Basic Research in Computer Science

BRICS RS-96-54 I. Walukiewicz: Pushdown Processes: Games and Model Checking

Pushdown Processes: Games and Model Checking

Igor Walukiewicz

BRICS Report Series **RS-96-54**

ISSN 0909-0878 **December 1996**

**Copyright © 1996, BRICS, Department of Computer Science
University of Aarhus. All rights reserved.**

**Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through World Wide Web and anonymous FTP:

`http://www.brics.dk/`
`ftp://ftp.brics.dk/`
This document in subdirectory `RS/96/54/`

Pushdown processes: games and model checking[1]

Igor Walukiewicz
Institute of Informatics
Warsaw University
Banacha 2
02-097 Warsaw, POLAND
e-mail: igw@mimuw.edu.pl

Abstract

Games given by transition graphs of pushdown processes are considered. It is shown that if there is a winning strategy in such a game then there is a winning strategy that is realized by a pushdown process. This fact turns out to be connected with the model checking problem for the pushdown automata and the propositional μ-calculus. It is show that this model checking problem is DEXPTIME-complete.

1 Introduction

Pushdown processes are, at least in this paper, just another name for a pushdown automata. The different name is used to underline the fact that we are mainly interested in the graph of configurations of a pushdown process and not in the language it recognises. This graph can be considered as a transition system. In general such a transition system may not be regular, i.e., may not be an unwinding of a finite transition system. Given a priority function mapping states of the automaton to natural numbers, such a

[1]This work was done at **B**asic **R**esearch **i**n **C**omputer **S**cience, Centre of the Danish National Research Foundation.

[2]This work was partially supported by Polish KBN grant No. 2 P301 009 06

transition system defines a two player parity game. In the game, moves of the players alternate. In a move, a player picks a state reachable by an edge from the current one. The result of a game is a finite or infinite path. The path is finite if one of the players cannot make a move; in this case the other player wins. If the path is infinite we find the smallest priority such that a state of this priority appears infinitely often on the path. Player I wins if this priority is even.

Pushdown processes are a generalisation of regular process which correspond to finite automata or regular transition systems [6]. It is stated in [7] that the extra expressive power of pushdown processes may be of use for describing hierarchically structured systems, such as multi-level cashes, or wide area networks. Considering pushdown games is interesting at least for two reasons. First, as we will show here, there is a connection with model checking for pushdown processes and the μ-calculus. The second reason is the problem of synthesis of correct programs (see for example [14, 21]). The conditions of a game may be seen as a specification, and the two players as the program and environment respectively. In this approach the winning strategy is identified with a reactive program satisfying the specification. Hence it is important to know whether a strategy can be implemented as, for example, a regular or pushdown process.

The decidability of the model checking for pushdown processes and the propositional μ-calculus follows from [17]. An elementary model checking procedure for pushdown processes and alternation free fragment of the calculus was given in [3]. We are not aware of any such elementary decision procedure for the whole μ-calculus. The decidability result mentioned above as well as extensions of it (for example [8, 22]) deal with monadic second order logic and reduce the problem to the decidability of $S2S$ formulas, hence give nonelementary algorithms.

Pushdown processes are a strict generalisation of processes from so called basic process algebra BPA (see [6] for a short survey). The processes from BPA can be considered as pushdown processes with only one state. If language recognition is concerned pushdown automata with one state can recognise the same languages as the general pushdown automata. This is not the case when configuration graph is considered. It was shown in [4] that there exists pushdown automaton which transition graph is not bisimilar to the transition graph of any BPA process. BPA is a subclass of process algebra (PA) [1]. For the other interesting subclass of PA, namely, basic parallel processes, the model checking is undecidable [12]. The question whether

pushdown games have pushdown strategies was posed in [20].
The main results of this paper are the following.

- We show that for every pushdown game G: if there is a winning strategy in G then there is a pushdown winning strategy in G.
- We reduce a model checking problem for the pushdown process and the whole μ-calculus to the model checking problem for finite transition systems and the μ-calculus. The reduction gives a transition system of size $\mathcal{O}((k2^{cmn_1n_2}))$ where m is the number of states of the pushdown process, k the size of its stack alphabet, n_1 is the size of the formula and n_2 is the alternation depth of the original formula. The formula given by reduction is of size n_2 and alternation depth n_2. In particular it turns out that the model checking problem for BPA and a fixed formula is polynomial.
- We show that there exists a formula α such that the model checking problem for a pushdown processes and this particular formula α is DEXPTIME-hard.

Let us mention that the restriction to parity games is not essential. One can use standard methods of translating Muller, Rabin or Streett conditions into parity conditions to obtain appropriate results for this kind of conditions.

The plan of the paper is as follows. We start with a preliminary section where we recall definitions of pushdown automata and the propositional μ-calculus. In the following section we present some facts about games with parity conditions. In the next section we prove that if there is a winning strategy on a pushdown tree then there is one realized by a pushdown automaton. In the last section we consider the model checking problem.

Acknowledgement: I would like to thank Damian Niwinski for his helpful comments.

2 Preliminaries

2.1 Pushdown processes

The set of finite sequences over Σ is denoted Σ^* and the set of finite nonempty sequences over Σ is denoted Σ^+. The empty sequence is denoted by ε. For

$s, s' \in \Sigma^*$ we let ss' denote concatenation of the two sequences.

For a given finite set Σ_s, let $Com(\Sigma_s) = \{skip, pop\} \cup \{push(z) : z \in \Sigma_s\}$ be the set of *stack commands* over Σ_s. The command *skip* does nothing, the meaning of the remaining two commands is standard.

A pushdown automaton (over one letter alphabet) is a tuple:

$$\mathcal{A} = \langle Q, \Sigma_s, q_0 \in Q, \bot \in \Sigma_s, \delta : Q \times \Sigma_s \to \mathcal{P}(Q \times Com(\Sigma_s)) \rangle \qquad (1)$$

where Q is a finite set of *states* and Σ_s a finite *stack alphabet*. State q_0 is the initial state of the automaton and $\bot$ is the initial stack symbol. A *configuration* of an automaton is a pair (s, q) with $s \in \Sigma_s^+$ and $q \in Q$. The initial configuration is $(\bot, q_0)$. We assume that $\bot$ can be neither put nor removed from the stack. We will sometimes write $(s, q) \to (s', q')$ if the automaton in one step can go from the configuration (s, q) to (s', q'). Let $\to^+$, $\to^*$ denote respectively the transitive closure of $\to$ and the reflexive and transitive closure of $\to$.

We will use q to range over states and z to range over letters of the stack alphabet.

Remark: In our definition of a pushdown automaton we have assumed that the automaton can put at most one symbol on the stack in one move. This is done only for convenience of the presentation. The main results also hold for the more general form of automata which can push many symbols on the stack in one move. Please note that we can simulate pushing more symbols on the stack by extending the alphabet and the set of states but the simulating automaton will be in general much bigger. This is because we are interested not in the language recognised by the automaton but in the tree of configurations it generates.

Definition 1 (Pushdown tree) A pushdown automaton $\mathcal{A}$ as in (1) defines a tree $T_{\mathcal{A}} \subseteq (\Sigma_s^+ \times Q)^+$ as follows:

- the root of the tree is $(\bot, q_0)$,
- for every node $(s_0, q_0), \ldots, (s_i, q_i)$, if $(s_i, q_i) \to (s, q)$ then the node has a son $(s_0, q_0), \ldots, (s_i, q_i), (s, q)$.

We call (s_i, q_i) the *label* of the node $(s_0, q_0), \ldots, (s_i, q_i)$.

2.2 Propositional μ-calculus

Let $Prop = \{p_1, p_2, \dots\}$ be a set of *propositional constants* and let $Var = \{X, Y, \dots\}$ be a set of *variables*. Formulas of the μ-calculus over these sets can be defined by the following grammar:

$$F := Prop \mid \neg Prop \mid Var \mid F \vee F \mid F \wedge F \mid \langle\ \rangle F \mid [\ ]F \mid \mu Var.F \mid \nu Var.F$$

Note that we allow negations only before propositional constants. As we will be interested in closed formulas this is not a restriction. In the following, $\alpha, \beta, \dots$ will denote formulas.

Formulas are interpreted in *transition systems* of the form $\mathcal{M} = \langle S, R, \rho \rangle$, where: S is a nonempty set of *states*, $R \subseteq S \times S$ is a binary relation on S and $\rho : Prop \to \mathcal{P}(S)$ is a function assigning to each propositional constant a set of states where this constant holds.

For a given model $\mathcal{M}$ and an *assignment* $V : Var \to \mathcal{P}(S)$, the set of states in which a formula φ is true, denoted $\| \varphi \|_V^{\mathcal{M}}$, is defined inductively as follows:

$$\begin{aligned}
\| p \|_V^{\mathcal{M}} = \rho(p)S \qquad \| \neg p \|_V^{\mathcal{M}} &= S - \rho(p) \\
\| X \|_V^{\mathcal{M}} &= Val(X) \\
\| \alpha \vee \beta \|_V^{\mathcal{M}} &= \| \alpha \|_V^{\mathcal{M}} \cup \| \beta \|_V^{\mathcal{M}} \\
\| \alpha \wedge \beta \|_V^{\mathcal{M}} &= \| \alpha \|_V^{\mathcal{M}} \cap \| \beta \|_V^{\mathcal{M}} \\
\| \langle\ \rangle \alpha \|_V^{\mathcal{M}} &= \{s : \exists s'.R(s, s') \wedge s' \in \| \alpha \|_V^{\mathcal{M}}\} \\
\| [\ ] \alpha \|_V^{\mathcal{M}} &= \{s : \forall s'.R(s, s') \Rightarrow s' \in \| \alpha \|_V^{\mathcal{M}}\} \\
\| \mu X.\alpha(X) \|_V^{\mathcal{M}} &= \bigcap \{S' \subseteq S : \| \alpha \|_{Val[S'/X]}^{\mathcal{M}} \subseteq S'\} \\
\| \nu X.\alpha(X) \|_V^{\mathcal{M}} &= \bigcup \{S' \subseteq S : S' \subseteq \| \alpha \|_{Val[S'/X]}^{\mathcal{M}}\}
\end{aligned}$$

here $V[S'/X]$ is the valuation such that, $V[S'/X](X) = S'$ and $V[S'/X](Y) = V(Y)$ for $Y \neq X$. We shall write $\mathcal{M}, s, V \vDash \varphi$ when $s \in \| \varphi \|_V^{\mathcal{M}}$ and $\mathcal{M}, s \vDash \varphi$ if $\mathcal{M}, s, V \vDash \varphi$ for arbitrary V.

We will use the following well known equivalences. They define the negation of arbitrary closed formula.

$$\begin{aligned}
\neg\langle\ \rangle \alpha &= [\ ]\neg\alpha & \neg[\ ]\alpha &= \langle\ \rangle\neg\alpha \\
\neg\mu X.\alpha(X) &= \nu X.\neg\alpha(\neg X) & \neg\nu X.\alpha(X) &= \mu X.\neg\alpha(\neg X)
\end{aligned} \qquad (2)$$

A model checking problem is to decide whether for a given model $\mathcal{M}$, state s and formula α without free variables, the relation $M, s \models \alpha$ holds. Here we will be interested in the case when M is a pushdown tree and s is the root of it.

3 Parity games and canonical strategies

In this section we recall the notion of parity games. We give an explicit description of winning strategies in *parity games*. We describe the set of winning positions by a fixpoint expression and derive a winning strategy from this expression using the concept of *signatures*. It turns out that this strategy is canonical in some sense.

The notion of signature was proposed by Streett and Emerson [19]. The proof of the existence of memoryless strategies in parity games was given independently by Mostowski [16] and by Emerson and Jutla [10]. Klarlund [13] proves a more general fact that a player has a memoryless winning strategy in a game if he has a winning strategy and his winning conditions are given as a Rabin condition.

Let $G = \langle V = V_I \cup V_{II}, E \subseteq V \times V, \Omega : V \to \{1, \ldots, n\}\rangle$ be a bipartite graph with vertices labelled by *priorities* from $\{1, \ldots, n\}$.

A *game* from some vertex $v_1 \in V_I$ is played as follows: first player I chooses a vertex v_2 with $E(v_1, v_2)$, then player II chooses a vertex v_3 with $E(v_2, v_3)$, and so on ad infinitum unless one of the players cannot make a move. If a player cannot make a move he looses. The result of an infinite play is an infinite path $v_1, v_2, v_3, \ldots$ This path is *winning* for player I if in the sequence $\Omega(v_1), \Omega(v_2), \Omega(v_3), \ldots$ the smallest number appearing infinitely often is even. The play from vertices of V_{II} is defined similarly but this time player II starts.

A *strategy* σ for player I is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex from V_I a vertex $\sigma(\vec{v}) \in V_{II}$, such that, $E(v, \sigma(\vec{v}))$. A strategy is *memoryless* iff $\sigma(\vec{v}) = \sigma(\vec{w})$ whenever $\vec{v}$ and $\vec{w}$ end in the same vertex. A strategy is *winning* iff it guarantees a win for player I whenever he follows the strategy. Similarly we define a strategy for player II.

We will often consider strategies which are partial functions. To fit our definition one can assume that these are total functions which values for some elements don't matter.

Our main goal is the following theorem:

Theorem 2 (Forgetful determinacy)
Let G be a parity game. From every node of G one of the players has a forgetful winning strategy.

The idea of the proof is the following. First we define a set of nodes $\mathcal{W}_I$ of G by a special fixpoint formula. Having this formula, for every vertex in $\mathcal{W}_I$ we associate a *signature* which intuitively says how far is the vertex from something good. We use signatures to define a winning memoryless strategy for player I from vertices in $\mathcal{W}_I$. Finally it turns out that the complement of $\mathcal{W}_I$ is defined by a formula of exactly the same shape as the one defining $\mathcal{W}_I$. This gives us a memoryless winning strategy for player II from vertices not in $\mathcal{W}_I$.

For the rest of this section let us fix a game graph:

$$G = \langle V = V_I \cup V_{II}, E, \Omega : V \to \{1, \ldots, n\} \rangle$$

In particular we assume that the range of Ω is $\{1, \ldots, n\}$ and that n is even. Clearly we can do so without a loss of generality. The graph G can be represented as a Kripke structure $G = \langle V, E, \{I^G, 1^G, \ldots, n^G\} \rangle$, where: V is now considered to be a set of states; E defines an edge relation between states and $\{I, 1, \ldots, n\}$ are propositions. Proposition I^G denotes the set of vertices of player I, i.e., the set V_I. Each proposition $i^G \in \{1^G, \ldots, n^G\}$ denotes the set of nodes with priority i, i.e., the set $\{v : \Omega(v) = i\}$.

Consider the formula:

$$\varphi_I(Z_1, \ldots, Z_n) = \big(I \Rightarrow \bigwedge_{i \in \{1,\ldots,n\}} (i \Rightarrow \langle\ \rangle Z_i)\big) \wedge \big(\neg I \Rightarrow \bigwedge_{i \in \{1,\ldots,n\}} (i \Rightarrow [\ ] Z_i)\big)$$

We will be interested in the set:

$$\mathcal{W}_I = \parallel \mu Z_1 . \nu Z_2 . \ldots . \mu Z_{n-1} . \nu Z_n . \varphi_I(Z_1, \ldots, Z_n) \parallel^G$$

(in the formula μ is used to close variables with odd indices and ν is used for even indices; n is even by our assumption).

Definition 3 When applied to n-tuples of ordinals symbols $=$, $<$, $\leq$ stand for corresponding relations in the lexicographical ordering. For every $i \in \{1, \ldots, n\}$ we use $=_i$ to mean that both arguments are defined and when truncated to first i positions the two vectors are equal; similarly for $<_i$ and $\leq_i$.

Definition 4 (Consistent signature assignment) A signature is a n-tuple of ordinals. An assignment $\mathcal{S}$ of signatures to nodes from some set $U \subseteq V$ will be called *consistent* if for every $v \in U \cap V_I$ there is a vertex $w \in U$ with $E(v,w)$ and such that:

$$\mathcal{S}(w) \leq_{\Omega(v)} \mathcal{S}(v) \text{ and the inequality is strict if } \Omega(v) \text{ is odd.} \tag{3}$$

similarly if $v \in U \cap V_{II}$ then for all vertices w with $E(v,w)$ we have $w \in U$ and the condition (3) holds.

We extend the syntax of the formulas by allowing constructions of the form $\mu^\tau Z.\alpha(Z)$, where τ is an ordinal and $\alpha(Z)$ is a formula from the extended syntax. The semantics is defined as follows:

$$\| \mu^0 Z.\alpha(Z) \|_V^{\mathcal{M}} = \emptyset \qquad \| \mu^{\tau+1} Z.\alpha(Z) \|_V^{\mathcal{M}} = \| \alpha(Z) \|^{\mathcal{M}}_{V[\|\mu^\tau Z.\alpha(Z)\|_V^{\mathcal{M}}/Z]}$$
$$\| \mu^\tau Z.\alpha(Z) \|_V^{\mathcal{M}} = \bigcup_{\rho<\tau} \| \mu^\rho Z.\alpha(Z) \|_V^{\mathcal{M}} \quad (\tau \text{ a limit ordinal})$$

By Knaster-Tarski theorem $\| \mu Z.\alpha(Z) \|_V^{\mathcal{M}} = \bigcup_\tau \| \mu^\tau Z.\alpha(Z) \|_V^{\mathcal{M}}$.

Definition 5 (Canonical signatures) A *canonical signature*, $Sig(v)$, of a vertex $v \in V$ is the smallest in the lexicographical ordering sequence of ordinals $(\tau_1, \ldots, \tau_n)$ such that:

$$v \in \| \varphi_I(P_1, \ldots, P_n) \|^G$$

where:

$$P_i = \mu^{\tau_i} Z_i.\nu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_1, \ldots, P_{i-1}, Z_i, \ldots, Z_n) \quad \text{for } i \text{ odd}$$
$$P_i = \nu Z_i.\mu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_1, \ldots, P_{i-1}, Z_i, \ldots, Z_n) \quad \text{for } i \text{ even}$$

As for an even i the ordinal τ_i is not used, the definition implies that $\tau_i = 0$ for every even i. We prefer to have this redundancy rather than to calculate right indices each time.

Fact 6 A vertex v belongs to $\mathcal{W}_I$ iff the canonical signature, $Sig(v)$, is defined.

Proof
Suppose $v \in \mathcal{W}_I$. Let τ be an ordinal of a cardinality bigger than the cardinality of G. By Knaster-Tarski theorem we have:

$$\mathcal{W}_I = \parallel \mu^\tau Z_1.\nu Z_2 \ldots \mu^\tau Z_{n-1}.\nu Z_n.\varphi_I(Z_1, \ldots, Z_n) \parallel^G$$

Hence $(\tau, \ldots, \tau)$ is an upper bound on the canonical signature for v. So the signature is defined.

Conversely, suppose $Sig(v)$ is defined. For every ordinal ρ and every formula $\alpha(X)$ we have $\parallel \mu^\rho X.\alpha(X) \parallel^G \subseteq \parallel \mu X.\alpha(X) \parallel^G$. Thus $v \in \mathcal{W}_I$ by monotonicity. □

Fact 7 The assignment $v \mapsto Sig(v)$ is a consistent signature assignment.

Proof
We will consider only the case when $v \in V_I$. Let $(\tau_1, \ldots, \tau_n)$ be the canonical signature of v and let $i = \Omega(v)$. Let us assume that i is odd. The case when i is even is simpler.

By our assumptions we have $v \in \parallel \varphi_I(P_1, \ldots, P_n) \parallel^G$. Expanding the definition of φ_I we obtain: $v \in \parallel \langle\ \rangle P_i \parallel^G$. Hence there is a vertex $w \in \parallel P_i \parallel^G$ with $E(v, w)$.

$$P_i = \mu^{\tau_i} Z_i.\nu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_1, \ldots, P_{i-1}, Z_i, \ldots, Z_n)$$

If τ_i is a limit ordinal then, by the definition of μ^τ, there is a successor ordinal $\rho < \tau_i$ such that:

$$w \in \parallel \mu^\rho Z_i.\nu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_1, \ldots, P_{i-1}, Z_i, \ldots, Z_n) \parallel^G$$

Once again referring to the definition of μ^τ we have:

$$w \in \parallel \nu Z_{i+1}.\mu Z_{i+2} \ldots \nu Z_n.\varphi_I(P_1, \ldots, P_i', Z_{i+1}, \ldots, Z_n) \parallel^G$$

where $P_i' = \mu^{\rho-1} Z_i.\nu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_1, \ldots, P_{i-1}, Z_i, \ldots, Z_n)$. This shows that the canonical signature of w cannot be bigger than $(\tau_1, \ldots, \rho - 1)$ on first i positions, for some $\rho \leq \tau_i$. Hence $Sig(w) <_i Sig(v)$. □

Definition 8 (Canonical strategy) A *canonical strategy* is a strategy taking for each node $v \in \mathcal{W}_I \cap V_I$ a son which has the smallest canonical signature.

Remark: Despite the name, canonical strategies may not be uniquely determined because a node may have many sons with the same signature.

The forgetful determinacy theorem follows from the next lemma.

Lemma 9 The canonical strategy is a winning memoryless strategy for player I from every node in $\mathcal{W}_I$. From every node not in $\mathcal{W}_I$ player II has a memoryless winning strategy.

Proof
Suppose $v_0 \in \mathcal{W}_I$. It should be clear that the canonical strategy is memoryless. We show that it is winning. Let $\mathcal{P} = v_0, v_1, \ldots$ be a history of a play when player I uses the canonical strategy. To arrive at a contradiction assume that $\mathcal{P}$ is winning for player II. In other words, that the smallest priority appearing infinitely often on $\mathcal{P}$ is some odd number p.

Take an infinite sequence of positions $j_1 < j_2 < \ldots$ such that: no vertex after v_{j_1} has priority smaller than p, and $\Omega(v_{j_k}) = p$ for $k = 1, \ldots$ From Fact 7 we obtain that $Sig(v_{j_{k+1}}) <_p Sig(v_{j_k})$. This is a contradiction because the lexicographical ordering on sequences of ordinals of bounded length is a well ordering.

To show the second statement of the theorem we use some propositional logic. From equivalences (2), the complement of $\mathcal{W}_I$ is the set:

$$\| \nu Z_1.\mu Z_2. \ldots \nu Z_{n-1}.\mu Z_n.\neg\varphi_I(\neg Z_1, \ldots, \neg Z_n) \|^G$$

Using the propositional tautology

$$\neg\big((p \Rightarrow q) \wedge (\neg p \Rightarrow r)\big) \equiv \big((p \Rightarrow \neg q) \wedge (\neg p \Rightarrow \neg r)\big)$$

we obtain

$$\neg\varphi_I(\neg Z_1, \ldots, \neg Z_n) = \big(I \Rightarrow \bigvee_{i\in\{1,\ldots,n\}} (i \wedge [\,]Z_i)\big) \wedge \big(\neg I \Rightarrow \bigvee_{i\in\{1,\ldots,n\}} (i \wedge \langle\,\rangle Z_i)\big)$$

Using the fact that in each vertex of G exactly one of the propositions $1, \ldots, n$ holds, the formula above is equivalent to:

$$\big(I \Rightarrow \bigwedge_{i\in\{1,\ldots,n\}} (i \Rightarrow [\,]Z_i)\big) \wedge \big(\neg I \Rightarrow \bigwedge_{i\in\{1,\ldots,n\}} (i \Rightarrow \langle\,\rangle Z_i)\big)$$

Consider the game $G' = \langle V = V_{II} \cup V_I, E, \Omega'\rangle$ obtained from G by interchanging the vertices of player I and player II and letting $\Omega'(v) = \Omega(v) + 1$. It

is easy to see that the strategy for player I in G' translates to a strategy for player II in G and vice versa.

In the formulas above let us increase indices of the variables by one. Adding two dummy variables Z_1, Z_{n+2} we can see that in G' the complement of $\mathcal{W}_I$ can be described by the formula:

$$\mu Z_1.\nu Z_2 \ldots .\mu Z_{n+1}.\nu Z_{n+2}.\varphi'_I(Z_1, \ldots, Z_{n+2})$$

where

$$\varphi'_I(Z_1, \ldots, Z_{n+2}) = \\ \big(I \Rightarrow \bigwedge_{i \in \{2,\ldots,n+1\}} (i \Rightarrow \langle\ \rangle Z_i)\big) \wedge \big(\neg I \Rightarrow \bigwedge_{i \in \{2,\ldots,n+1\}} (i \Rightarrow [\ ] Z_i)\big)$$

(observe that the variables Z_1 and Z_{n+2} are not used). By the first statement of the theorem we know that in G' there exists a memoryless winning strategy for player I from every node not in $\mathcal{W}_I$. This strategy translates to a winning memoryless strategy for player II in G. □

Let us finish with a remark that points out an interesting property of canonical signatures. One can show that every strategy induces a consistent signature assignment and vice versa. Hence we can compare strategies by comparing signatures. The next fact implies that the canonical strategy is in some sense an optimal strategy.

Fact 10 The canonical signature assignment $(v \mapsto Sig(v))$ is the least consistent signature assignment. In other words, for every consistent signature assignment $\mathcal{S}$ whenever for some node v, $\mathcal{S}(v)$ is defined then $Sig(v)$ is defined and $Sig(v) \leq \mathcal{S}(v)$.

Proof
Assume conversely that there is a consistent signature assignment $\mathcal{S}$ for which the set of vertices $\{w : \mathcal{S}(w) < Sig(w)\}$ is not empty. Consider vertices from this set for which the difference is at the smallest possible position. Let v be one of such vertices for which $\mathcal{S}(v)$ is the smallest possible up to this position. More precisely v is a vertex such that for some i we have:

1. $\mathcal{S}(v) <_i Sig(v)$,
2. for every w, $\mathcal{S}(w) =_{i-1} Sig(w)$,

3. for every w, if $\mathcal{S}(w) <_i Sig(w)$ then $\mathcal{S}(v) \leq_i \mathcal{S}(w)$.

Given a set of vertices Q we consider the formula:

$$\nu Z_{i+1} \ldots \mu Z_{n-1} . \nu Z_n . \varphi_I(Q, \ldots, Q, Z_{i+1}, \ldots, Z_n)$$

We abbreviate this formula by $\overrightarrow{\sigma Z}.\psi(Q, \vec{Z})$. Observe that, by the definition, i must be odd.

Claim 10.1 Let u be a vertex and Q a set of vertices. If $\mathcal{S}(u)$ is defined and $u \notin \| \overrightarrow{\sigma Z}.\psi(Q, \vec{Z}) \|^G$ then, when player I uses $\mathcal{S}$, player II can force the play into:

$$\neg Q \cap \{w : \mathcal{S}(w) <_i \mathcal{S}(u)\} \tag{4}$$

and the play visits only vertices of priority bigger than i before reaching this set.

Proof: If $u \notin \| \overrightarrow{\sigma Z}.\psi(Q, \vec{Z}) \|^G$ then:

$$u \in \| \mu Z_{i+1} . \nu Z_{i+2} \ldots \nu Z_{n-1} . \mu Z_n . \overline{\varphi}_I(\neg Q, \ldots, \neg Q, Z_{i+1}, \ldots, Z_n) \|^G \tag{5}$$

where

$$\overline{\varphi}_I(Z_1, \ldots, Z_n) = \big(I \Rightarrow \bigwedge_{i \in \{1,\ldots,n\}} (i \Rightarrow [\,] Z_i)\big) \wedge \big(\neg I \Rightarrow \bigwedge_{i \in \{1,\ldots,n\}} (i \Rightarrow \langle\,\rangle Z_i)\big)$$

Let $\overline{Sig}(u)$ denote the signature of the formula in (5) in the node u.

Suppose $\Omega(u) > i$. If $u \in V_I$ then it is the turn of player I. He chooses a successor u' of u with the smallest possible value of $\mathcal{S}$. If $u \in V_{II}$ then we let player II to choose a son u' of u with the smallest possible value of $\overline{Sig}$. By consistency of $\mathcal{S}$, in both cases we know that $\mathcal{S}(u') \leq_{\Omega(u)} \mathcal{S}(u)$ and that $\mathcal{S}(u')$ is strictly smaller if $\Omega(u)$ is odd. It is also easy to check that $\overline{Sig}(u') \leq_{\Omega(u)} \overline{Sig}(u)$ and that $\overline{Sig}(u')$ is strictly smaller if $\Omega(u)$ is even.

We claim that after a finite number of steps as the one above, we must arrive to a vertex of priority not bigger than i. Suppose it is not the case then the above play is infinite. Let $p > i$ be the smallest priority such that states with this priority appeared infinitely often during the play. This priority cannot be odd because, by consistency of $\mathcal{S}$, it would mean that the

prefix of length p of signatures given by $\mathcal{S}$ was decreased infinitely often and increased only finitely often. Similarly it cannot be even because then the prefix of length p of signatures given by $\overline{Sig}$ would be decreased infinitely often and increased only finitely many times. A contradiction.

Hence the play eventually must reach a node w with $\Omega(w) \leq i$. From the way the play was constructed it follows that $\mathcal{S}(w) \leq_i \mathcal{S}(u)$.

If $w \in V_I$ then player I chooses w' with the smallest possible value of $\mathcal{S}$ function. By consistency of $\mathcal{S}$ and the fact that i is odd we know that $\mathcal{S}(w') <_i \mathcal{S}(w) \leq_i \mathcal{S}(u)$. Because w satisfies the formula (5) we get $w' \in \neg Q$.

If $w \in V_{II}$ then by consistency of $\mathcal{S}$ we know that for every w' with $E(w, w')$ we have $\mathcal{S}(w') <_i \mathcal{S}(w)$. Because w satisfies the formula (5) we know that there exists a vertex $w' \in \neg Q$ with $E(w, w')$. $\square$

We proceed with the proof of the fact. Recall that the vertex v was fixed at the beginning of the proof. It is a vertex from $\{u : \mathcal{S}(u) <_i Sig(u)\}$ that has the smallest $\mathcal{S}$-signature up to position i.

We take $Q = \{w : Sig(w) <_i Sig(v) - (0, \ldots, 1, \ldots, 0)\}$, where the last vector has only one nonzero element on position i. We claim that $v \notin \| \overrightarrow{\sigma Z}.\psi(Q, Z) \|^G$ because otherwise v would have smaller signature. By Claim 10.1 we can find a node $w \in \neg Q$ with $\mathcal{S}(w) <_i \mathcal{S}(v)$. By the definition $w \in \neg Q$ means $Sig(w) \geq_i Sig(v) - (0, \ldots, 1, \ldots, 0)$. Hence $\mathcal{S}(w) <_i \mathcal{S}(v) \leq_i Sig(v) - (0, \ldots, 1, \ldots, 0) \leq_i Sig(w)$. A contradiction with the choice of v.

$\square$

4 Existence of pushdown strategies

Let $\mathcal{A}$ be a pushdown automaton as in (1). For simplicity of the presentation let us assume that the set Q of states of $\mathcal{A}$ is partitioned into two sets Q_I and Q_{II}. We also assume that transitions from states in Q_I lead only to states in Q_{II} and vice versa. More formally we require that for every q, q', z, z': whenever $(push(z'), q')$ or (pop, q') is in $\delta(z, q)$ then: $q \in Q_I$ iff $q' \in Q_{II}$.

The automaton $\mathcal{A}$ defines a pushdown tree $T_{\mathcal{A}}$. Together with a priority function $\Omega : Q \to \mathbb{N}$ this defines a parity game.

Definition 11 (Pushdown game) An automaton $\mathcal{A}$ and a priority function Ω define the pushdown game $G_{\mathcal{A}} = \langle V, E, \Omega : V \to \{0, \ldots, n\}\rangle$ where $\langle V, E\rangle$ is a pushdown tree $\mathcal{T}_{\mathcal{A}}$ and $\Omega(v) = \Omega(q)$ for q the state in the label of

v. A partition of V into V_I and V_{II} is defined by the partition of Q: $v \in V_I$ iff the state occurring in the label of v belongs to Q_I.

Remark: From our assumption about partition of the states of $\mathcal{A}$ and assuming that the initial state belongs to Q_{II} we have that in the game $G_{\mathcal{A}}$ player II moves from the vertices on the even levels of $\mathcal{T}_{\mathcal{A}}$ and player I moves from the vertices on odd levels. Observe that, as the game is played on a tree, a strategy can be identified with the subset of the game tree. An important point is what priority assignment functions we allow. We have chosen to allow only functions which are defined in terms of states of the automaton. This choice of the method of assigning priorities is motivated by the fact that we are interested in the winning conditions definable in $S1S$.

Next let us try to make it precise what we mean by a pushdown strategy. Such a strategy should be given by an automaton reading moves of player II and outputting moves for player I. In the infinity, if player II moved according to the rules then the obtained sequence of moves should determine a path of $\mathcal{T}_{\mathcal{A}}$ which is winning for player I.

A *move* is an element of $Q \times Com(\Sigma_s)$, i.e., a pair consisting of a state of $\mathcal{A}$ and a stack command. A path of $\mathcal{T}_{\mathcal{A}}$ *determines* a sequence of moves that the automaton made on this path. Other way around, a sequence of moves may determine a sequence of configurations, i.e., a path of $\mathcal{T}_{\mathcal{A}}$. Some sequences of moves do not determine paths because they contain invalid moves. Let us call *valid*, the sequences determining paths of $\mathcal{T}_{\mathcal{A}}$.

A *strategy automaton* is a deterministic automaton with input and output:

$$\mathcal{B} = \langle Q_{\mathcal{B}}, \Sigma_i, \Sigma_o, \Sigma_{s,\mathcal{B}}, q_0, \bot, \\ \delta_{\mathcal{B}} : Q_{\mathcal{B}} \times \Sigma_s \times (\Sigma_i \times \{\tau\}) \to Q_{\mathcal{B}} \times Com(\Sigma_s) \times (\Sigma_o \times \{\tau\})\rangle \quad (6)$$

where $Q_{\mathcal{B}}$ is a finite set of states; $\Sigma_i, \Sigma_o, \Sigma_{s,\mathcal{B}}$ are finite input, output and stack alphabets respectively. State q_0 is the initial state and $\bot$ is the initial stack symbol. If $\delta_{\mathcal{B}}(q, z, a) = (q', com, b)$ then in the state q with z on the top of the stack and a on the input tape, the automaton changes the state to q', performs the stack command com, and outputs the symbol b. If $a = \tau$ then the automaton does not read the input (and does not move the input head). If $b = \tau$, the automaton outputs nothing.

To be a strategy automaton, $\mathcal{B}$ should have the property that it should output one move of player I after reading one move of player II. Moreover it should output valid moves, i.e.: if $m_1, \ldots, m_k$ is a sequence of read moves, $n_1, \ldots, n_{k-1}$ is a sequence of output moves and $m_1, n_1, \ldots, m_{k-1}, n_{k-1}, m_k$ is

a valid sequence of moves then $\mathcal{B}$ should output some move n_k such that $m_1, n_1, \ldots, m_k, n_k$ is a valid sequence. Finally, in the infinity, if the obtained sequence of moves is valid then it should determine a path of $\mathcal{T}_{\mathcal{A}}$ that is winning for player I (see the definition on page 6).

We say that there is a pushdown strategy in $G_{\mathcal{A}}$ if there is a strategy automaton for $\mathcal{A}$. Our goal in this section is the following theorem.

Theorem 12
If there is a winning strategy for player I in $G_{\mathcal{A}}$ then there is a winning pushdown strategy.

Let us now try to explain an idea of the construction of the strategy. First, we know that if there is a strategy for player I then there is a canonical strategy. This strategy depends only on the current configuration and consists of picking a configuration with the smallest possible signature. Unfortunately, a strategy automaton cannot know the current configuration as there are potentially infinitely many of them. Our strategy automaton, looking at its state and the top of its stack, will be able to tell what is currently on the top of the stack of $\mathcal{A}$ and what is the current state of $\mathcal{A}$. It will also have some finite information about the rest of the stack of $\mathcal{A}$ as we try to describe now.

Let us look at a run of the automaton $\mathcal{A}$. Suppose that in a configuration (s_0, q_0) one of the players performs $(q, push(z))$. Because the game is given by a pushdown automaton, the part of the game from the obtained configuration (sz, q) up to the nearest configurations where z is taken from the stack does not depend on s. What depends on s is the rest of the play when z is taken out from the stack and the current configuration becomes (s, q') for some q'. Hence it should be enough if player I iinstead of knowing the whole s just knew which states he can reach when taking z from the stack. In general he will need a sequence of sets of states $\vec{A} = \{A^p\}_{p=1,\ldots,n}$, each set A^p containing the states that can be reached provided the smallest priority met between pushing and popping z is p.

The definition below formalises this intuition in the notion of subgame $G(\vec{A}, z, \theta, q)$. The additional parameter θ is used to capture the situation when some moves on the current level were already taken. In this case θ would be the smallest priority among states met when z was on the stack.
Notation: We assume that $\{1, \ldots, n\}$ is the range of Ω. We use $\vec{A}$ to range over n element vectors of sets of states and θ to range over $\{1, \ldots, n\}$. We also use z to range over stack symbols and q to range over states.

Definition 13 (Sub-game) For every quadruple $\vec{A}$, z, θ, q we define the game $G(\vec{A}, z, \theta, q)$ as follows. The arena of the game is a subtree of $T_{\mathcal{A}}$ starting from a node with the configuration $(\perp z, q)$. Every node labelled with a configuration $(\perp, q')$, for some q', is marked winning or loosing. We mark the node *winning* if $q' \in A^{\min(p,\theta)}$, where p is the lowest priority of a state appearing on the path to the node (counting q but not q'). Otherwise we mark the node loosing. Whenever a play reaches a marked node then: player I wins if this node is marked winning otherwise player II is the winner. If a play is infinite: player I wins iff the obtained path is winning (as defined at the beginning of Section 3).

Remark: In our definition of the game we did not have the concept of marking but we allowed vertices with no sons, and had the rule that the player looses if he cannot make a move. Hence we can simulate marking of vertices with cutting the paths. We find the metaphor of markings more useful here.

Summarising, player I will have only partial information about the current configuration, namely: the current state, the current symbol on the top of the stack, the sets of states it is allowed to reach when popping this symbol and the lowest priority met from the time when this symbol was pushed on the stack. The size of this information is bounded. To accomplish his task of winning the sub-game he can try to use the canonical signatures.

Definition 14 (Signature, Hint) Suppose that player I has a winning strategy in a game $G(\vec{A}, z, \theta, q)$. Define $Sig(\vec{A}, z, \theta, q)$ to be the canonical signature of the root of the game.

If $q \in Q_I$ then let v be a son of the root having the smallest signature (if there is more than one such son then fix one arbitrary). If v is labelled by $(\perp, q')$ then let $Hint(\vec{A}, z, \theta, q) = (q', pop)$. If v is labelled $(\perp z, q')$ then let $Hint(\vec{A}, z, \theta, q) = (q', skip)$. Otherwise v is labelled by $(\perp z z', q')$ and let $Hint(\vec{A}, z, \theta, q) = (q', push(z'))$.

Finally, when a new push operation is performed, player I should calculate new sets of goal states just using the information he has at hand.

Definition 15 (Update function) Define $Up(\vec{A}, z, \theta, q)$ to be the sequence of sets $\vec{A}_1 = \{A_1^p\}_{p \in \{1,\ldots,n\}}$, where each A_1^p is the set of states q' such that:

$$Sig(\vec{A}, z, \min(\Omega(q), p, \theta), q') \leq_{\min(\Omega(q),p)} Sig(\vec{A}, z, \theta, q)$$

in the case $\min(\Omega(q), p)$ is even and

$$Sig(\vec{A}, z, \min(\Omega(q), p, \theta), q') <_{\min(\Omega(q),p)} Sig(\vec{A}, z, \theta, q)$$

otherwise.

Now we have all the components to define the strategy automaton.

Definition 16 (Strategy automaton) The strategy automaton $\mathcal{B}$ for $G_{\mathcal{A}}$ has the same set Q of states as $\mathcal{A}$. Its input and output alphabets are the moves of $\mathcal{A}$, i.e., $\Sigma_i = \Sigma_o = Q \times Com_s(\Sigma_s)$. Its stack alphabet $\Sigma_{s,\mathcal{B}}$ is $\mathcal{P}(Q)^n \times \Sigma_s \times \{1, \ldots, n\}$. Before defining the transition relation $\delta_{\mathcal{B}}$ let us introduce an abbreviation. We introduce a new stack command $repmin(\theta')$ which means: if on the top of the stack there is some triple $\vec{A}z\theta$, replace it with $\vec{A}z\theta_1$, where $\theta_1 = \min(\theta, \theta')$. We also introduce a semicolon operation, so $\delta_{\mathcal{B}}(q, \vec{A}z\theta, a) = (q', pop; repmin(\theta'))$ means that first $\vec{A}z\theta$ is removed from the stack, then possibly the third component of the triple currently at the top of the stack is changed and the new state becomes q'. Hence, if we had a configuration $(s\vec{A}_1 z_1 \theta_1 \vec{A} z \theta, q)$ then after this operation we obtain the configuration $(s\vec{A}_1 z_1 \min(\theta_1, \theta'), q')$. Let us proceed with the definition of $\delta_{\mathcal{B}}$:

- If $q \in Q_I$ then:
  - $\delta_{\mathcal{B}}(q, \vec{A}z\theta, \tau) = (q', skip, \text{“}(q', skip)\text{”})$ if $Hint(\vec{A}, z, q, \theta) = (q', skip)$.
  - $\delta_{\mathcal{B}}(q, \vec{A}z\theta, \tau) = (q', pop; repmin(\min(\theta, \Omega(q))), \text{“}(q', pop)\text{”})$ if $Hint(\vec{A}, z, q, \theta) = (q', pop)$.
  - $\delta_{\mathcal{B}}(\vec{A}z\theta, q, \tau) = (q', repmin(\Omega(q)); push(\vec{A}'z'n), \text{“}(q', push(z'))\text{”})$ if $Hint(\vec{A}, z, q, \theta) = (q', push(z'))$ and $\vec{A}' = Up(\vec{A}, z, \theta, q)$
- If $q \in Q_{II}$ then:
  - $\delta_{\mathcal{B}}(q, \vec{A}z\theta, \text{“}(q', skip)\text{”}) = (q', skip, \tau)$ if $(q', skip) \in \delta_{\mathcal{A}}(q, z)$
  - $\delta_{\mathcal{B}}(q, \vec{A}z\theta, \text{“}(q', pop)\text{”}) = (q', pop; repmin(\min(\theta, \Omega(q))), \tau)$ if $(q', pop) \in \delta_{\mathcal{A}}(q, z)$.
  - $\delta_{\mathcal{B}}(q, \vec{A}z\theta, \text{“}(q', push(z'))\text{”}) = (q', repmin(\Omega(q)); push(\vec{A}'z'n), \tau)$ if $(q', push(z')) \in \delta_{\mathcal{A}}(q, z)$ and $\vec{A}' = Up(\vec{A}, z, \theta, q)$.

Lemma 17 Suppose that player I can win in $G(\vec{A}, z, \theta, q)$. If

$$\delta_{\mathcal{B}}(q, \vec{A}z\theta, \tau) = (q_1, repmin(\Omega(q)); push(\vec{A}_1 z_1' n), \text{“}(push(z'), q)\text{”})$$

or

$$\delta_{\mathcal{B}}(q, \vec{A}z\theta, \text{“}(q_1, push(z_1))\text{”}) = (q_1, repmin(\Omega(q)); push(\vec{A}_1 z_1 n), \tau)$$

then $Sig(\vec{A}_1, z_1, n, q_1) \leq_{\Omega(q)} Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\Omega(q)$ is odd.

Proof
Consider the games $G = G(\vec{A}, z, \theta, q)$ and $G_1 = G(\vec{A}_1, z_1, n, q_1)$. Define a function $\mathcal{F} : G_1 \to G$ by $\mathcal{F}((\perp s', q') \ldots (\perp s'', q'')) = (\perp z s', q') \ldots (\perp z s'', q'')$, i.e. to every configuration of the path we add z just after $\perp$. It is an injective function respecting descendancy relation and priorities of nodes. This function assigns to the root of G_1 the node v labelled $(\perp z z_1, q_1)$. This node is a son of the root of G.

Let σ denote the canonical strategy in G. Because $v \in \sigma$ we can use the function $\mathcal{F}$ to obtain a strategy $\sigma_1 = \mathcal{F}^{-1}(\sigma)$ in G_1. We will show that this strategy is winning.

Let $\mathcal{P}$ be a result of a play in the game G_1 when player I uses σ_1. If $\mathcal{P}$ is infinite then $\mathcal{F}(\mathcal{P})$ is a result of a play in G when player I uses σ. Hence $\mathcal{P}$ is winning for I. Suppose $\mathcal{P}$ is finite ending in some node w. The label of w is $(\perp, q')$ for some state q'. We will show that this node is marked winning. Let p be the minimum of priorities of states appearing on the path from v to $\mathcal{F}(w)$. According to Definition 13 the node is winning if $q' \in \vec{A}_1^p$. By definition of the automaton $\mathcal{B}$ we know that $\vec{A}_1 = Up(\vec{A}, z, \theta, q)$. Hence we have to show that:

$$Sig(\vec{A}, z, \min(\Omega(q), p, \theta), q') \leq_{\min(\Omega(q),p)} Sig(\vec{A}, z, \theta, q)$$

Let us use the subscript G in $Sig_G(v)$ to stress that this is the canonical signature in the game G.

Claim 17.1 $Sig_G(\mathcal{F}(w)) \leq_{\min(\Omega(q),p)} Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\min(\Omega(q), p)$ is odd.

Proof: As v is a son of the root of G on the path to $F(w)$, by consistency of canonical signatures (Fact 7) $Sig_G(\mathcal{F}(w)) \leq_p Sig_G(v)$ and it is strictly

smaller if p is odd. By the same fact we have $Sig_G(v) \leq_{\Omega(q)} Sig(\vec{A}, z, \theta, q)$ and is strictly smaller if $\Omega(q)$ is odd. $\Box$

Claim 17.2 $Sig_G(\mathcal{F}(w)) = Sig_G(\vec{A}, z, \min(\Omega(q), p, \theta), q')$

Proof: We show that the game $G(\vec{A}, z, \min(\Omega(q), p, \theta), q')$ is isomorphic to the part of G issued from $\mathcal{F}(w)$. To see this, we have to check that a node is marked winning in $G(\vec{A}, z, \min(\Omega(q), p, \theta), q')$ iff it is marked winning in G. Let q'' be a state and u a node labelled by $(\perp, q'')$. Let also p'' be the minimum of priorities of states that appeared between $\mathcal{F}(w)$ and u. The node u is marked winning in G iff $q'' \in A^{\min(\min(\Omega(q),p,p''),\theta)}$. It is marked winning in $G(\vec{A}, z, \min(\Omega(q), p, \theta), q')$ iff $q'' \in A^{\min(p'', \min(\Omega(q),p,\theta))}$. $\Box$

Knowing that σ_1 is winning in G_1 we can define a signature assignment by $\mathcal{S}(w) = Sig(\mathcal{F}(w))$ for every w reachable in a play of G_1 when player I uses σ_1. This is a consistent signature assignment, hence by Fact 10 we have that $Sig(\vec{A}_1, z_1, \theta_1, q_1) \leq \mathcal{S}(\mathcal{F}^{-1}(v))$. By consistency of $\mathcal{S}$ we have that $\mathcal{S}(\mathcal{F}^{-1}(v)) \leq_{\Omega(q)} Sig_G(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\Omega(q)$ is odd. $\Box$

Lemma 18 If a configuration $(sz\vec{A}\theta, q)$ is reachable from the initial configuration then $Sig(\vec{A}, z, \theta, q)$ is defined.

Proof
Induction on the length of the derivation with a help of Lemma 17. $\Box$

Lemma 19 Suppose that player I can win the game $G_{\mathcal{A}}$. Let $(sz\vec{A}\theta, q)$ be a configuration of $\mathcal{B}$ reachable from the initial one. Suppose also that on some finite input sequence w the automaton $\mathcal{B}$ goes from $(sz\vec{A}\theta, q)$ to a configuration $(sz\vec{A}\theta', q')$ and $sz\vec{A}$ is always on the stack during this derivation. Let p be the minimum of the priorities of the states appearing in the derivation. We have:

1. $\theta' = \min(p, \theta)$
2. $Sig(\vec{A}, z, \theta', q') \leq_p Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if p is odd (in particular both signatures are defined).

Proof
The proof proceeds by induction the length of derivation. The case $(s\vec{A}z\theta, q) \rightarrow^a (s\vec{A}z\theta, q')$ follows directly from the construction of the automaton.

Suppose:

$$(s\vec{A}z\theta, q) \to^a (s\vec{A}z\theta''\vec{A}_1 z_1 n, q_1) \to^u (s\vec{A}z\theta''\vec{A}_1 z_1 \theta_1', q_1') \to^b (s\vec{A}z\theta', q')$$

and $\vec{A}_1 z_1$ was not popped while reading the sequence u. By induction assumption, θ_1' is the minimum of priorities of states which appeared in the part of the derivation when there was $s\vec{A}z\theta''\vec{A}_1 z_1$ on the stack. Let $p_1 = \min(\theta_1', \Omega(q_1'))$. From Lemma 18 we know that the signature $Sig(\vec{A}_1, z_1, \theta_1', q_1')$ is defined. Hence $q' \in A_1^{p_1}$. This, by definition, means:

$$Sig(\vec{A}, z, \min(\Omega(q), p, \theta), q') \leq_{\min(\Omega(q),p)} Sig(\vec{A}, z, \theta, q)$$

and the inequality is strict if $\min(\Omega(q), p)$ is odd. It is easy to see that $\theta' = \min(\Omega(q), p, \theta)$.

The remaining case is when $(s\vec{A}z\theta, q) \to^u (s\vec{A}z\theta_1, q_1) \to^v (sAz\theta', q')$. This follows directly from two applications of the induction assumption. □

Lemma 20 $\mathcal{B}$ is a strategy automaton.

Proof

The automaton $\mathcal{B}$ is constructed in such a way that from the current configuration of $\mathcal{B}$ it is easy to *extract* the current configuration of $\mathcal{A}$; it is enough to throw away $\vec{A}$ and θ components from the stack. We have also the property that whenever $\mathcal{B}$ read $m_1, \ldots, m_i$, outputed $n_1, \ldots, n_i$ and $m_1, n_i, \ldots, m_i$ determines a path in $\mathcal{T}_\mathcal{A}$ then $m_1, n_1, \ldots, m_i, n_i$ also determines a path in $\mathcal{T}_\mathcal{A}$. Moreover this path ends in a configuration of $\mathcal{A}$ which is extracted from the current configuration of $\mathcal{B}$.

Now, assume conversely that $\mathcal{B}$ is not a strategy automaton. Let $w_i = m_1, m_2, \ldots$ be an input word on which $\mathcal{B}$ outputs $w_o = n_1, n_2, \ldots$ and the sequence $m_1, n_1, \ldots$ determines a loosing (for player I) path in $\mathcal{T}_\mathcal{A}$.

Suppose that $\mathcal{P}$ is finite, i.e., $\mathcal{B}$ cannot make a move from some configuration. Say it is $(sz\vec{A}\theta, q)$. If $q \in Q_{II}$ then, by the definition of $\mathcal{B}$, it means that the next move in the input sequence is invalid. Hence $q \in Q_I$. From Lemma 18 it follows that $Sig(\vec{A}, z, \theta, q)$ is defined. Hence $Hint(\vec{A}, z, \theta, q)$ is defined and $\mathcal{B}$ can make a move. A contradiction.

Suppose that $\mathcal{P}$ is infinite. This means that the smallest priority of a state appearing i.o. on the path determined by $m_1, n_1, \ldots$ is odd. Call it p. From what was said in the first paragraph, this means that p is the smallest priority of a state appearing i.o. in the run of $\mathcal{B}$ on w_i. Using these observations we

will construct a sequence of strictly decreasing signatures. This will be a contradiction with the fact that the signatures are well ordered.

Let x_0 be a position in the run such that: (i) after x_0 no state with a priority smaller than p appears on the run, (ii) on the position x_0 in the run there is a configuration $(sz\vec{A}\theta, q)$ and for every position after x_0 we have $sz\vec{A}$ on the stack.

Suppose there is a position x_1 after x_0 with a configuration $(sz\vec{A}\theta_1, q_1)$ and a state of the priority p occurs in a configuration between x_0 and x_1. By Lemma 19 we have that $Sig(\vec{A}, z, \theta_1, q_1) <_p Sig(\vec{A}, z, \theta, q)$. Next from x_1 we can look for a position x_2 with a configuration $(sz\vec{A}\theta_2, q_2)$ such that a state of the priority p appears between x_1 and x_2. This way we construct a sequence of positions $x_0, x_1, \ldots, x_i$. Because the signatures decrease, this sequence must be finite. Hence form some position, say x_i, we will not be able find a bigger position with the required properties. As a state of priority p appears infinitely often on the run, there must be a position x_{i+1} after x_i with a configuration $(sz\vec{A}\theta_i z'\vec{A}'n, q_{i+1})$ such that $(sz\vec{A}\theta_i z'\vec{A}')$ is on the stack of every configuration after x_{i+1}. By Lemmas 17 and 19 we have $Sig(\vec{A}', z', n, q_{i+1}) \leq_p Sig(\vec{A}, z, \theta_i, q_i)$ and the inequality is strict if a state of priority p appeared between x_i and x_{i+1}. From x_{i+1} we can repeat exactly the same construction as from x_0. Repeating this reasoning infinitely often we obtain an infinite sequence of strictly decreasing signatures. A contradiction. □

Remark: The automaton $\mathcal{B}$ is exponentially larger than $\mathcal{A}$. One can show that in general the strategy automaton must be exponentially larger, although it is not clear that the exponent must be $\mathcal{O}(n|Q|)$ as it is in the case of $\mathcal{B}$. This situation is different from the situation for parity games on finite transition systems where no memory is need.

An example of a game that has only big strategies is the following. Player *II* starts by choosing a sequence of n symbols: 0 or 1. Then player *II* chooses a position $i \in \{1, \ldots, n\}$ and asks what symbol stands on this position. Player *I* has to answer correctly. Then Player *II* asks about another position and Player *I* wins if he answers correctly also this time. It is easy to see that the graph of such a game can be defined by a pushdown automaton of size $\mathcal{O}(n)$. Every strategy automaton must have the size $\mathcal{O}(2^n)$.

5 Model checking for pushdown trees

We consider a problem of checking whether a given pushdown tree satisfies a given formula of the propositional μ-calculus. We will reduce this problem to the problem of establishing existence of a winning strategy in a game on a pushdown tree. Next we will use results from the previous section to show how one can solve this later problem. Finally we will show the lower bound on the complexity of the model checking problem

5.1 Reduction to games

We will show how to reduce a model checking problem to a problem of establishing existence of a winning strategy in a game on a pushdown tree. Let us start with some technical definitions concerning μ-calculus formulas. These will facilitate the description of the reduction.

Definition 21 (Binding) We call a formula *well named* if every variable is bound at most once in the formula and free variables are distinct from bound variables. For a variable X bound in a well named formula φ there exists a unique subterm of φ of the form $\mu X.\beta(X)$ or $\nu X.\beta(X)$, from now on called the *binding definition of* X *in* φ and denoted $\mathcal{D}_\varphi(X)$. We call X a *μ-variable* when $\mathcal{D}_\varphi(X) = \mu X.\beta(X)$ for some β, otherwise we call X a *ν-variable*.

The function $\mathcal{D}_\varphi$ assigning to every bound variable its *binding definition* in φ will be called the *binding function* associated with φ.

Definition 22 (Dependency order) Given a formula φ we define the *dependency order* over the bound variables of φ, denoted $\leq_\varphi$, as the least partial order relation such that if X occurs free in $\mathcal{D}_\varphi(Y)$ then $X \leq_\varphi Y$. We will say that a bound variable Y depends on a bound variable X in φ when $X \leq_\varphi Y$.

Definition 23 (Fisher-Ladner closure) For a given formula α we denote by $FL(\alpha)$ (Fisher-Ladner closure) the set of subformulas of α (including α itself).

Let φ be a μ-calculus formula without free variables. Without a loss of generality we may assume that it is well named. Let $X_1, \ldots, X_n$ be some linearisation of the dependency order $\leq_\varphi$, i.e., if $X_i \leq_\varphi X_j$ then $i \leq j$. We will assume that the variables with even indices are ν-variables and the variables with odd indices are μ-variables. If it is not the case, we can add

dummy variables to the list. This assumption is not essential but simplifies the presentation as the index immediately determines whether it is a μ or a ν-variable.

Let:

$$\mathcal{A} = \langle Q, \Sigma_s, q_0 \in Q, \bot \in \Sigma_s, \delta : \Sigma_s \times Q \to Com(\Sigma_s) \times Q \rangle$$

be a pushdown automaton as in (1).

In the previous section we have assumed that the states of the automaton defining a game are partitioned into Q_I and Q_{II} and transitions from states in one set lead to states in the other set. Here we will still assume the that the set of states is partitioned but it may now happen that a transition leads to a state from the same set. We can avoid this by adding some dummy states. The number of added states will be at most linear in the size of the automaton.

Now we define our target pushdown game. Consider the automaton:

$$\mathcal{C} = \langle Q \times FL(\varphi), \Sigma_s, q_0, \bot, \delta_{\mathcal{C}} \rangle$$

where $\delta_{\mathcal{C}}$ is defined as follows:

$$\begin{aligned}
\delta_{\mathcal{C}}((q, \alpha \vee \beta), z) =& \{((q, \alpha), skip), ((q, \beta), skip)\} \\
\delta_{\mathcal{C}}((q, \alpha \wedge \beta), z) =& \{((q, \alpha), skip), ((q, \beta), skip)\} \\
\delta_{\mathcal{C}}((q, \mu X.\alpha(X)), z) =& \delta_{\mathcal{C}}((q, \nu X.\alpha(X)), z) = \{((q, X), skip)\} \\
\delta_{\mathcal{C}}((q, X), z) =& \{((q, \alpha(X)), skip)\} \quad \text{if } \mathcal{D}_{\varphi}(X) = \sigma X.\alpha(X) \\
((q', \alpha), push(z')) \in& \delta_{\mathcal{C}}((q, \langle\ \rangle \alpha), z) = \delta_{\mathcal{C}}(z, (q, [\ ]\alpha)) \\
& \text{if } \delta(q, z) = (q', push(z')) \\
((q', \alpha), push(z')) \in& \delta_{\mathcal{C}}((q, \langle\ \rangle \alpha), z) = \delta_{\mathcal{C}}((q, [\ ]\alpha), z) \\
& \text{if } \delta(q, z) = (q', pop)
\end{aligned}$$

It remains to define in which nodes player I is to move and what is the priority of each state. Player I moves when the game is in a node which label contains a state: $(q, \alpha \vee \beta)$, $(q, \mu X.\alpha(X))$, $(q, \nu X.\alpha(X))$, (q, X), or $(q, \langle\ \rangle \alpha)$; for some $q \in Q$ and some formulas α, β. In the remaining nodes player II is to move. Priority function Ω is defined by: $\Omega((q, X_i)) = i$ and $\Omega((q, \alpha)) = n+1$, for α not a variable and $q \in Q$.

Theorem 24
$\mathcal{T}_{\mathcal{A}} \models \varphi$ iff there is a winning strategy for player I in the game $\mathcal{T}_{\mathcal{C}}$ with the priority function Ω.

For finite transition systems a very similar theorem was shown by Emerson, Jutla and Sistla in [9]. For left to right implication one can use signatures of φ. For right to left implication assume conversely and show that player II has a winning strategy. See for example [19] or [18] for similar arguments.

5.2 Establishing existence of winning strategies

Let $\mathcal{A}$ be a pushdown automaton as in (1) and let $\Omega : Q \to \{1, \ldots, n\}$ be an indexing function. These define the game on $G_{\mathcal{A}}$. Here we are concerned with the problem: given $\mathcal{A}$ and Ω establish whether there exists a winning strategy for player I in $G_{\mathcal{A}}$. We will reduce this problem to the problem of establishing existence of a winning strategy in a game on some finite graph. Let $\mathcal{A}$ and Ω be fixed for the rest of this subsection.

Definition 25 (Game $\mathcal{M}_{\mathcal{A}}$) Let $\mathcal{M}_{\mathcal{A}} = \langle V_{\mathcal{A}}, \to, \Omega_{\mathcal{A}} \rangle$ be a game on a finite graph defined as follows. For every $\vec{A}, \vec{A}_1,\ z, z_1,\ \theta,\ q, q_1$ and $p \in \{1, \ldots, n\}$ we have nodes:

$$\begin{array}{ll} Check(\vec{A}, z, \theta, p, q) & Push(\vec{A}, z, \theta, q) \\ Move((\vec{A}, z, \theta, q), (?, z_1, q_1)) & Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1)) \\ Pop(q) & Err(q) \end{array}$$

Here '?' is a special symbol. We have the following transitions between the nodes:

$$\begin{array}{l} Check(\vec{A}, z, \theta, p, q) \to Check(\vec{A}, z, \theta, p, q') \quad \text{if } (q', skip) \in \delta(q, z) \\ Check(\vec{A}, z, \theta, p, q) \to Pop(q') \quad \text{if } (q', pop) \in \delta(q, z) \text{ and } q' \in A^{\min(\Omega(q), \theta)} \\ Check(\vec{A}, z, \theta, p, q) \to Err(q') \quad \text{if } (q', pop) \in \delta(q, z) \text{ and } q' \notin A^{\min(\Omega(q), \theta)} \\ Check(\vec{A}, z, \theta, p, q) \to Move((\vec{A}, z, \theta, q), (?, z_1, q_1)) \\ \qquad\qquad \text{if } (q_1, push(z_1)) \in \delta(q, z) \end{array}$$

and exactly the same transitions from $Push(\vec{A}, z, \theta, q)$, moreover we have:

$$\begin{array}{l} Move((\vec{A}, z, \theta, q), (?, z_1, q_1)) \to Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1)) \\ Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1)) \to Push(\vec{A}_1, z_1, n, q_1) \\ Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1)) \to Check(\vec{A}, z, \min(\theta, p), p, q'') \\ \qquad\qquad \text{if } p \leq \Omega(q) \text{ and } q'' \in A_1^p \end{array}$$

The set V_I of nodes where Player I makes a move consists of nodes:

$$Check(\vec{A}, z, \theta, p, q), \quad Push(\vec{A}, z, \theta, q), \quad Move((\vec{A}, z_1, \theta, q_1), (?, z_2, q_2))$$
for $q \in Q_I$ and arbitrary $\vec{A}, z, z_1, z_2, \theta, p, q_1, q_2$

The set V_{II} of nodes where Player II makes a move consists of nodes:

$$Check(\vec{A}, z, \theta, p, q), \quad Push(\vec{A}, z, \theta, q), \quad Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$$
for $q \in Q_{II}$ and arbitrary $\vec{A}, z, z_1, z_2, \theta, p, q_1, q_2$

Priority function Ω_M is defined by:

$$\Omega_M(Check(\vec{A}, z, \theta, p, q)) = p \qquad \Omega_M(Push(\vec{A}, z, \theta, q)) = \Omega(q)$$
$$\Omega_M(m) = n + 1 \quad \text{for all other nodes } m \text{ of } \mathcal{M}_{\mathcal{A}}$$

Player I wins in the game $\mathcal{M}_{\mathcal{A}}$ if either:

- after finitely many steps player II cannot make a move or a node labelled $Pop(q)$, for some q, is reached.
- the game is infinite and the infinite path $\mathcal{P}$ which is the result of the play is winning for I. Recall that this means that the minimal priority of states appearing infinitely often on $\mathcal{P}$ is even.

Theorem 26
Player I has a winning strategy in the game $G_{\mathcal{A}}$ iff he has a winning strategy in the game $\mathcal{M}_{\mathcal{A}}$ from the node $Check((\emptyset, \ldots, \emptyset), \bot, n, n, q_0)$.

Proof
First, let us consider the left to right implication. We define a strategy σ_M for player I on $\mathcal{M}_{\mathcal{A}}$ as follows.

- if $q \in Q_I$, $Sig(\vec{A}, z, \theta, q)$ is defined and $Hint(\vec{A}, z, \theta, q) = (q_1, skip)$ then:

$$\sigma_M(Check(\vec{A}, z, \theta, p, q)) = Check(\vec{A}, z, \theta, p, q_1)$$
$$\sigma_M(Push(\vec{A}, z, \theta, q)) = Push(\vec{A}, z, \theta, q_1)$$

- if $q \in Q_I$, $Sig(\vec{A}, z, \theta, q)$ is defined and $Hint(\vec{A}, z, \theta, q) = (q_1, pop)$ then:

$$\sigma_M(Check(\vec{A}, z, \theta, p, q)) = \sigma_M(Push(\vec{A}, z, \theta, q)) = Pop(q_1)$$

- If $q \in Q_I$, $Sig(\vec{A}, z, \theta, q)$ is defined and $Hint(\vec{A}, z, \theta, q) = (q_1, push(z_1))$ then:

$$\sigma_M(Check(\vec{A}, z, \theta, p, q)) = \sigma_M(Push(\vec{A}, z, \theta, q)) = Move((\vec{A}, z, \theta, q), (?, z_1, q_1))$$

- if $q \in Q_I \cup Q_{II}$, $Sig(\vec{A}, z, \theta, q)$ is defined and $\vec{A}_1 = Up(\vec{A}, z, \theta, q)$ then let:

$$\sigma_M(Move((\vec{A}, z, \theta, q), (?, z_1, q_1))) = Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$$

Let $\rightarrow_{\sigma_M}$ denote the subset of the transition relation of $\mathcal{M}_\mathcal{A}$ defined by the strategy σ_M, i.e., $\rightarrow_{\sigma_M} = \{(m, n) : \text{if } m \in V_I \text{ then } \sigma_M(m) = n\} \cap \rightarrow$. We will show that every path along $\rightarrow_{\sigma_M}$ is winning for player I.

The initial state of $\mathcal{M}_\mathcal{A}$ is $Check((\emptyset, \ldots, \emptyset), \bot, n, n, q_0)$. From the assumption that player I can win in the game $G_\mathcal{A}$ it follows that $Sig((\emptyset, \ldots, \emptyset), \bot, n, q_0)$ is defined. Let us observe the following properties:

- There is always $Pop(q_1)$ as required in the first clause of the definition of $\rightarrow_{\sigma_M}$.
- There is no $\rightarrow_{\sigma_M}$ transition to $Err(q_1)$, for any q_1.
- If $Check(\vec{A}, z, \theta, p, q) \rightarrow_{\sigma_M} Check(\vec{A}, z, \theta, q_1)$ then $Sig(\vec{A}_1, z_1, n, q_1) \leq_{\Omega(q)} Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\Omega(q)$ is odd. Similarly for $Push$ instead of $Check$.
- If $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1)) \rightarrow_{\sigma_M} Push(\vec{A}_1, z_1, n, q_1)$ then $Sig(\vec{A}_1, z_1, n, q_1) \leq_{\Omega(q)} Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\Omega(q)$ is odd.
- If $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1)) \rightarrow_{\sigma_M} Check(\vec{A}, z, \theta_2, p, q_2)$ then $Sig(\vec{A}, z, \theta_2, q_2) \leq_p Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if p is odd.

A standard argument about the signatures shows that every path in $\mathcal{M}_\mathcal{A}$ starting from the initial vertex and proceeding along $\rightarrow_{\sigma_M}$ transitions is winning for player I.

For the right to left implication assume that there is a winning strategy σ_M in $\mathcal{M}_\mathcal{A}$. We construct a strategy automaton $\mathcal{C}$:

$$\mathcal{C} = \langle Q_C, Q \times Com(\Sigma), Q \times Com(\Sigma), S_M, q_0, Check((\emptyset, \ldots, \emptyset), \bot, n, n, q_0), \delta_C \rangle$$

Where $Q_{\mathcal{C}}$ is some set of auxiliary states needed to "implement" the necessary behaviour of δ_C that we describe below. The automaton $\mathcal{C}$ will work in macro steps. Each step will begin and end in the state $q_0 \in Q_{\mathcal{C}}$. It will be also the case that at the beginning and end of each macro step there will be *Check* or *Push* node on the top of the stack.

Assume that m is the current symbol at the top of the stack. Assume also that m is of the form $Check(\vec{A}, z, \theta, p, q)$ or $Push(\vec{A}, z, \theta, q)$.

If $q \in Q_I$ then for every transition $m \rightarrow_{\sigma_M} u$ we add to $\delta(q_0, m, \tau)$ the following transitions:

- If u is $Check(\vec{A}, z, \theta, p, q')$ or $Push(\vec{A}, z, \theta, q')$ then replace m by u on the top of the stack and output "$(q', skip)$".
- If u is $Pop(q')$ then $q' \in A^{\min(\Omega(q),\theta)}$. Pop elements from the stack until a *Push* node is popped. The current top node of the stack becomes some $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$. Push the node $Check(\vec{A}, z, \min(\theta, p), p, q')$ where $p = \min(\Omega(q'), \theta_1, \Omega(q))$. Output "$(q', pop)$".
- If u is a $Move((\vec{A}, z, \theta, q), (?, z_1, q_1))$ node then there are nodes u', u'' such that $u \rightarrow_{\sigma_M} u' \rightarrow_{\sigma_M} u''$. Moreover u' is $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$ and u'' is $Push(\vec{A}_1, z_1, n, q_1)$. Add to $\delta(q_0, m, \tau)$ operations that push u' and u'' on the stack and output "$(q_1, push(z_1))$"

If $q \in Q_{II}$ then for every transition $m \rightarrow_{\sigma_M} n$ we add the following transitions:

- If u is $Check(\vec{A}, z, \theta, p, q')$ or $Push(\vec{A}, z, \theta, q')$ then on input "$(q', skip)$" replace m by u on the top of the stack. Do not produce any output.
- If u is $Pop(q')$ then $q' \in A^{\min(\Omega(q),\theta)}$. Start from $\delta(q_0, m, \text{"}(q', Pop)\text{"})$ an operation which pops elements from the stack until a *Push* node is popped. The current top node of the stack becomes some node $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$. Push the node $Check(\vec{A}, z, \min(\theta, p), p, q')$ where $p = \min(\Omega(q'), \theta_1, \Omega(q))$. Do not produce any output.
- If u is a $Move((\vec{A}, z, \theta, q), (?, z_1, q_1))$ node then there are nodes u', u'' such that $u \rightarrow_{\sigma_M} u' \rightarrow_{\sigma_M} u''$, where u' is $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$ and u'' is $Push(\vec{A}_1, z_1, n, q_1)$. Start from $\delta(q_0, m, \text{"}(q_1, Push(z_1))\text{"})$ an operation that pushes u' and u'' on the stack. Do not produce any output.

After the end of the macro step we arrive back at the configuration when the state is q_0 and the node on the top of the stack is either *Push* or *Check* node.

To see that $\mathcal{C}$ is a strategy automaton, one uses the following observation.

Observation 26.1 Let $m_1, \ldots, m_i$ be a sequence of moves of $\mathcal{A}$. Suppose the automaton $\mathcal{C}$ on this sequence goes from a configuration (sm, q_0) to a configuration $(s'm', q_0)$ and outputs $n_1, \ldots, n_i$ in the process. If p is the smallest priority of a state appearing in $m_1, n_1, \ldots, m_i, n_i$ then the signature of m' is not bigger on positions up to p than the signature of m; the first signature is strictly smaller if p is odd.

□

The size of the transition system $\mathcal{M}$ is $\mathcal{O}(k2^{cmn})$ where k is the size of the stack alphabet, m is the number of states of $\mathcal{A}$, n is the range of the priority function Ω and c is a constant. The task of establishing existence of a winning strategy in $\mathcal{M}$ is equivalent to checking satisfiability of the specific μ-calculus formula. Hence any model checking algorithm will solve the problem. Using currently known algorithms [11, 15] we obtain that the whole problem can be solved in time $\mathcal{O}((k2^{cmn})^n)$. This is the estimation only for the problem of establishing existence of a winning strategy. Putting it together with the reduction from the previous subsection we obtain:

Corollary 27 For a given automaton $\mathcal{A}$ with m states and k stack symbols and a formula φ of size n_1 with alternation depth n_2 there is an algorithm deciding in time $\mathcal{O}((k2^{cmn_1 n_2})^{n_2})$ whether φ holds in the root of the pushdown tree $\mathcal{T}_{\mathcal{A}}$.

5.3 The lower bound

Finally we show a deterministic exponential time lower bound on the model checking problem for pushdown automata and (non alternating) μ-calculus. It follows from a quite standard reduction by simulating alternating linear space bounded Turing machines. The simulating automaton is very similar to the one described by Chandra, Kozen and Stockmeyer in [5].

Let M be an alternating linear space bounded Turing machine. We will assume that M has only one tape and on the input of size n it uses at most $n + 1$ tape squares along any computation path. Let $Q = Q_\exists \cup Q_\forall$ be the

set of states of M which is partitioned into existential and universal sates. Let Γ be a tape alphabet and let $\delta : Q \times \Gamma \to (Q \times \Gamma \times \{\text{left}, \text{right}\})^2$ be a transition function. A configuration of M is a string wqw' where $w, w' \in \Gamma^*$ and $q \in Q$, moreover ww' is of length $n+1$.

For a given machine M and a word w we construct a pushdown automaton $\mathcal{A}$ such that player I can reach a leaf in the game $\mathcal{T}_{\mathcal{A}}$ iff w is accepted by M. Initially the automaton $\mathcal{A}$ pushes on the stack the sequence of $n+1$ letters from $Q \cup \Gamma$ with exactly one letter from Q. Then the pushdown automaton arrives at an existential or universal state depending on whether the state q which was pushed on the stack was existential or universal. In this state pushdown automaton also remembers q and the letter a pushed on the stack just after q. It consults the transition function and chooses an element from the pair given by $\delta(q, a)$ or makes a universal branching to both elements. He puts this element on the stack and repeats the whole process of putting $n+1$ letters from $Q \cup \Gamma$ on the stack and deciding upon a next configuration. This cycle stops if in the latest sequence of $n+1$ letters an accepting state was pushed on the stack. At this point the pushdown automaton goes to an universal state *Check* and on the stack we have a sequence:

$$c_0(q_1, a_1, d_1)c_1 \ldots (q_k, a_k, d_k)c_k$$

The universal branching of $\mathcal{A}$ is used to check whether it is a sequence of configurations of M provided for every $i = 1, \ldots, k$, in the step i the move (q_i, a_i, d_i) was taken. Using universal choice the automaton goes to a state which checks whether c_k can be reached from c_{k-1} in the step (q_k, a_k, d_k) and it also goes to another state which just takes $(q_k, a_k, d_k)c_k$ from the stack and starts checking lower configurations. This is repeated until the first configuration is reached. The automaton then checks that this first configuration is of the form $q_0 w B^i$, where B^i is a sequence of blanks of appropriate length. Please observe that checking that one configuration follows from the previous one can be done using universal branching in finite memory linear in the size of M. The automaton just checks each letter separately. It can do so as it can count to $n+5$ in its finite control. If at some point one of the tests described above fails then the automaton enters into an infinite loop otherwise each of the tests stops in some final state. From this reduction we obtain.

Fact 28 There exists a formula α (without alternations) such that the problem "given a pushdown automaton $\mathcal{A}$, is α satisfied in the root of $T_{\mathcal{A}}$" is

DEXPTIME-hard. (Formula α expresses the fact that player I can reach a final state.)

Remark: This argument does not work for BPA processes as they correspond to pushdown automata without states and we needed states in our reduction. Indeed looking and the complexity of our algorithm we can see that if the automaton has only one state and k stack symbols and the formula has size n_1 and alternation depth n_2 then we can solve the model checking problem in time $\mathcal{O}((k2^{n_1 n_2})^{n_2})$. Hence in polynomial time if n_1, n_2 are fixed. In the case of alternation free formulas a similar complexity result was obtained in [2].
Remark: We conjecture that model checking is exponential also in the second parameter. That is, there exists a fixed pushdown process $\mathcal{A}$ such that the problem: "given a formula α, is α satisfied in the root of $T_{\mathcal{A}}$" is DEXPTIME hard.

References

[1] J. Bergstra and J. Klop. Process theory based on bisimulation semantics. volume 354 of *LNCS*, 1988.

[2] O. Burkart and B. Steffen. Model checking for context-free processes. In *CONCUR '92*, volume 630 of *LNCS*, pages 123–137, 1992.

[3] O. Burkart and B. Steffen. Pushdown processes: Parallel composition and model checking. In *CONCUR '94*, volume 836 of *LNCS*, 1994.

[4] D. Caucal and Monfort. On the transition graphs of automata and grammars. In *Graph-Theoretic Concepts in Computer Science, WG'90*, volume 484 of *LNCS*, 1991.

[5] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[6] S. Christiensen and H. Huttel. Deciding issues for infinite-state processes – a survey. *Bulletin of EATCS*, 51:156–166, October 1993.

[7] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency*, volume 803 of *LNCS*, pages 124–175. Springer-Verlag, 1993.

[8] B. Courcelle. On the extension to infinite graphs of properties of finite ones. In preparation.

[9] E. A. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *CAV'93*, volume 697 of *LNCS*, pages 385–396, 1993.

[10] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS 91*, 1991.

[11] E. A. Emerson and C. Lei. Efficient model checking in fragments of propositional mu-calculus. In *First IEEE Symp. on Logic in Computer Science*, pages 267–278, 1986.

[12] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In *CAV '95*, volume 939 of *LNCS*, pages 353–366, 1995.

[13] N. Klarund. Progress measures, immediate determinacy and a subset construction for tree automata. In *LICS '92*, pages 382–393, 1992.

[14] H. Lescow. On polynomial–size programs winning finite–state games. In *CAV '95*, volume 939 of *LNCS*, pages 239–252, 1995.

[15] D. E. Long, A. Browne, E. M. Clarke, S. Jha, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In *CAV'94*, volume 818 of *LNCS*, pages 338–350, 1994.

[16] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.

[17] D. Muller and P. Schupp. The theory of ends, pushdown automata and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[18] D. Niwiński and I. Walukiewicz. Games for μ-calculus. *Theoretical Computer Science*, 163:99–116, 1996.

[19] R. S. Streett and E. A. Emerson. An automata theoretic procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1989.

[20] W. Thomas. On the synthesis of strategies in infinite games. In *STACS '95*, volume 900 of *LNCS*, pages 1–13, 1995.

[21] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *CAV '95*, volume 939 of *LNCS*, pages 267–278, 1995.

[22] I. Walukiewicz. Monadic second order logic on tree-like structures. In *STACS '96*, volume 1046 of *LNCS*, pages 401–414, 1996.

Recent Publications in the BRICS Report Series

RS-96-54 Igor Walukiewicz. *Pushdown Processes: Games and Model Checking*. December 1996. 31 pp. Appears in Alur and Henzinger, editors, *8th International Conference on Computer-Aided Verification*, CAV '96 Proceedings, LNCS 1102, 1996, pages 62–74.

RS-96-53 Peter D. Mosses. *Theory and Practice of Action Semantics*. December 1996. 26 pp. Appears in Penczek and Szalas, editors, *Mathematical Foundations of Computer Science: 21st International Symposium*, MFCS '96 Proceedings, LNCS 1113, 1996, pages 37–61.

RS-96-52 Claus Hintermeier, Hélène Kirchner, and Peter D. Mosses. *Combining Algebraic and Set-Theoretic Specifications (Extended Version)*. December 1996. 26 pp. Appears in Haveraaen, Owe and Dahl, editors, *Recent Trends in Data Type Specification: 11th Workshop on Specification of Abstract Data Types, joint with 8th COMPASS Workshop*, Selected Papers, LNCS 1130, 1996, pages 255–274.

RS-96-51 Claus Hintermeier, Hélène Kirchner, and Peter D. Mosses. *R^n- and G^n-Logics*. December 1996. 19 pp. Appears in Gilles, Heering, Meinke and Möller, editors, *Higher-Order Algebra, Logic, and Term-Rewriting: 2nd International Workshop*, HOA '95 Proceedings, LNCS 1074, 1996, pages 90–108.

RS-96-50 Aleksandar Pekeč. *Hypergraph Optimization Problems: Why is the Objective Function Linear?* December 1996. 10 pp.

RS-96-49 Dan S. Andersen, Lars H. Pedersen, Hans Hüttel, and Josva Kleist. *Objects, Types and Modal Logics*. December 1996. 20 pp. To be presented at the *4th International Workshop on the Foundations of Object-Oriented*, FOOL4, 1997.

RS-96-48 Aleksandar Pekeč. *Scalings in Linear Programming: Necessary and Sufficient Conditions for Invariance*. December 1996. 28 pp.