper we have shown that deciding the winner of a parity game on a graph of a pebble pushdown system is nonelementary.

For the nonelementary lower bound we reduced the FO satisfiability problem on words — known to be nonelementary from [34] — to the problem of deciding whether player 0 has a winning strategy from some starting configuration in a pebble pushdown parity game.

For the upper bound, we replaced parameters by pebbles acting as registers, leading to a more general model, and then used a reducton to higher-order pushdown systems parity games to solve parity games. Since solving parity games on higher-order pushdown systems with level n stack is n-EXP-complete [8, 9], this provides an $(n+1)$-EXP upper bound for solving parametric parity games on parametric pushdown systems with n parameters.

References

1 Alfred V Aho. Nested stack automata. *Journal of the ACM (JACM)*, 16(3):383–406, 1969.

2 André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical computer science*, 303(1):7–34, 2003.

3 Mohamed Faouzi Atig. Model-checking of ordered multi-pushdown automata. *Logical Methods in Computer Science*, 8, 2012.

4 Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM Journal on Computing*, 38(2):658–701, 2008.

5 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi Reghizzi. Multi-push-down languages and grammars. *International Journal of Foundations of Computer Science*, 7(03):253–291, 1996.

6 Chris Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *International Colloquium on Automata, Languages, and Programming*, pages 165–176. Springer, 2012.

7 Thierry Cachat. Two-way tree automata solving pushdown games. In *Automata logics, and infinite games*, pages 303–317. Springer, 2002.

8 Thierry Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), Eindhoven (The Netherlands)*, number 2719 in Lecture Notes in Computer Science, pages 556–569. Springer, 2003.

9 Thierry Cachat and Igor Walukiewicz. The complexity of games on higher order pushdown automata. *arXiv preprint arXiv:0705.0262*, 2007.

10 Arnaud Carayol. *Automates infinis, logiques et langages*. PhD thesis, Université Rennes 1, 2006.

11 Arnaud Carayol and Stefan Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003), Mumbai (India)*, Lecture Notes in Computer Science. Springer, 2003.

12 Didier Caucal. On infinite terms having a decidable monadic theory. In Krzysztof Diks and Wojciech Rytter, editors, *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS 2002), Warsaw (Poland)*, number 2420 in Lecture Notes in Computer Science, pages 165–176. Springer, 2002.

13 Wojciech Czerwiński, Piotr Hofman, and Sławomir Lasota. Reachability problem for weak multi-pushdown automata. In *International Conference on Concurrency Theory*, pages 53–68. Springer, 2012.

14 Joost Engelfriet and Hendrik Jan Hoogeboom. Tree-walking pebble automata. In *Jewels are forever*, pages 72–83. Springer, 1999.

15 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *Journal of the ACM (JACM)*, 63(1):1–48, 2016.

16 Javier Esparza and Jens Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *International Conference on Foundations of Software Science and Computation Structure*, pages 14–30. Springer, 1999.

17 Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems. *Electronic Notes in Theoretical Computer Science*, 9:27–37, 1997.

18 Marie Fortin, Anca Muscholl, and Igor Walukiewicz. Model-checking linear-time properties of parametrized asynchronous shared-memory pushdown systems. In *International Conference on Computer Aided Verification*, pages 155–175. Springer, 2017.

19 Noa Globerman and David Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169(2):161–184, 1996.

20 Sheila A Greibach. Full afls and nested iterated substitution. *Information and Control*, 16(1):7–35, 1970.

21 Inène Guessarian. Pushdown tree automata. *Mathematical systems theory*, 16(1):237–263, 1983.

22 Matthew Hague. Parameterised pushdown systems with non-atomic writes. *arXiv preprint arXiv:1109.6264*, 2011.

23 John E Hopcroft and Jeffrey D Ullman. *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., 1969.

24 Vineet Kahlon. Parameterization as abstraction: A tractable approach to the dataflow analysis of concurrent programs. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 181–192. IEEE, 2008.

25 Juhani Karhumäki, Hermann Maurer, Gheorghe Paun, and Grzegorz Rozenberg. *Jewels are Forever: Contributions on Theoretical Computer Science in Honor of Arto Salomaa*. Springer Science & Business Media, 2012.

26 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In *International Conference on Foundations of Software Science and Computation Structures*, pages 205–222. Springer, 2002.

27 Orna Kupferman and Moshe Y Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *International Conference on Computer Aided Verification*, pages 36–52. Springer, 2000.

28 AN Maslov. Multilevel stack automata. *Problemy peredachi informatsii*, 12(1):55–62, 1976.

29 Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for xml transformers. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 11–22, 2000.

30 Andrzej S Murawski, Steven J Ramsay, and Nikos Tzevelekos. Reachability in pushdown register automata. *Journal of Computer and System Sciences*, 87:58–83, 2017.

31 Thomas Reps, Stefan Schwoon, Somesh Jha, and David Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Science of Computer Programming*, 58(1-2):206–263, 2005.

32 Ryoma Senda, Yoshiaki Takata, and Hiroyuki Seki. Forward regularity preservation property of register pushdown systems. *IEICE TRANSACTIONS on Information and Systems*, 104(3):370–380, 2021.

33 Ryoma Senda, Yoshiaki Takata, and Hiroyuki Seki. LTL model checking for register pushdown systems. *IEICE Transactions on Information and Systems*, 104(12):2131–2144, 2021.

34 Larry J. Stockmeyer. *The complexity of decision problems in automata and logic*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.

35 Igor Walukiewicz. Pushdown processes: Games and model checking. In *International Conference on Computer Aided Verification*, pages 62–74. Springer, 1996.

36 Stefan Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products*. Dissertation, RWTH Aachen, 2005.

37 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.