# Round-Bounded Control of Parameterized Systems*

Benedikt Bollig[1], Mathieu Lehaut[2], and Nathalie Sznajder[2]

[1] CNRS, LSV & ENS Paris-Saclay, Université Paris-Saclay, France
[2] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Abstract.** We consider systems with unboundedly many processes that communicate through shared memory. In that context, simple verification questions have a high complexity or, in the case of pushdown processes, are even undecidable. Good algorithmic properties are recovered under round-bounded verification, which restricts the system behavior to a bounded number of round-robin schedules. In this paper, we extend this approach to a game-based setting. This allows one to solve synthesis and control problems and constitutes a further step towards a theory of languages over infinite alphabets.

## 1 Introduction

Ad-hoc networks, mobile networks, cache-coherence protocols, robot swarms, and distributed algorithms have (at least) one thing in common: They are referred to as *parameterized* systems, as they are usually designed to work for *any* number of processes. The last few years have seen a multitude of approaches to parameterized verification, which aims to ensure that a system is correct no matter how many processes are involved. We refer to [15] for an overview.

Now, the above-mentioned applications are usually part of an open world, i.e., they are embedded into an environment that is not completely under the control of a system. Think of scheduling problems, in which an unspecified number of jobs have to be assigned to (a fixed number of) resources with limited capacity. The arrival of a job and its characteristics are typically not under the control of the scheduler. However, most available verification techniques are only suitable for closed systems: A system is correct if *some* or *every* possible behavior satisfies the correctness criterion, depending on whether one considers reachability or, respectively, linear-time objectives.

This paper is a step towards a theory of synthesis and *control*, which provides a more fine-grained way to reason about parameterized systems. Our system model is essentially that from [24], but defined in a way that reveals similarities with data automata/class-memory automata, a certain automata model over infinite alphabets [8, 9]. Actually, we consider *parameterized pushdown systems*, as each process has a dedicated stack to model recursion. A parameterized pushdown system distinguishes between a finite-state *global process* (sometimes

* Partly supported by ANR FREDDA (ANR-17-CE40-0013).

referred to as a *global store* or *leader process*) and a *local process*. The global process can spawn new local processes. Thus, while a system configuration contains only one global state, the number of instantiations of local processes is unbounded. Moreover, when a local process takes a transition, it is allowed to read, and modify, the global store.

So far so good. Now, it is well-known that reachability is undecidable as soon as two pushdown processes communicate through shared memory. And even when local processes are finite-state, the problem is at least as hard as reachability in Petri nets [9]. This led La Torre, Madhusudan, and Parlato to consider round-bounded verification of parameterized systems, which restricts system executions to a bounded number of round-robin schedules [24]. Not only did they show that reachability drops to PSPACE, but the corresponding fixed-point computation also turned out to be practically feasible. Moreover, they give a sound method (i.e., a sufficient criterion) for proving that all reachable states can already be reached within a bounded number of round-robin schedules. This is done using a game that is different from the one we introduce here. Actually, we extend their model by adding the possibility to distinguish, in parameterized pushdown automata, between controllable global states and uncontrollable ones.

The classical reachability problem then turns into a reachability objective in an infinite-state game. As our main result, it is shown that the winner of such a game can be computed, though in (inherently) non-elementary time. Our proof makes a detour via games on multi-pushdown systems, which are undecidable in general but decidable under a bound on the number of *phases*, each restricting the number of pop operations to a dedicated stack [5,29]. Note that round-robin schedules maintain processes in a queue fashion. However, bounding the number of rounds allows us to store both the states of a local process as well as its stack contents in a configuration of a multi-pushdown system. It is worth noting that multi-pushdown systems have been employed in [23], too, to solve seemingly different verification problems involving queues.

**Related Work.** As already mentioned, there is a large body of literature on parameterized verification, mostly focusing on closed systems (e.g., [2,4,14,15]).

Infinite-state games have been extensively studied over vector addition systems with states (VASS) (e.g., [3,7,10,12,19]). However, reachability is already undecidable for simple subclasses of VASS games, unless coverability objectives are considered. Unfortunately, the latter do not allow us to require that *all* local processes terminate in a final state. Interestingly, tight links between VASS/energy games and games played on infinite domains have recently been established [16].

Underapproximate verification goes back to Qadeer and Rehof [27]. In the realm of multi-threaded recursive programs, they restricted the number of control switches between different threads. The number of processes, however, was considered to be fixed. Another kind of bounded verification of *parameterized* systems with thread creation was studied in [6]. Contrary to our restriction, the order in which processes evolve may vary from round to round.

We believe that our results will fertilize *synthesis* of parameterized systems [18] and more classical questions whose theoretical foundations go back to the 50s and Church's synthesis problem. Let us cite Brütsch and Thomas, who observed a lack of approaches to synthesis over infinite alphabets [11]: "It is remarkable, however, that a different kind of 'infinite extension' of the Büchi-Landweber Theorem has not been addressed in the literature, namely the case where the input alphabet over which $\omega$-sequences are formed is infinite." Indeed, an execution of a parameterized system can be considered as a sequence of letters, each containing the process identifier of the process involved in performing the corresponding action. Recall that our model of parameterized systems is largely inspired by data automata/class-memory automata [8,9], which were originally defined as language acceptors over infinite alphabets. The automata studied in [11] are quite different. Since synthesis problems are often reduced to game-theoretic questions, our work can be considered as an orthogonal step towards a theory of synthesis over infinite alphabets.

**Outline.** We define parameterized pushdown systems in Section 2, where we also recall known results on reachability questions. The control problem is addressed in Section 3, and we conclude in Section 4. Missing proof details can be found at the following link: `https://hal.archives-ouvertes.fr/hal-01849206`

## 2 Reachability in Parameterized Systems

We start with some preliminary definitions.

**Words.** Let $\Sigma$ be a (possibly infinite) set. A *word* $w$ over $\Sigma$ is a finite or (countably) infinite sequence $a_0a_1a_2\dots$ of elements $a_i \in \Sigma$. Let $\Sigma^*$ denote the set of finite words over $\Sigma$, $\Sigma^\omega$ the set of infinite words, and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Given $w \in \Sigma^\infty$, we denote by $|w|$ the *length* of $w$, i.e., $|w| = n$ if $w = a_0 \dots a_{n-1} \in \Sigma^*$, and $|w| = \omega$ if $w \in \Sigma^\omega$. In particular, the length $|\varepsilon|$ of the empty word $\varepsilon$ is 0.

**Transition Systems.** A *transition system* is a triple $\mathcal{T} = (V, E, v_{\mathsf{in}})$ such that $V$ is a (possibly infinite) set of *nodes*, $E \subseteq V \times V$ is the *transition relation*, and $v_{\mathsf{in}} \in V$ is the initial node. For $(u, v) \in E$, we call $v$ a *successor* of $u$.

A *partial run* of $\mathcal{T}$ is a non-empty, finite or infinite sequence $\rho = v_0v_1v_2 \dots \in V^\infty$ such that, for all $0 < i < |\rho|$, $v_i$ is a successor of $v_{i-1}$. If, in addition, we have $v_0 = v_{\mathsf{in}}$, then we call $\rho$ a *run*. A (partial) run from $u$ to $v$ is a finite (partial) run of the form $u \dots v$. In particular, $u$ is a partial run (of length 1) from $u$ to $u$.

### 2.1 Parameterized Pushdown Systems

We consider parameterized systems in which processes may be created dynamically. Every process can manipulate a stack as well as its *local* state. Information shared by all the processes is modeled in terms of a *global* state.

**Definition 1.** *A* parameterized pushdown system *(PPS) is given by a tuple* $\mathcal{P} = (S, L, \Gamma, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \Delta, F_{\mathsf{glob}}, F_{\mathsf{loc}})$ *where*

- $S$ *is the finite set of* global states*, including the* initial global state $s_{\mathsf{in}}$*,*
- $L$ *is the finite set of* local states*, including the* initial local state $\ell_{\mathsf{in}}$*,*
- $\mathit{\Gamma}$ *is the finite* stack alphabet*,*
- $\mathit{\Delta} \subseteq (S \times L) \times (\mathit{Act} \times \mathit{\Gamma}) \times (S \times L)$ *is the* transition relation *with* $\mathit{Act} = \{\mathsf{push}, \mathsf{pop}, \mathsf{int}\}$ *(where* $\mathsf{int}$ *stands for* internal*), and*
- $F_{\mathsf{glob}} \subseteq S$ *and* $F_{\mathsf{loc}} \subseteq L$ *are the sets of* accepting global states *and* accepting local states*, respectively. We assume that* $s_{\mathsf{in}} \notin F_{\mathsf{glob}}$*.*

A *configuration* of $\mathcal{P}$ is a tuple $c = (s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k))$ where $k \in \mathbb{N}$ (possibly $k = 0$), $s \in S$ is the current global state, and, for each $p \in \{1, \ldots, k\}$, $\ell_p \in L$ and $\gamma_p \in \mathit{\Gamma}^*$ are respectively the local state and stack content of process $p$. We let $C_{\mathcal{P}}$ denote the set of configurations of $\mathcal{P}$. The *initial configuration* is $(s_{\mathsf{in}})$ and a configuration $c = (s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k))$ is *final* if $s \in F_{\mathsf{glob}}$ and $\{\ell_1, \ldots, \ell_k\} \subseteq F_{\mathsf{loc}}$. The *size* $|c|$ of a configuration $c$ is the number $k$ of processes in $c$.

The semantics of a PPS $\mathcal{P}$ is defined as a transition system $[\![\mathcal{P}]\!] = (V, E, v_{\mathsf{in}})$ where $V = C_{\mathcal{P}}$, $v_{\mathsf{in}} = (s_{\mathsf{in}})$, and the transition relation is $E = \bigcup_{p \geq 1} E_p$ with $E_p$ defining the transitions of process $p$. Actually, $E_p$ contains two types of transitions. The first type corresponds to the activity of a process that has already been created. Formally, for two configurations $(s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k))$ and $(s', (\ell'_1, \gamma'_1), \ldots, (\ell'_k, \gamma'_k))$ of size $k \geq 1$,

$$((s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k)), (s', (\ell'_1, \gamma'_1), \ldots, (\ell'_k, \gamma'_k))) \in E_p$$

if and only if $p \leq k$ and there are $op \in \mathit{Act}$ and $A \in \mathit{\Gamma}$ such that

- $((s, \ell_p), (op, A), (s', \ell'_p)) \in \mathit{\Delta}$,
- $\ell_q = \ell'_q$ and $\gamma_q = \gamma'_q$ for all $q \in \{1, \ldots, k\} \setminus \{p\}$, and
- one of the following holds: $(i)$ $op = \mathsf{push}$ and $\gamma'_p = A \cdot \gamma_p$, $(ii)$ $op = \mathsf{pop}$ and $\gamma_p = A \cdot \gamma'_p$, or $(iii)$ $op = \mathsf{int}$ and $\gamma_p = \gamma'_p$ (in which case $A$ is meaningless).

Note that the topmost stack symbol can be found at the leftmost position of $\gamma_p$.

The second type of transition is when a new process joins the system. For a configuration $(s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k))$ of size $k \geq 0$,

$$((s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k)), (s', (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k), (\ell_{k+1}, \gamma_{k+1}))) \in E_p$$

if and only if $p = k + 1$ and there are $op \in \mathit{Act}$ and $A \in \mathit{\Gamma}$ such that $((s, \ell_{\mathsf{in}}), (op, A), (s', \ell_{k+1})) \in \mathit{\Delta}$ and one of the following holds: $(i)$ $op = \mathsf{push}$ and $\gamma_{k+1} = A$, or $(ii)$ $op = \mathsf{int}$ and $\gamma_{k+1} = \varepsilon$.

A *run* of $\mathcal{P}$ is a run of the transition system $[\![\mathcal{P}]\!]$. A finite run of $\mathcal{P}$ is *accepting* if it ends in a final configuration.

Similarly, we define a *parameterized finite-state system (PFS)*, which is a PPS without stacks. That is, a PFS is a tuple $\mathcal{P} = (S, L, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \mathit{\Delta}, F_{\mathsf{glob}}, F_{\mathsf{loc}})$ where $\mathit{\Delta} \subseteq (S \times L) \times (S \times L)$ and the rest is defined as in PPS. Configurations in $C_{\mathcal{P}}$ are tuples $c = (s, \ell_1, \ldots, \ell_k)$ with $k \geq 0$. The semantics of $\mathcal{P}$ is $[\![\mathcal{P}]\!] = (C_{\mathcal{P}}, E, (s_{\mathsf{in}}))$ with $E = \bigcup_{p \geq 1} E_p$ defined as follows:

$$((s, \ell_1, \ldots, \ell_k), (s', \ell'_1, \ldots, \ell'_k)) \in E_p$$

if and only if $p \leq k$, $((s, \ell_p), (s', \ell'_p)) \in \Delta$, and $\ell_q = \ell'_q$ for all $q \neq p$, and

$$((s, \ell_1, \ldots, \ell_k), (s', \ell_1, \ldots, \ell_k, \ell_{k+1})) \in E_p$$

if and only if $p = k + 1$ and $((s, \ell_{\mathsf{in}}), (s', \ell_{k+1})) \in \Delta$. The notions of runs and accepting runs are defined accordingly.

**Reachability Problems.** Consider Table 1. The problem PPS-Reachability (respectively, PFS-Reachability) consists in deciding if, in a given PPS (respectively, PFS), there is an accepting run, starting in the initial configuration.

In the general case, these problems are already known and we recall here the results. The first is folklore (cf. also [28]), as two stacks are already sufficient to simulate a Turing machine. For the second, we observe that parameterized systems without stacks are essentially Petri nets (cf. [9]).

**Theorem 1.** PPS-Reachability *is undecidable, while* PFS-Reachability *is decidable (and as hard as Petri-net reachability).*

## 2.2 Round-Bounded Behaviors

To regain decidability in the case of PPS, we restrict ourselves to runs that are *round-bounded*, a notion introduced in [24]. Intuitively, during a *round*, the first process will do any number of transitions (possibly 0), then the second process will do any number of transitions, and so on. Once process $p + 1$ has started performing transitions, process $p$ cannot act again in this round. A run is then said to be *B-round bounded* if it uses at most $B$ rounds. Formally, given a natural number $B \geq 1$ and a PPS $\mathcal{P} = (S, L, \Gamma, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \Delta, F_{\mathsf{glob}}, F_{\mathsf{loc}})$, we define the *bounded semantics* of $\mathcal{P}$ as the transition system $[\![\mathcal{P}]\!]^B = (V^B, E^B, v^B_{\mathsf{in}})$ where

- nodes are *enhanced configurations* of the form $v = (c, p, r)$ with $c \in C_{\mathcal{P}}$ a configuration, say, of size $k$, $p \in \{0, \ldots, k\}$ represents the last process that made a transition (or 0 if it is not yet defined), and $r \in \{1, \ldots, B\}$ is the number of the current round,
- the initial node is $v^B_{\mathsf{in}} = ((s_{\mathsf{in}}), 0, 1)$, and
- there is an edge between $(c, p, r)$ and $(c', p', r')$ if, in $[\![\mathcal{P}]\!] = (V, E, v_{\mathsf{in}})$, there is an edge $(c, c')$ in $E_{p'}$ and either
  - $p' \geq p$ and $r' = r$, or
  - $p' < p$, $r < B$, and $r' = r + 1$.

The bounded semantics of a PFS is defined accordingly.

A *B-run* (or simply *run* if $B$ is understood) of $\mathcal{P}$ is a run of $[\![\mathcal{P}]\!]^B$. A $B$-run is *accepting* if it is finite and ends in a node $(c, p, r)$ where $c$ is a final configuration.

Consider the problems on the right-hand side of Table 1 (note that $B$ is encoded in unary). Deciding the existence of an accepting $B$-run is PSPACE-complete for both PPS and PFS.

**Table 1.** Reachability Problems

| PPS-Reachability | PPS-Reachability$^{\mathrm{rb}}$ |
|---|---|
| **I:** PPS $\mathcal{P}$ | **I:** PPS $\mathcal{P}$; $B \geq 1$ (given in unary) |
| **Q:** Is there an accepting run of $\mathcal{P}$? | **Q:** Is there an accepting $B$-run of $\mathcal{P}$? |

| PFS-Reachability | PFS-Reachability$^{\mathrm{rb}}$ |
|---|---|
| **I:** PFS $\mathcal{P}$ | **I:** PFS $\mathcal{P}$; $B \geq 1$ (given in unary) |
| **Q:** Is there an accepting run of $\mathcal{P}$? | **Q:** Is there an accepting $B$-run of $\mathcal{P}$? |

**Theorem 2.** PPS-Reachability$^{\mathrm{rb}}$ *and* PFS-Reachability$^{\mathrm{rb}}$ *are PSPACE-complete.*

The rest of this section is devoted to the proof of this theorem. Actually, we prove that PPS-Reachability$^{\mathrm{rb}}$ is in PSPACE and PFS-Reachability$^{\mathrm{rb}}$ is PSPACE-hard. The upper bound has already been stated in [24], the lower bound in [25], for a similar model. For the sake of completeness, we give proofs for both bounds.

**PPS-Reachability$^{\mathrm{rb}}$ is in PSPACE.** We give an (N)PSPACE algorithm solving the problem PPS-Reachability$^{\mathrm{rb}}$ using a slight variant of the notion of interfaces as described in [24]. Let $\mathcal{P} = (S, L, \Gamma, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \Delta, F_{\mathsf{glob}}, F_{\mathsf{loc}})$ be a PPS and $B \geq 1$ be the maximal number of rounds.

An interface for a single process is a triple $\mathcal{I} = [t, (s_1, \ldots, s_B), (s'_1, \ldots, s'_B)] \in \{1, \ldots, B\} \times S^B \times S^B$ satisfying the following conditions:

1. For all $1 \leq i < t$, we have $s_i = s'_i$.
2. There are local states $\ell_{t-1}, \ldots, \ell_B$ and stack contents $\gamma_{t-1}, \ldots, \gamma_B$ such that (*i*) for all $t \leq i \leq B$ there is a finite partial run in $[\![\mathcal{P}]\!]$ from $c_i = (s_i, (\ell_{i-1}, \gamma_{i-1}))$ to $c'_i = (s'_i, (\ell_i, \gamma_i))$, (*ii*) this run has length at least two (i.e., it performs at least one transition) if $i = t$, and (*iii*) $\ell_{t-1}$ is the initial local state, $\gamma_{t-1} = \varepsilon$, and $\ell_B$ is an accepting local state.

We refer to the first $B$-tuple of $\mathcal{I}$ as $\mathcal{I}^\ell$ and to the second $B$-tuple as $\mathcal{I}^r$. The natural number $t$ is the starting round and is referred to as $t_{\mathcal{I}}$. We say that an interface $\mathcal{I}_1$ is *compatible* with an interface $\mathcal{I}_2$ if $t_{\mathcal{I}_1} \leq t_{\mathcal{I}_2}$ and $\mathcal{I}_1^r = \mathcal{I}_2^\ell$.

Intuitively, an interface represents the possibility of a computation of a single process during a run of the PPS. Global states are the only piece of information needed to be able to coordinate between different processes, since a process cannot access the local content of another one. Moreover, when a process is created, it takes the last position in a round. The starting round $t$ of each interface is needed to check that the order of the processes respects the order of their creation. In other words, interfaces can be viewed as the skeleton of a run of $\mathcal{P}$. This is formalised in the following lemma, which is illustrated in Figure 1.

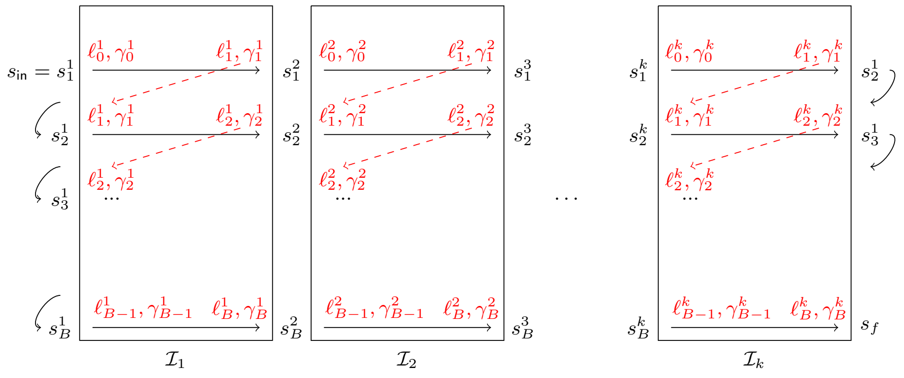


**Fig. 1.** A run as the composition of compatible interfaces; all starting rounds are 1

**Lemma 1.** *There is an accepting $B$-run of $\mathcal{P}$ if and only if there are $k$ interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ for $k \geq 1$ verifying the following conditions:*

- *For all $1 < i \leq k$, $\mathcal{I}_{i-1}$ is compatible with $\mathcal{I}_i$.*
- *Let $\mathcal{I}_1^\ell = (s_1, \ldots, s_B)$ and $\mathcal{I}_k^r = (s'_1, \ldots, s'_B)$. Then, $s_1$ is the initial global state $s_{\mathsf{in}}$, $s'_B$ is an accepting global state, and $s_j = s'_{j-1}$ for all $1 < j \leq B$.*

Given $\mathcal{I} = [t, (s_1, \ldots, s_B), (s'_1, \ldots, s'_B)]$, one can check in polynomial time whether $\mathcal{I}$ is an interface. To do this, we check the emptiness of a pushdown automaton that simulates the actions of $\mathcal{P}$ on a single process and has special transitions to change the global state from $s'_j$ to $s_{j+1}$. As non-emptiness of a pushdown automaton can be checked in polynomial time [17], so can the validity of a given interface.

The algorithm to solve PPS-Reachability$^{\mathrm{rb}}$ first guesses an interface $\mathcal{I}_1$ for the first process, and stores $t_{\mathcal{I}_1}$, $\mathcal{I}_1^\ell$, and $\mathcal{I}_1^r$. Then, it guesses an interface $\mathcal{I}_2$ for the second process, checks that it is compatible by comparing $t_{\mathcal{I}_2}$ and $\mathcal{I}_2^\ell$ with the previously stored $t_{\mathcal{I}_1}$ and $\mathcal{I}_1^r$, and then replaces $\mathcal{I}_1^r$ by $\mathcal{I}_2^r$ and $t_{\mathcal{I}_1}$ by $t_{\mathcal{I}_2}$ (so only $\mathcal{I}_1^\ell$, $t_{\mathcal{I}_2}$, and $\mathcal{I}_2^r$ are stored). We continue guessing compatible interfaces, storing at each step $i$ the values of $\mathcal{I}_1^\ell$, $t_{\mathcal{I}_i}$, and $\mathcal{I}_i^r$. Eventually, the algorithm guesses that the last process has been reached. At that point, there are two halves of interfaces stored in memory: the left interface $\mathcal{I}_1^\ell = (s_1, \ldots, s_B)$ of the first process, and the right interface $\mathcal{I}_k^r = (s'_1, \ldots, s'_B)$ of the last process. We accept if, for all $i \in \{1, \ldots, B-1\}$, we have that $s'_i = s_{i+1}$, $s_1 = s_{\mathsf{in}}$, and $s'_B \in F_{\mathsf{glob}}$. By Lemma 1, there is an accepting $B$-run of $\mathcal{P}$.

**PFS-Reachability$^{\mathrm{rb}}$ is PSPACE-hard.** This can be shown by a reduction from the non-emptiness of the intersection of a collection of finite automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, which is PSPACE-complete [21]. The bound $B$ on the number of rounds will be $n$. We construct a PFS that non-deterministically guesses a word $w$ in the first round. Moreover, in round $i$, it will check that $w$ is accepted by $\mathcal{A}_i$. To do this, each process simulates one transition of $\mathcal{A}_i$ on one letter of $w$. That is, the number of processes is $|w|$. Each process performs exactly one action each round, and, to ensure that the word $w$ is the same for each $\mathcal{A}_i$, stores the corresponding letter in its local state. The global state stores the state of the currently simulated automaton.

## 3 Round-Bounded Control of Parameterized Systems

We will extend parameterized pushdown systems to a game-based setting with the aim of modeling systems with a centralized control that are embedded into an uncontrollable environment.

### 3.1 Parameterized Pushdown Games

**Games.** A *game* is given by an *arena*, i.e., a transition system $\mathcal{G} = (V, E, v_{\mathsf{in}})$ where $V = V_0 \uplus V_1$ is partitioned into the set of states controlled by Player 0 and Player 1, respectively, along with a winning condition $\mathcal{W} \subseteq V^\infty$.

A *play* of $\mathcal{G}$ is a run of the underlying transition system. A play is *maximal* if it is infinite, or ends in a node that has no successor. A maximal play is *winning* for Player 0 if it is in $\mathcal{W}$, otherwise it is winning for Player 1.

We will be concerned with two winning conditions: A *reachability condition* is given by a set of nodes $\mathcal{F} \subseteq V$. It induces the set $\mathcal{W}_{\mathcal{F}} = \{\rho = v_0 v_1 v_2 \ldots \in V^\infty \mid v_i \in \mathcal{F}$ for some $0 \leq i < |\rho|\}$. A *parity condition* is given by a ranking function $\alpha : V \to Col$ where $Col \subseteq \mathbb{N}$ is a finite set of colors. It induces the set $\mathcal{W}_\alpha = \{\rho \in V^\omega \mid \min(\mathrm{Inf}_\alpha(\rho))$ is even$\}$ with $\mathrm{Inf}_\alpha(v_0 v_1 v_2 \ldots) = \{m \in Col \mid m$ appears infinitely often in $\alpha(v_0)\alpha(v_1)\alpha(v_2)\ldots\}$. I.e., $\mathcal{W}_\alpha$ contains an infinite run if and only if the minimal color seen infinitely often is even.

Let $j \in \{0, 1\}$. A *strategy* for Player $j$ is a partial mapping $f_j : V^* V_j \to V$ such that, for all $w \in V^*$ and $v \in V_j$, the following hold: if $f_j(wv)$ is defined, then $(v, f_j(wv)) \in E$; otherwise, $v$ has no successor.

Fix strategies $f_0$ and $f_1$ for Players 0 and 1, respectively. An $(f_0, f_1)$-*play* of $\mathcal{G}$ is a maximal play $\rho = v_0 v_1 v_2 \ldots$ such that, for all $0 < i < |\rho|$ and $j \in \{0, 1\}$, if $v_{i-1} \in V_j$, then $f_j(v_0 \ldots v_{i-1}) = v_i$.

We say that $f_j$ is *winning* if, for all strategies $f_{1-j}$, the unique maximal $(f_0, f_1)$-play is winning for Player $j$. A game is *determined* if either Player 0 has a winning strategy, or Player 1 has a winning strategy. Furthermore, we say that $f_j$ is *memoryless* if, for all $w, w' \in V^*$ and $v \in V_j$, we have $f_j(wv) = f_j(w'v)$, i.e., the strategy only depends on the last node.

**Theorem 3 (cf. [13, 33]).** *Games with a parity winning condition are determined, and if Player $j$ has a winning strategy, then Player $j$ has a winning memoryless strategy.*

**Parameterized Pushdown Games.** We now introduce the special case of games played on the infinite transition system induced by a round-bounded PPS.

A round-bounded parameterized pushdown game is described by a PPS $\mathcal{P} = (S, L, \Gamma, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \Delta, F_{\mathsf{glob}}, F_{\mathsf{loc}})$ together with a partition $S = S_0 \uplus S_1$. For a bound $B \geq 1$, the *B-round-bounded parameterized pushdown game* induced by $\mathcal{P}$ is the game $\mathcal{G}_{\mathcal{P}}^{B}$ given by the transition system $[\![\mathcal{P}]\!]^{B} = (V^{B}, E^{B}, v_{\mathsf{in}}^{B})$ where a node $v = (c, p, r) \in V^{B}$ with $c = (s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k))$ belongs to Player $j$ if $s \in S_j$. We consider the reachability winning condition $\mathcal{W}_{\mathcal{F}}$ given by $\mathcal{F} = \{(c, p, r) \in V^{B} \mid c \text{ is a final configuration of } \mathcal{P}\}$. Since a reachability game can be easily transformed into a parity game, Theorem 3 implies that $\mathcal{G}_{\mathcal{P}}^{B}$ is determined.

Parameterized games on PFS are defined similarly as for PPS. Note that, without a bound on the number of rounds, games on PFS are already undecidable, which is shown by an easy adaptation of the undecidability proof for VASS games [1]. Therefore, we only define control for round-bounded games:

| CONTROL$^{\text{rb}}$ |
|---|
| **I:** PPS $\mathcal{P} = (S_0 \uplus S_1, L, \Gamma, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \Delta, F_{\mathsf{glob}}, F_{\mathsf{loc}})$; $B \geq 1$ |
| **Q:** Does Player 0 have a winning strategy in $\mathcal{G}_{\mathcal{P}}^{B}$ ? |

We are now ready to present our main result, which is shown in the remainder of this section:

**Theorem 4.** CONTROL$^{\text{rb}}$ *is decidable, and inherently non-elementary.*

## 3.2 Upper bound

Decidability of CONTROL$^{\text{rb}}$ comes from decidability of games on phase-bounded multi-pushdown systems (short: multi-pushdown games), which were first studied in [29] and rely on the phase-bounded multi-pushdown automata from [22].

**Multi-Pushdown Games.** Intuitively, a *phase* is a sequence of actions in a run during which only one fixed "active" stack can be read (i.e., either make a pop transition or a zero-test transition), but push and internal transitions are unrestricted. There are no other constraints on the number of transitions or the order of the transitions done during a phase.

**Definition 2.** *A* multi-pushdown system (MPS) *is a tuple* $\mathcal{M} = (\kappa, N, S_0 \uplus S_1, \Gamma, \Delta, s_{\mathsf{in}}, \alpha)$ *where the natural number* $\kappa \geq 1$ *is the* phase bound, $N \in \mathbb{N}$ *is the number of stacks,* $S = S_0 \uplus S_1$ *is the partitioned finite set of* states, $\Gamma$ *is the finite* stack alphabet, $\Delta \subseteq S \times \mathit{Act}_{\mathsf{zero}} \times \{1, \ldots, N\} \times \Gamma \times S$ *is the* transition relation *where* $\mathit{Act}_{\mathsf{zero}} = \{\mathsf{push}, \mathsf{pop}, \mathsf{int}, \mathsf{zero}\}$, $s_{\mathsf{in}} \in S$ *is the initial state, and* $\alpha : S \to \mathit{Col}$ *with* $\mathit{Col} \subseteq \mathbb{N}$ *a finite set is the* ranking function.

The associated game $\mathcal{G}_{\mathcal{M}}$ is then played on the transition system $[\![\mathcal{M}]\!] = (V = V_0 \uplus V_1, E, v_{\mathsf{in}})$ defined as follows.

A node $v \in V$ is of the form $v = (s, \gamma_1, \ldots, \gamma_N, \mathit{st}, \mathit{ph})$ where $s \in S$, $\gamma_\sigma \in \Gamma^*$ is the content of stack $\sigma$, and $\mathit{st} \in \{0, \ldots, N\}$ and $\mathit{ph} \in \{1, \ldots, \kappa\}$ are used to keep track of the current active stack (0 when it is undefined) and the current phase, respectively. For $j \in \{0, 1\}$, we let $V_j = \{(s, \gamma_1, \ldots, \gamma_N, \mathit{st}, \mathit{ph}) \in V \mid s \in S_j\}$.

Given nodes $v = (s, \gamma_1, \ldots, \gamma_N, \mathit{st}, \mathit{ph}) \in V$ and $v' = (s', \gamma'_1, \ldots, \gamma'_N, \mathit{st}', \mathit{ph}') \in V$, we have an edge $(v, v') \in E$ if and only if there exist $\mathit{op} \in \mathit{Act}_{\mathsf{zero}}$, $\sigma \in \{1, \ldots, N\}$, and $A \in \Gamma$ such that $(s, \mathit{op}, \sigma, A, s') \in \Delta$ and the following hold:

- $\gamma_\tau = \gamma'_\tau$ for all $\tau \neq \sigma$,
- $\gamma_\sigma = \gamma'_\sigma$ if $op = \mathsf{int}$, $\gamma'_\sigma = A \cdot \gamma_\sigma$ if $op = \mathsf{push}$, $\gamma_\sigma = A \cdot \gamma'_\sigma$ if $op = \mathsf{pop}$, and $\gamma_\sigma = \gamma'_\sigma = \varepsilon$ if $op = \mathsf{zero}$,
- if $op \in \{\mathsf{int}, \mathsf{push}\}$, then $st = st'$ and $ph = ph'$ (the active stack and, hence, the phase do not change),
- if $op \in \{\mathsf{pop}, \mathsf{zero}\}$, then either $st = 0$, $st' = \sigma$, and $ph = ph' = 1$ (this is the first time a current stack is defined), or $st = \sigma$, $st' = \sigma$, and $ph = ph'$ (the stack $\sigma$ corresponds to the current active stack), or $st \neq \sigma$, $ph < \kappa$, $st' = \sigma$, and $ph' = ph + 1$ (stack $\sigma$ is not the active stack so that a new phase starts).

The initial node is $v_{\mathsf{in}} = (s_{\mathsf{in}}, \varepsilon, \ldots, \varepsilon, 0, 1)$. The winning condition of $\mathcal{G}_\mathcal{M}$ is a parity condition given by $\overline{\alpha} : V \to \mathit{Col}$ where, for $v = (s, \gamma_1, \ldots, \gamma_N, st, ph)$, we let $\overline{\alpha}(v) = \alpha(s)$.

The control problem for MPS, denoted by Control$^{\mathrm{MPS}}$, is defined as follows: Given an MPS $\mathcal{M}$, does Player 0 have a winning strategy in $\mathcal{G}_\mathcal{M}$?

**Theorem 5 ([5,29]).** Control$^{\mathrm{MPS}}$ *is decidable, and is non-elementary in the number of phases.*

The upper bound was first shown in [29] by adopting the technique from [32], which reduces pushdown games to games played on finite-state arenas. On the other hand, [5] proceeds by induction on the number of phases, reducing a ($\kappa$+1)-phase game to a $\kappa$-phase game. Similarly, we could try a direct proof of our Theorem 4 by induction on the number of rounds. However, this proof would be very technical and essentially reduce round-bounded parameterized systems to multi-pushdown systems. Therefore, we proceed by reduction to multi-pushdown games, providing a modular proof with clearly separated parts.

**From Parameterized Pushdown Games to Multi-Pushdown Games.** We reduce Control$^{\mathrm{rb}}$ to Control$^{\mathrm{MPS}}$. Let $\mathcal{P} = (S, L, \mathit{\Gamma}, s_{\mathsf{in}}, \ell_{\mathsf{in}}, \mathit{\Delta}, F_{\mathsf{glob}}, F_{\mathsf{loc}})$, with $S = S_0 \uplus S_1$, be a PPS and $B \geq 1$. We will build an MPS $\mathcal{M}$ such that Player 0 has a winning strategy in $\mathcal{G}^B_\mathcal{P}$ if and only if Player 0 has a winning strategy in $\mathcal{G}_\mathcal{M}$. In the following, given $s \in S$, we let $pl(s) \in \{0, 1\}$ denote the player associated with $s$, i.e., $pl(s) = 0$ if and only if $s \in S_0$.

The main idea of the reduction is to represent a configuration

$$(s, (\ell_1, \boxed{\uparrow \gamma_1}), \ldots, (\ell_{p-1}, \boxed{\uparrow \gamma_{p-1}}), (\ell_p, \boxed{\uparrow \gamma_p}), (\ell_{p+1}, \boxed{\uparrow \gamma_{p+1}}), \ldots, (\ell_k, \boxed{\uparrow \gamma_k}), p, r)$$

of $\mathcal{G}^B_\mathcal{P}$ as a configuration in $\mathcal{G}_\mathcal{M}$ of the form depicted in Figure 2.

Component $j \in \{0, 1\}$ of the global state denotes the current player (which, by default, is $pl(s)$). We explain $f_1$ and $f_2$ further below.

The process $p$ that has moved last is considered as the *active* process whose local state $\ell_p$ is kept in the global state of $\mathcal{G}_\mathcal{M}$ along with $s$, and whose stack contents $\gamma_p$ is accessible on stack 1 (in the correct order). This allows the multi-pushdown game to simulate transitions of process $p$, modifying its local state and stack contents accordingly (see *Basic Transitions* in the formalization below).

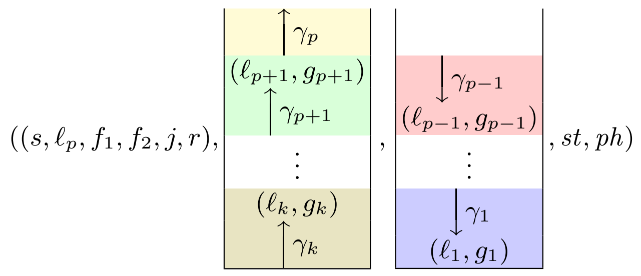


**Fig. 2.** Encoding of a configuration in $\mathcal{G}_{\mathcal{P}}^{B}$ by a configuration in $\mathcal{G}_{\mathcal{M}}$

If a player decides to take a transition for some process $p' > p$, she will store $\ell_p$ on stack 2 and shift the contents of stack 1 onto stack 2 until she retrieves the local state $\ell_{p'}$ of $p'$ along with its stack contents $\gamma_{p'}$ (see Figure 3 and *Transitions for Process Change* in the formalization of $\mathcal{M}$).

If, on the other hand, the player decides to take a transition for some process $p' < p$, then she stores $\ell_p$ on stack 1 and shifts the contents of stack 2 onto stack 1 to recover the local state $\ell_{p'}$ and stack contents $\gamma_{p'}$ (see Figure 4 and *Transitions for Round Change*). This may imply two phase switches, one to shift stack symbols from 2 to 1, and another one to continue simulating the current process on stack 1. However, $2B - 1$ phases are sufficient to simulate $B$ rounds.

There are a few subtleties: First, at any time, we need to know whether the current configuration of $\mathcal{G}_{\mathcal{M}}$ corresponds to a final configuration in $\mathcal{G}_{\mathcal{P}}^{B}$. To this aim, the state component $(s, \ell_p, f_1, f_2, j, r)$ of $\mathcal{M}$ contains the flags $f_1, f_2 \in \{\checkmark, \text{✗}\}$ where, as an invariant, we maintain $f_1 = \checkmark$ if and only if $\{\ell_{p+1}, \ldots, \ell_k\} \subseteq F_{\mathsf{loc}}$ and $f_2 = \checkmark$ if and only if $\{\ell_1, \ldots, \ell_{p-1}\} \subseteq F_{\mathsf{loc}}$. Thus, Player 0 wins in $\mathcal{G}_{\mathcal{M}}$ as soon as she reaches a configuration with global state $(s, \ell, f_1, f_2, j, r)$ such that $s \in F_{\mathsf{glob}}$, $\ell \in F_{\mathsf{loc}}$, and $f_1 = f_2 = \checkmark$. To faithfully maintain the invariant, every local state $\ell_q$ that is pushed on one of the two stacks, comes with an additional flag $g_q \in \{\checkmark, \text{✗}\}$, which is $\checkmark$ if and only if all local states *strictly below* on the stack are contained in $F_{\mathsf{loc}}$. It is then possible to keep track of a property of *all* local states on a given stack simply by inspecting and locally updating the topmost stack symbols.

Second, one single transition in $\mathcal{P}$ is potentially simulated by several transitions in $\mathcal{M}$ in terms of the gadgets given in Figures 3 and 4. The problem here is that once Player $j$ commits to taking a transition by entering a gadget, she is not allowed to get stuck. To ensure progress, there are transitions from inside a gadget to a state $\mathrm{win}_{1-j}$ that is winning for Player $1 - j$.

Third, suppose that, in a non-final configuration of $\mathcal{G}_{\mathcal{P}}^{B}$, it is Player 1's turn, but no transition is available. Then, Player 1 wins the play. But how can Player 1 prove in $\mathcal{G}_{\mathcal{M}}$ that no transition is available in the original game $\mathcal{G}_{\mathcal{P}}^{B}$? Actually, he will give the control to Player 0, who will eventually get stuck and, therefore, lose (cf. transitions for *Change of Player* below).

Let us define the MPS $\mathcal{M} = (\kappa, N, S' = S'_0 \uplus S'_1, \Gamma', \Delta', s'_{\mathsf{in}}, \alpha)$ formally. We let $\kappa = 2B - 1$, $N = 2$ (the number of stacks), and $\Gamma' = \Gamma \uplus (L \times \{✓, ✗\})$.

**States.** The set of states is $S' = \{s'_{\mathsf{in}}\} \uplus S_{\text{sim}} \uplus \{\text{win}_0, \text{win}_1\} \uplus \mathfrak{I}$ where $s'_{\mathsf{in}}$ is the initial state. Moreover, $S_{\text{sim}} = S \times L \times \{✓, ✗\}^2 \times \{0, 1\} \times \{1, \ldots, B\}$. A state $(s, \ell, f_1, f_2, j, r) \in S_{\text{sim}}$ stores the global state $s$ and the local state $\ell$ of the last process $p$ that executed a transition. The third and forth component $f_1$ and $f_2$ tell us whether all processes $p' > p$ and, respectively, $p' < p$ of the current configuration are in a local final state (indicated by ✓). Then, $j$ denotes the player that is about to play (usually, we have $j = pl(s)$, but there will be deviations). Finally, $r$ is the current round that is simulated. Recall that $(s, \ell, f_1, f_2, j, r)$ represents a final configuration if and only if $s \in F_{\mathsf{glob}}$, $\ell \in F_{\mathsf{loc}}$, and $f_1 = f_2 = ✓$. Let $\mathfrak{F} \subseteq S_{\text{sim}}$ be the set of such states. The states $\text{win}_0$ and $\text{win}_1$ are self-explanatory. Finally, we use several intermediate states, contained in $\mathfrak{I}$, which will be determined below along with the transitions.

The partition $S' = S'_0 \uplus S'_1$ is defined as follows: First, we have $s'_{\mathsf{in}} \in S'_{pl(s_{\mathsf{in}})}$. Concerning states from $S_{\text{sim}}$, we let $(s, \ell, f_1, f_2, j, r) \in S'_j$. The states $\text{win}_0$ and $\text{win}_1$ both belong to Player 0 (but this does not really matter). Membership of intermediate states is defined below. The ranking function $\alpha$ maps $\text{win}_0$ to 0, and everything else to 1. In fact, we only need a reachability objective and use the parity condition to a very limited extent.

**Initial Transitions.** For all transitions $(s_{\mathsf{in}}, \ell_{\mathsf{in}}) \xrightarrow{(op,A)} (s', \ell')$ in $\mathcal{P}$, we introduce, in $\mathcal{M}$, a transition $s'_{\mathsf{in}} \xrightarrow{(op,1,A)} (s', \ell', ✓, ✓, pl(s'), 1)$.

**Final Transitions.** For all states $(s, \ell, f_1, f_2, j, r) \in \mathfrak{F}$, we will have a transition $(s, \ell, f_1, f_2, j, r) \xrightarrow{\mathsf{int}} \text{win}_0$ (we omit the stack symbol, as it is meaningless), which will be the only transition outgoing from $(s, \ell, f_1, f_2, j, r)$. Moreover, $\text{win}_0 \xrightarrow{\mathsf{int}} \text{win}_0$ and $\text{win}_1 \xrightarrow{\mathsf{int}} \text{win}_1$.

**Basic Transitions.** We now define the transitions of $\mathcal{M}$ simulating transitions of $\mathcal{P}$ that do not change the process. For all $(s, \ell, f_1, f_2, j, r) \in S_{\text{sim}} \setminus \mathfrak{F}$ and transitions $(s, \ell) \xrightarrow{(op,A)} (s', \ell')$ from $\Delta$ (in $\mathcal{P}$), the MPS $\mathcal{M}$ has a transition $(s, \ell, f_1, f_2, j, r) \xrightarrow{(op,1,A)} (s', \ell', f_1, f_2, pl(s'), r)$.

**Transitions for Process Change.** For all $(s, \ell, f_1, f_2, j, r) \in S_{\text{sim}} \setminus \mathfrak{F}$, we introduce, in $\mathcal{M}$, the gadget given in Figure 3. As we move to another process, the current local state $\ell$ is pushed on stack 2, along with flag $f_2$, which tells us whether, henceforth, all states on stack 2 *below* the new stack symbol are local accepting states. Afterwards, the value of $f_2$ kept in the global state has to be updated, depending on whether $\ell \in F_{\mathsf{loc}}$ or not. Actually, maintaining the value of $f_2$ is done in terms of additional (but finitely many) states. For the sake of readability, however, we rather consider that $f_2$ is a variable and use $\mathsf{upd}(f_2, \ell)$ to update its value. We continue shifting the contents of stack 1 onto stack 2 (updating $f_2$ when retrieving a local state). Now, there are two possibilities. We

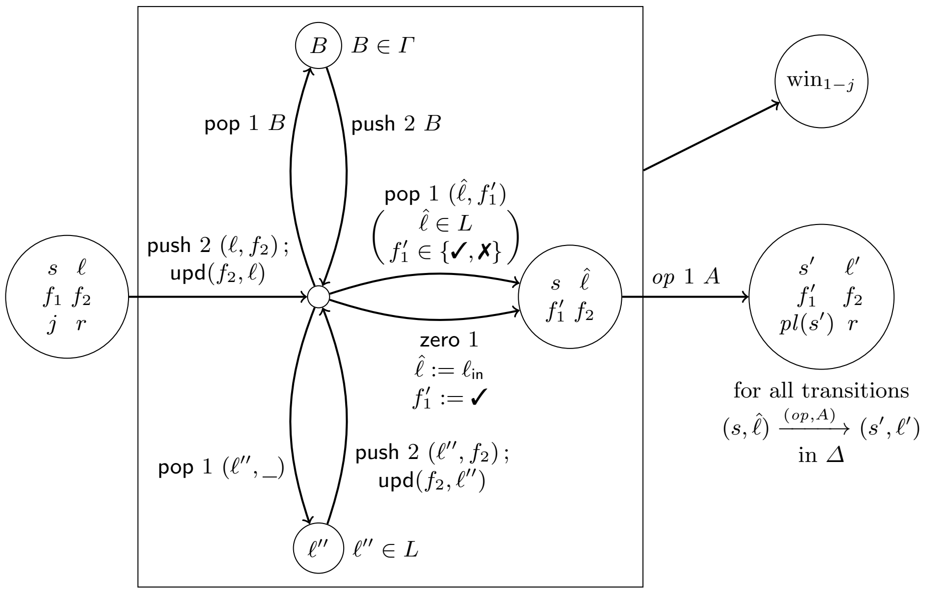


**Fig. 3.** Change from process $p$ to some process $p' > p$ (staying in the same round). All intermediate states belong to Player $j$; from every intermediate state, there is an outgoing internal transition to win$_{1-j}$. Moreover, $\mathsf{upd}(f_2, \bar{\ell})$ stands for the update rule IF $(f_2 = ✓ \wedge \bar{\ell} \in F_{\mathsf{loc}})$ THEN $f_2 := ✓$ ELSE $f_2 := ✗$.

may eventually pop a new current local state $\hat{\ell}$ and then simulate the transition of the corresponding existing process. Or, when there are no more symbols on stack 1, we create a new process.

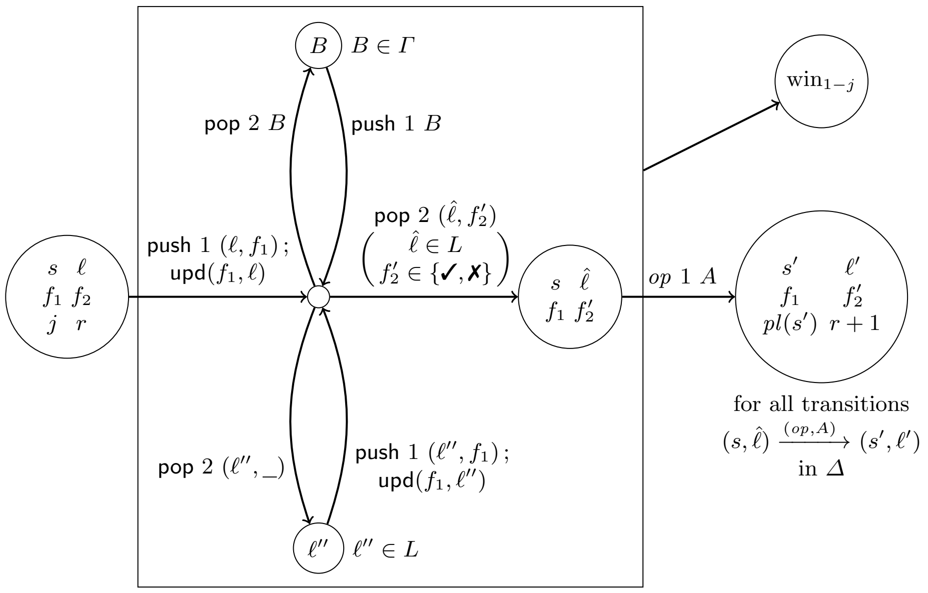
**Transitions for Round Change.** For all $(s, \ell, f_1, f_2, j, r) \in S_{\text{sim}} \setminus \mathfrak{F}$ such that $r < B$, we introduce, in $\mathcal{M}$, the gadget given in Figure 4. It is similar to the previous gadget. However, we now shift symbols from stack 2 onto stack 1 and have to update $f_1$ accordingly.

**Change of Player.** When Player 1 thinks he does not have an outgoing transitions (in $\mathcal{P}$), he can give the token to Player 0. That is, for all $(s, \ell, f_1, f_2, 1, r) \in S_{\text{sim}} \setminus \mathfrak{F}$, we introduce the transition $(s, \ell, f_1, f_2, 1, r) \xrightarrow{\mathsf{int}} (s, \ell, f_1, f_2, 0, r)$.

**Lemma 2.** *Player 0 has a winning strategy in $\mathcal{G}_{\mathcal{M}}$ if and only if Player 0 has a winning strategy in $\mathcal{G}^B_{\mathcal{P}}$.*

## 3.3 Lower bound

Our lower-bound proof is inspired by [5], but we reduce from the satisfiability problem for first-order formulas on finite words, which is known to be non-elementary [30]. Note that the lower bound already holds for PFS.



$B$ $B \in \Gamma$
pop 2 $B$
push 1 $B$
win$_{1-j}$
pop 2 $(\hat{\ell}, f_2')$
$\begin{pmatrix} \hat{\ell} \in L \\ f_2' \in \{✓, ✗\} \end{pmatrix}$
push 1 $(\ell, f_1)$ ;
upd$(f_1, \ell)$
$s$ $\ell$ $f_1$ $f_2$ $j$ $r$
$s$ $\hat{\ell}$ $f_1$ $f_2'$
$op$ 1 $A$
$s'$ $\ell'$ $f_1$ $f_2'$ $pl(s')$ $r+1$
for all transitions
$(s, \hat{\ell}) \xrightarrow{(op,A)} (s', \ell')$
in $\Delta$
pop 2 $(\ell'', \_)$
push 1 $(\ell'', f_1)$ ;
upd$(f_1, \ell'')$
$\ell''$ $\ell'' \in L$

**Fig. 4.** Go from a process $p$ to some process $p' < p$ (involving a round change). All intermediate states belong to Player $j$; from every intermediate state, there is an outgoing internal transition to win$_{1-j}$. Moreover, upd$(f_1, \bar{\ell})$ stands for the update rule If $(f_1 = ✓ \wedge \bar{\ell} \in F_{\mathsf{loc}})$ Then $f_1 := ✓$ Else $f_1 := ✗$.

Let Var be a countably infinite set of variables and $\Sigma$ a finite alphabet. Formulas $\varphi$ are built by the grammar $\varphi ::= a(x) \mid x < y \mid \neg(x < y) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \forall x.\varphi$ where $x, y \in \text{Var}$ and $a \in \Sigma$.

Let $w = a_0 \dots a_{n-1} \in \Sigma^*$ be a word. Variables are interpreted as positions of $w$, so a valuation is a (partial) function $\nu : \text{Var} \to \{0, \dots, n-1\}$. The satisfaction relation is defined as follows. We let $w, \nu \models a(x)$ if and only if $a_{\nu(x)} = a$. Moreover, $w, \nu \models x < y$ if and only if $\nu(x) < \nu(y)$. Quantification, negation, disjunction, and conjunction are defined as usual. We refer to [31] for details. A formula $\varphi$ without free variables is *satisfiable* if there is a word $w$ such that $w, \emptyset \models \varphi$. We suppose that $\varphi$ is given in prenex normal form.

We build a PFS-based round-bounded game that is winning for Player 0 if and only if $\varphi$ is satisfiable. In the first round of the game, Player 0 chooses a word $w$ by creating a different process for each letter of $w$, each of them holding the corresponding letter in its local state. To prove that $w$ is indeed a model of $\varphi$, the following rounds are devoted to the valuation of the variables appearing in $\varphi$, $\nu(x) = i$ being represented by memorizing the variable $x$ in the local state of the $i^{\text{th}}$ process. If $x$ appears in the scope of a universal quantifier, the choice of the process is made by Player 1, otherwise it is made by Player 0. The last round is used to check the valuation of the variables. To this end, the players will inductively choose a subformula to check, until they reach an atomic proposition:

If the subformula is a disjunction $\varphi_1 \vee \varphi_2$, Player 0 chooses either $\varphi_1$ or $\varphi_2$; if it is a conjunction, Player 1 chooses the next subformula. Finally, to verify whether $a(x)$ is satisfied, we check that there is a process with letter $a$ and variable $x$ in its local state. For $x < y$, we check that the process with $x$ in its local state is eventually followed by a distinct process with $y$ in its local state. This check is done during the same round, which guarantees that the positions corresponding to $x$ and $y$ are in the correct order. The number of states needed and the number of rounds are linearly bounded in the length of the formula.

## 4 Conclusion

We extended the verification of round-bounded parameterized systems to a game-based setting, which allows us to model an uncontrollable environment. It would be interesting to consider game-based extensions for the setting from [6], too. Moreover, as games constitute an important approach to verifying branching-time properties (e.g., [26]), our results may be used for branching-time model checking of parameterized systems (using a variant of data logics [20] and a reduction of the model-checking problem to a parameterized pushdown game).

## References

1. P. A. Abdulla, A. Bouajjani, and J. d'Orso. Deciding monotonic games. In *CSL'03*, volume 2803 of *LNCS*, pages 1–14. Springer, 2003.
2. P. A. Abdulla and G. Delzanno. Parameterized verification. *Int. J. Softw. Tools Technol. Transf.*, 18(5):469–473, October 2016.
3. P. A. Abdulla, R. Mayr, A. Sangnier, and J. Sproston. Solving parity games on integer vectors. In *CONCUR'13*, volume 8052, pages 106–120. Springer, 2013.
4. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.
5. M. F. Atig, A. Bouajjani, K. Narayan Kumar, and P. Saivasan. Parity games on bounded phase multi-pushdown systems. In *NETYS'17*, volume 10299 of *LNCS*, pages 272–287, 2017.
6. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011.
7. B. Bérard, S. Haddad, M. Sassolas, and N. Sznajder. Concurrent games on VASS with inhibition. In *CONCUR'12*, volume 7454 of *LNCS*, pages 39–52. Springer, 2012.
8. H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.
9. M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.
10. T. Brázdil, P. Jancar, and A. Kucera. Reachability games on extended vector addition systems with states. In *ICALP'10, Part II*, volume 6199 of *LNCS*, pages 478–489. Springer, 2010.
11. B. Brütsch and W. Thomas. Playing games in the Baire space. In *Proc. Cassting Workshop on Games for the Synthesis of Complex Systems and 3rd Int. Workshop on Synthesis of Complex Parameters*, volume 220 of *EPTCS*, pages 13–25, 2016.

12. J.-B. Courtois and S. Schmitz. Alternating vector addition systems with states. In *MFCS'14*, volume 8634 of *LNCS*, pages 220–231. Springer, 2014.
13. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of FOCS'91*, pages 368–377. IEEE Computer Society, 1991.
14. E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. S.*, 14(4):527–550, 2003.
15. J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Leibniz-Zentrum für Informatik, 2014.
16. D. Figueira and M. Praveen. Playing with repetitions in data words using energy games. In *Proceedings of LICS'18*, pages 404–413. ACM, 2018.
17. J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computability.* Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 2000.
18. S. Jacobs and R. Bloem. Parameterized synthesis. *Log. Methods Comput. Sci.*, 10(1), 2014.
19. P. Jancar. On reachability-related games on vector addition systems with states. In *RP'15*, volume 9328 of *LNCS*, pages 50–62. Springer, 2015.
20. A. Kara. *Logics on data words: Expressivity, satisfiability, model checking.* PhD thesis, Technical University of Dortmund, 2016.
21. D. Kozen. Lower bounds for natural proof systems. In *Proceedings of SFCS'77*, pages 254–266. IEEE Computer Society, 1977.
22. S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.
23. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proceedings of TACAS'08*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.
24. S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV'10*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.
25. S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. Technical Report 2142/15410, University of Illinois, 2010. Available at http://hdl.handle.net/2142/15410.
26. M. Lange and C. Stirling. Model checking games for branching time logics. *J. Log. Comput.*, 12(4):623–639, 2002.
27. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
28. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.
29. A. Seth. Games on multi-stack pushdown systems. In *LFCS'09*, volume 5407 of *LNCS*, pages 395–408. Springer, 2009.
30. L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, MIT, 1974.
31. W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.
32. I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.
33. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1-2):135–183, 1998.