Pushdown processes: games and model-checking[1,2]

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

Abstract

A pushdown game is a two player perfect information infinite game on a transition graph of a pushdown automaton. A winning condition in such a game is defined in terms of states appearing infinitely often in the play. It is shown that if there is a winning strategy in a pushdown game then there is a winning strategy realized by a pushdown automaton. An EXPTIME procedure for finding a winner in a pushdown game is presented. The procedure is then used to solve the model-checking problem for the pushdown processes and the propositional μ-calculus. The problem is shown to be 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 interested in the graph of configurations of a pushdown process and not in the language it recognizes. On such a graph of configurations we can define a two player infinite game. A move in the game consists of prolonging a path constructed so far. The result of the game is an infinite path. Winning

[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

conditions are defined in terms of states appearing infinitely often in the play. We call such games *pushdown games*.

A graph of a pushdown game may be an infinite graph that is not an unwinding of any finite graph (see [4] for interesting examples). In this way pushdown games generalize finite games. On the other hand, pushdown games can be presented in a finite way so it makes sense to ask what is the complexity of deciding who has a winning strategy in such a game. It is also interesting to know whether a winning strategy can be presented in a finite way. These are the questions we answer in this paper.

A motivation for studying the complexity question comes from the model-checking problem. The μ-calculus model-checking problem is: given a transition system and a formula of the μ-calculus decide if the formula holds in the initial state of the transition system. If the transition system is finite and given explicitly then the problem is in co-NP$\cap$NP [9] but its exact complexity is unknown. We show that in the case of transition systems given by pushdown processes the model-checking problem is EXPTIME-complete and it is so even for some fixed formula.

A motivation for studying the question of finite representations of winning strategies comes from program synthesis. It was suggested by Wolfgang Thomas [25]. One may consider a nonterminating reactive program as a player in a two person infinite game, the other player being the environment. The program is correct if it wins the game no matter what the environment does. Hence a correct program is a winning strategy in the game (see [19] for background and references). In the case of finite graphs, whenever a winning condition in a game is given by a property of the set of states visited infinitely often (i.e., by a Muller condition) then the winning strategy for each of the players is finite [13]. On the other hand, Wolfgang Thomas [25] shows an example of a game defined by a Turing machine in which the first player has a strategy but no hyperarithmetical one. In the same paper he asks what happens in the "intermediate case" when a game is given by a pushdown automaton. We show that in this case the strategy can be also given by a pushdown automaton but the size of this automaton may be in generally exponentially bigger than the size of the automaton defining the game.

The decidability of the model-checking problem for pushdown processes and the propositional μ-calculus follows from [18]. This decidability result as well as extensions of it (for example [7]) deal with monadic second order logic and reduce the problem to the decidability of $S2S$, hence give nonelementary algorithms. An elementary model-checking procedure for pushdown processes and alternation free fragment of the calculus was given in [3]. Independently from the present work, Sebastian Seibert [22] has shown that in every pushdown game there exist (for the player who wins) a winning strategy realizable by a pushdown automaton and that this strategy can be computed effectively.

Pushdown processes are a strict generalization 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 recognize the same languages as the general pushdown automata. This is not the case when configuration graphs are considered. It was shown in [4] that there exists a pushdown automaton whose 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 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. Throughout the paper we will work only with parity conditions. The results are almost the same for other winning conditions. We mention the differences next to the results. 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.

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 the 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, *pop* deletes the top element of the stack, *push*(z) puts z on the top of the stack.

A *pushdown process* (or 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.

As we can see, pushdown processes are syntactically just pushdown automata. A different name is used to stress the difference in the semantics. We will not be interested in the language accepted by a pushdown process but in the graph of configurations it generates. It will be more convenient to consider unwindings of this graph to a tree.

Definition 1 (Pushdown tree) Let $\mathcal{A}$ be a pushdown automaton as in (1). The *pushdown tree* determined by $\mathcal{A}$ is the smallest tree $\mathcal{T}_{\mathcal{A}} \subseteq (\Sigma_s^+ \times Q)^+$ such that:

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

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

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 that 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. Our case is different than the case when we are interested in the languages accepted by automata; in the later case the blowup is only polynomial.

2.2 Propositional μ-calculus

Let $Prop = \{p_1, p_2, \ldots\}$ be a set of *propositional constants* and let $Var = \{X, Y, \ldots\}$ 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, $\alpha, \beta, \ldots$ 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 $\rho : \mathit{Prop} \to \mathcal{P}(S)$ 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* $V : \mathit{Var} \to \mathcal{P}(S)$, the set of states in which a formula φ is true, denoted $\| \varphi \|_V^{\mathcal{M}}$, is defined inductively as follows:

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

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 an arbitrary closed formula.

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

A μ-calculus formula α is *alternation free* if it has no subformula of the form $\mu X. \beta(\nu Y. \gamma(X,Y), X)$ (or with μ and ν interchanged) with the occurrence of X in γ being free in $\beta(\mu Y. \gamma(X,Y), X))$. In other words α should have no true nestings of different fixpoints. An *alternation depth* of a formula is the longest chain of true nestings in the formula. We refer the reader to [20] for the full definition of this notion as well as for the background and intuitions behind it. Here we will use alternation depth only when quoting results from the literature.

A *model-checking problem* is to decide whether for a given model $\mathcal{M}$, state s, and formula α without free variables, the relation $\mathcal{M}, s \vDash \alpha$ holds. Here we will be interested in the case when $\mathcal{M}$ 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* and 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 [24]. The proof of the existence of memoryless strategies in parity games was given independently by Mostowski [17] and by Emerson and Jutla [10] (the case of finite graphs follows already from [8]). Klarlund [14] 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.

The proof below is a variation of the proof by Emerson and Jutla. The difference is that we use the notion of signature to a bigger extent. This approach allows us to show Proposition 11 which is essential to the argument in the next section.

A *game* $G = \langle V, V_I, V_{II}, E \subseteq V \times V, \Omega : V \to \{1, \dots, n\}\rangle$ is a bipartite labelled graph with the partition $V_I, V_{II} \subseteq V$ and the labelling Ω. The labels $1, \dots, n$ are called *priorities*. We say that a vertex v' is a *successor* of a vertex v if $E(v, v')$ holds.

A *play* from some vertex $v_1 \in V_I$ proceeds as follows: first player I chooses a successor v_2 of v_1, then player II chooses a successor v_3 of v_2, 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 $v_1, v_2, v_3, \dots$ This *path is winning* for player I if in the sequence $\Omega(v_1), \Omega(v_2), \Omega(v_3), \dots$ 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* ξ for player I is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex v from V_I a vertex $\xi(\vec{v}) \in V_{II}$ such that $E(v, \xi(\vec{v}))$ holds. A strategy is *memoryless* iff $\xi(\vec{v}) = \xi(\vec{w})$ 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 whose values for some elements don't matter.

Our main goal is the following theorem:

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

memoryless winning strategy.

The idea of the proof is the following. First we define a set $\mathcal{W}_I$ of nodes of G by a special fixpoint formula. Using this formula, to 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 the vertices not in $\mathcal{W}_I$.

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

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

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

Consider the formula:

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

where

$$N_i = \begin{cases} \bigvee\{Z_j : j \le i,\ j \text{ odd}\} & \text{if } i \text{ is odd} \\ \bigwedge\{Z_j : j \le i,\ j \text{ even}\} & \text{if } i \text{ is even} \end{cases}$$

We will be interested in the set:

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

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

To understand some intuitions behind this formula consider the formula: $\mu Z_n.\varphi_I(Z_1, \dots, Z_n)$. This formula holds in a node of the structure G if from this node player I can force the play in a finite number of steps into a node of a priority $i < n$ from which it is possible/necessary (depending on whose node it is) to reach a node in Z_j, for some odd $j < n$ or for all even $j < n$, respectively. Similarly the greatest fixpoint formula $\nu Z_n.\varphi_I(Z_1, \dots, Z_n)$

describes that player I can either stay forever in nodes of priority n or he can reach a node of a priority $i < n$ and, as before, form this node it is possible/necessary to reach a node in Z_j, for some odd $j < n$ or for all even $j < n$, respectively. Hence choosing appropriate fixpoint we can decide whether we should force a play to reach some smaller priority or whether it is enough to meet a given priority infinitely often.

Definition 3 When applied to n-tuples of ordinals symbols $=$, $<$, $\leq$ stand for the 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 an n-tuple of ordinals. An assignment $\mathcal{S}$ of signatures to nodes from some set $U \subseteq V$ is called *consistent* if for every $u \in U$ either: (i) $u \in V_I$ and there is a successor vertex $w \in U$ such that:

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

or (ii) $u \in V_{II}$ and for all successor vertices 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:

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

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:

$$v \in \| \varphi_I(P_1^{\vec{\tau}}, \ldots, P_n^{\vec{\tau}}) \|^G$$

where $P_1^{\vec{\tau}}, \ldots, P_n^{\vec{\tau}}$ are defined inductively by:

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

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:

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

Hence $(\tau, \dots, \tau)$ 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 $\| \mu^\rho X.\alpha(X) \|^G \subseteq \| \mu X.\alpha(X) \|^G$. Thus $v \in \mathcal{W}_I$ by monotonicity. $\square$

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

Proof
We will consider only the case when $v \in V_I$; the other case is analogous.

Let $(\tau_1, \dots, \tau_n)$ be the canonical signature of v. By the definition of the signature $Sig(v)$ we have $v \in \| \varphi_I(P_1^{\vec{\tau}}, \dots, P_n^{\vec{\tau}}) \|^G$ with $P_1^{\vec{\tau}}, \dots, P_n^{\vec{\tau}}$ as in that definition. Expanding the definition of φ_I we obtain: $v \in \| \langle\, \rangle N_{\Omega(v)} \|^G$. Hence there is a successor w of v with $w \in N_{\Omega(v)}$.

First, let us check the case when $\Omega(v)$ is odd. Expanding the definition of $N_{\Omega(v)}$ we get:

$$w \in \| \bigvee \{P_j^{\vec{\tau}} : j \leq \Omega(v), j \text{ odd}\} \|^G$$

So $w \in \| P_j^{\vec{\tau}} \|^G$ for some odd $j \leq \Omega(v)$. Recall that

$$P_j^{\vec{\tau}} = \mu^{\tau_j} Z_j.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{j-1}^{\vec{\tau}}, Z_j, \overrightarrow{Z})$$

(where $\overrightarrow{\sigma Z}$ abbreviates the sequence of fixpoint operators). It may happen that τ_j is not a successor ordinal but, by the definition of μ^τ, there is a successor ordinal $\rho \leq \tau_j$ such that:

$$w \in \| \mu^\rho Z_j.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{j-1}^{\vec{\tau}}, Z_j, \overrightarrow{Z}) \|^G$$

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

$$w \in \| \overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{j-1}^{\vec{\tau}}, P'_j, \overrightarrow{Z} \|^G$$

where $P'_j = \mu^{\rho-1}Z_j.\overrightarrow{\sigma Z}.\ \varphi_I(P_1^{\vec{\tau}}, \dots, P_{j-1}^{\vec{\tau}}, Z_j, \overrightarrow{Z})$. This shows that the canonical signature of w is not bigger than $(\tau_1, \dots, \rho - 1)$, on the first j positions, for some successor ordinal $\rho \leq \tau_j$. Hence, as $j \leq \Omega(v)$, we get $Sig(w) <_{\Omega(v)} Sig(v)$.

Now consider the case when $\Omega(v)$ is even. In this case expanding the definition of $N_{\Omega(v)}$ we obtain:

$$w \in \| \bigwedge\{P_j^{\vec{\tau}} : j \leq \Omega(v),\ j \text{ even}\} \|^G$$

In particular $w \in \| P_{\Omega(v)} \|^G$. Recall that:

$$P_{\Omega(v)}^{\vec{\tau}} = \nu Z_{\Omega(v)}.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{\Omega(v)-1}^{\vec{\tau}}, Z_{\Omega(v)}, \overrightarrow{Z})$$

Hence the canonical signature of w is not bigger than $(\tau_1, \dots, \tau_{\Omega(v)})$ on the first $\Omega(v)$ positions. $\square$

Definition 8 (Minimizing strategy) A *minimizing strategy* is a strategy taking for each node $v \in \mathcal{W}_I \cap V_I$ a successor having the smallest canonical signature.

Remark: Minimizing strategies may not be uniquely determined because a node may have many successors with the same signature.

The memoryless determinacy theorem follows from the two following lemmas.

Lemma 9 A minimizing strategy is a winning memoryless strategy for player I from every node in $\mathcal{W}_I$.

Proof
A minimizing strategy is memoryless as it is defined only using properties of a node in question. We show that it is winning.

Suppose $v_0 \in \mathcal{W}_I$. Let $\mathcal{P} = v_0, v_1, \dots$ be a history of a play when player I uses the minimizing 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 $j_1 < j_2 < \dots$ such that: no vertex after v_{j_1} has priority smaller than p, and $\Omega(v_{j_k}) = p$ for $k = 1, \dots$ 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. $\square$

Lemma 10 From every node not in $\mathcal{W}_I$ player II has a memoryless winning strategy.

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

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

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

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

Where N_i' is obtained from N_i by replacing each Z_j by $\neg Z_j$. Using the fact that in each vertex of G exactly one of the propositions $1,\ldots,n$ holds, the formula above is equivalent to:

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

Consider the game $G' = \langle V, V_{II}, V_I, E, \Omega'\rangle$ 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 a winning strategy for player I in G' translates to a strategy for player II in G and vice versa. In G' the complement of $\mathcal{W}_I$ is described by

$$\begin{aligned}&\nu Z_1.\mu Z_2.\ldots\nu Z_{n-1}.\mu Z_n.\\ &\quad\big(\neg I \Rightarrow \bigwedge_{i=1,\ldots,n} ((i+1) \Rightarrow [\,]\neg N_i')\big) \wedge \big(I \Rightarrow \bigwedge_{i=1,\ldots,n} ((i+1) \Rightarrow \langle\,\rangle\neg N_i')\big) \qquad (4)\end{aligned}$$

The change is that I is replaced by $\neg I$ and each i is replaced by $i+1$.

Please observe that $\neg N_i' = \bigwedge\{Z_i : j < i, j \text{ odd}\}$ for i odd and $\neg N_i' = \bigvee\{Z_i : j < i, j \text{ even}\}$ for i even. Denote by N_i'' the formula $\neg N_i'$ with indices of the variables increased by one. With this notation we can rewrite formula (4) to:

$$\Phi = \nu Z_2.\mu Z_2.\dots \nu Z_{n-1}.\mu Z_n.$$
$$\Big(\neg I \Rightarrow \bigwedge_{i=1,\dots,n} ((i+1) \Rightarrow [\ ]N''_i)\Big) \wedge \Big(I \Rightarrow \bigwedge_{i=1,\dots,n} ((i+1) \Rightarrow \langle\ \rangle N''_i)\Big) \quad (5)$$

We want to show that the formula (5) is equivalent over G' to the formula:

$$\mu Z_1.\nu Z_2.\dots \mu Z_{n+1}.\nu Z_{n+2}.\varphi_I(Z_1,\dots,Z_{n+2}) \quad (6)$$

First, as there are no vertices of priority 1 or $n+2$ in G', we can remove from $\varphi_I(Z_1,\dots,Z_{n+2})$ implications starting with 1 and $n+2$. So (6) above is equivalent to

$$\Psi = \mu Z_1.\nu Z_2.\dots \mu Z_{n+1}.\varphi'_I(Z_1,\dots,Z_{n+1})$$

where φ'_I does not have the above mentioned conjuncts and does not have νZ_{n+2} fixpoint because Z_{n+2} does not appear in the formula φ'_I.

From the definition of N''_i we get that $N''_i = N_{i+1}$ for odd i and $N''_i \vee Z_1 = N_{i+1}$ for even i. We have that

$$\parallel \Phi \parallel^{G'} = \parallel \nu Z_2.\dots \mu Z_{n+1}.\varphi'_I(\bot, Z_2,\dots,Z_{n+1}) \parallel^{G'}$$

Hence $\parallel \Phi \parallel^{G'} \subseteq \parallel \Psi \parallel^{G'}$.

Summarizing the proof of the lemma. If we take a vertex $v \notin \mathcal{W}_I$ then $v \in \parallel \Phi \parallel^{G'}$. Using $\parallel \Phi \parallel^{G'} \subseteq \parallel \Psi \parallel^{G'}$ and Lemma 10 we know that player I has a winning strategy from v in G'. By the definition of G' it means that player II has a winning strategy from v in G. $\square$

Let us finish with a fact pointing 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 signature assignments. The next fact implies that a minimizing strategy is in some sense an optimal strategy.

Proposition 11 The canonical signature assignment ($v \mapsto \mathit{Sig}(v)$) 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 $\mathit{Sig}(v)$ is defined and $\mathit{Sig}(v) \leq \mathcal{S}(v)$.

Before proving the proposition we need to show one more property stating the monotonicity of the formula φ_I with respect to signatures. In the lemma below we use the notation from Definition 5.

Lemma 12 Let $\vec{\tau} = (\tau_1, \dots, \tau_n)$ and $\vec{\rho} = (\rho_1, \dots, \rho_n)$ be two tuples of ordinals such that $\vec{\tau} <_k \vec{\rho}$ for some odd k. We have

$$\models \varphi_I(P_1^{\vec{\tau}}, \dots, P_n^{\vec{\tau}}) \Rightarrow \varphi_I(P_1^{\vec{\rho}}, \dots, P_n^{\vec{\rho}})$$

Proof
Let $\vec{\tau}$, $\vec{\rho}$ and k be as in the assumption of the lemma. We want to show that $\models N_i^{\vec{\tau}} \Rightarrow N_i^{\vec{\rho}}$ for all $i = 1, \dots, n$ where

$$N_i^{\vec{\tau}} = \begin{cases} \bigvee\{P_j^{\vec{\tau}} : j \leq i,\ j \text{ odd}\} & \text{if } i \text{ is odd} \\ \bigwedge\{P_j^{\vec{\tau}} : j \leq i,\ j \text{ even}\} & \text{if } i \text{ is even} \end{cases}$$

and similarly for $N_i^{\vec{\rho}}$. Clearly this would imply the thesis of the lemma.

By definition of $P_i^{\vec{\tau}}$ we get that $P_i^{\vec{\tau}} = P_i^{\vec{\rho}}$ for $i < k$. So $N_i^{\vec{\tau}} = N_i^{\vec{\rho}}$ for $i < k$.

For $i = k$ we have:

$$\begin{aligned} P_k^{\vec{\tau}} =& \mu^{\tau_k} Z_k.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{k-1}^{\vec{\tau}}, Z_k, \overrightarrow{Z}) \\ P_k^{\vec{\rho}} =& \mu^{\rho_k} Z_k.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\rho}}, \dots, P_{k-1}^{\vec{\rho}}, Z_k, \overrightarrow{Z}) \end{aligned}$$

As $\tau_k < \rho_k$ we get $\models P_k^{\vec{\tau}} \Rightarrow P_k^{\vec{\rho}}$ and $\models N_k^{\vec{\tau}} \Rightarrow N_k^{\vec{\rho}}$.

To show that $\models N_i^{\vec{\tau}} \Rightarrow N_i^{\vec{\rho}}$ for all odd $i > k$ it is enough to show that $\models P_i^{\vec{\tau}} \Rightarrow P_k^{\vec{\rho}}$ for all odd $i > k$. For this later fact it is even enough to show $\models P_{k+1}^{\vec{\tau}} \Rightarrow P_k^{\vec{\rho}}$. From the definition we have:

$$\begin{aligned} P_{k+1}^{\vec{\tau}} =& \nu Z_{k+1}.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_k^{\vec{\tau}}, Z_{k+1}, \vec{Z}) \\ P_k^{\vec{\rho}} =& \mu_k^{\rho} Z_k.\nu Z_{k+1}.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{k-1}^{\vec{\tau}}, Z_k, Z_{k+1}, \vec{Z}) \end{aligned}$$

in the above we use $P_i^{\vec{\tau}}$ instead of $P_i^{\vec{\rho}}$ because the two are the same for $i < k$.

We know that $\tau_k < \rho_k$, so in particular $\tau_k + 1 \leq \rho_k$. Let

$$P'' = \mu^{\tau_k+1} Z_k.\nu Z_{k+1}.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{k-1}^{\vec{\tau}}, Z_k, Z_{k+1}, \vec{Z})$$

We have $\models P'' \Rightarrow P_k^{\vec{\rho}}$ and $P'' = P_{k+1}^{\vec{\tau}}$ as

$$P'' \;=\; \nu Z_{k+1}.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{k-1}^{\vec{\tau}}, P_k^{\vec{\tau}}, Z_{k+1}, \vec{Z})$$

Finally we want to show that $\models P_i^{\vec{\tau}} \Rightarrow P_i^{\vec{\rho}}$ for all even $i = 1, \dots, n$. As $P_i^{\vec{\tau}} = P_i^{\vec{\rho}}$ for $i < k$ it is enough to consider the induction step for $i > k$. Consider

$$\begin{aligned} P_i^{\vec{\tau}} =& \nu Z_i.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{i-1}^{\vec{\tau}}, Z_i, \vec{Z}) \\ P_i^{\vec{\rho}} =& \nu Z_i.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\rho}}, \dots, P_{i-1}^{\vec{\rho}}, Z_i, \vec{Z}) \end{aligned}$$

Recall that

$$\varphi_I(P_1^{\vec{\tau}}, \ldots, P_{i-1}^{\vec{\tau}}, Z_i, \vec{Z}) =$$
$$(I \Rightarrow \bigwedge_{j=1,\ldots,n} (j \Rightarrow \langle\ \rangle \overline{N_j^{\vec{\tau}}})) \wedge (\neg I \Rightarrow \bigwedge_{j=1,\ldots,n} (j \Rightarrow [\ ] \overline{N_j^{\vec{\tau}}}))$$

where

$$\overline{N_j^{\vec{\tau}}} = \begin{cases} \bigvee\{P_l^{\vec{\tau}} : l < i, l \leq j, l \text{ odd}\} \cup \{Z_l : i \leq l \leq j, l \text{ odd}\} & j \text{ odd} \\ \bigwedge\{P_l^{\vec{\tau}} : l < i, l \leq j, l \text{ even}\} \cup \{Z_l : i \leq l \leq j, l \text{ even}\} & j \text{ even} \end{cases}$$

Similarly for ρ with $P_l^{\vec{\rho}}$ instead of $P_l^{\vec{\tau}}$ $(l = 1, \ldots, i-1)$.

By the induction hypothesis we have $\models P_l^{\vec{\tau}} \Rightarrow P_l^{\vec{\rho}}$ for all even $l < i$. By remarks made before we know that $\models P_l^{\vec{\tau}} \Rightarrow P_l^{\vec{\rho}}$ for all odd l. Hence $\models \overline{N_j^{\vec{\tau}}} \Rightarrow \overline{N_j^{\vec{\rho}}}$ for all j and we are done. □

Proof (of Proposition 11)
Assume conversely that there is a consistent signature assignment $\mathcal{S}$ for which the set of vertices $\{w : \mathcal{S}(w) < \mathit{Sig}(w)\}$ is not empty. Consider vertices from this set for which the difference is at the least position. Let v be one of such vertices for which $\mathcal{S}(v)$ is as small as possible. More precisely v is a vertex such that for some i we have:

- $\mathcal{S}(v) <_i \mathit{Sig}(v)$,
- for every w, $\mathcal{S}(w) \geq_{i-1} \mathit{Sig}(w)$,
- for every w, if $\mathcal{S}(w) <_i \mathit{Sig}(w)$ then $\mathcal{S}(v) \leq_i \mathcal{S}(w)$.

Observe that, the definition implies that i is odd.

Given a sequence of sets of vertices $\vec{Q} = (Q_1, \ldots, Q_i)$ we consider the formula:

$$\nu Z_{i+1} \ldots \mu Z_{n-1}.\nu Z_n.\varphi_I(Q_1, Q_2, \ldots, Q_i, Z_{i+1}, \ldots, Z_n)$$

(this is the formula used to define canonical signatures with variables $Z_1, \ldots, Z_i$ replaced by sets $Q_1, \ldots, Q_i$.) We abbreviate this formula by $\overrightarrow{\sigma Z}.\psi(\vec{Q}, \vec{Z})$.

Claim 12.1 Let u be a vertex and $\vec{Q}$ a sequence of sets of vertices with $\mathcal{S}(u)$ defined and $u \notin \parallel \overrightarrow{\sigma Z}.\psi(\vec{Q}, \vec{Z}) \parallel^G$. Suppose player I always chooses a vertex with the minimal value of $\mathcal{S}$ signature. We claim that player II can force the play into a vertex from $\{w : \Omega(w) \leq i \wedge \mathcal{S}(w) \leq_i \mathcal{S}(u)\} \cap \parallel \neg\overrightarrow{\sigma Z}.\psi(\vec{Q}, \vec{Z}) \parallel^G$.

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

$$u \in \parallel \mu Z_{i+1}.\nu Z_{i+2}\ldots\nu Z_{n-1}.\mu Z_n.\overline{\varphi}_I(\neg Q_1,\ldots,\neg Q_i,Z_{i+1},\ldots,Z_n) \parallel^G \tag{7}$$

where

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

and

$$\overline{N}_i = \begin{cases} \bigvee\{Z_j : j \leq i,\ j \text{ even}\} & \text{if } i \text{ is even} \\ \bigwedge\{Z_j : j \leq i,\ j \text{ odd}\} & \text{if } i \text{ is odd} \end{cases}$$

Let $\overline{Sig}(u)$ denote the signature of the formula (7) 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 successor 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 $\mathcal{S}(w) \leq_i \mathcal{S}(u)$ and $w \in \parallel \neg\sigma\overrightarrow{Z}.\psi(\vec{Q},\vec{Z}) \parallel^G$ $\square$

We proceed with the proof of the proposition. 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.

Let $\mathcal{S}(v) = (\tau_1,\ldots,\tau_n)$. Because $\mathcal{S}(v) <_i Sig(v)$ we know from Lemma 12 that:

$$v \notin \parallel \nu Z_{i+1}.\mu Z_{i+2}\ldots\nu Z_n.\varphi_I(P_1^{\vec{\tau}},\ldots,P_i^{\vec{\tau}},Z_{i+1},\ldots,Z_n) \parallel^G \tag{8}$$

where $P_1^{\vec{\tau}},\ldots,P_i^{\vec{\tau}}$ are as in Definition 5. Let us abbreviate the formula in (8) by θ.

By Claim 12.1 we can find a node $w \notin \| \theta \|^G$ with $\mathcal{S}(w) \le \mathcal{S}(v)$ and $\Omega(w) \le i$.

Suppose $\Omega(w)$ is even. Using $w \notin \| \theta \|^G$ we can find a successor w' of w with $\mathcal{S}(w') \le_{\Omega(w)} \mathcal{S}(w)$ and $w' \notin \bigwedge\{P_j^{\vec{\tau}} : 1 \le j \le \Omega(w),\ j \text{ even}\}$. Hence $w' \notin P_j^{\vec{\tau}}$ for some even $j \le \Omega(w)$. This means that

$$w' \notin \nu Z_j.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{j-1}^{\vec{\tau}}, Z_j, \vec{Z})$$

By Lemma 12, we know that $Sig(w') > (\tau_1, \dots, \tau_j, 0, \dots, 0)$. So $Sig(w') >_j \mathcal{S}(v) \ge_{\Omega(w)} \mathcal{S}(w')$. A contradiction with the choice of v because $j < i$.

Now suppose $\Omega(w)$ is odd. From $w \notin \| \theta \|^G$ we obtain that there is a successor w' of w with $\mathcal{S}(w') <_{\Omega(w)} \mathcal{S}(w)$ and $w' \notin \bigvee\{P_j^{\vec{\tau}} : 1 \le j \le \Omega(w),\ j \text{ odd}\}$. Let k be the maximal odd position not greater than $\Omega(w)$ with $\tau_k > 0$. Such a position must exist because $\mathcal{S}(w') <_{\Omega(w)} \mathcal{S}(w) \le_i \mathcal{S}(v)$ and this implies that there should be a nonzero value on one of the first i positions of $\mathcal{S}(v)$. We get $w' \notin P_k^{\vec{\tau}}$ and expanding the definition of $P_k^{\vec{\tau}}$ we have:

$$w' \notin \mu^{\tau_k} Z_k.\overrightarrow{\sigma Z}.\varphi_I(P_1^{\vec{\tau}}, \dots, P_{k-1}^{\vec{\tau}}, Z_k, \vec{Z})$$

Hence, by Lemma 12, we know that $Sig(w') \ge (\tau_1, \dots, \tau_k, 0, \dots, 0)$. So $Sig(w') \ge_k \mathcal{S}(v)$. By our choice of k we have $\tau_{k+1}, \dots, \tau_{\Omega(w)} = 0$, hence $\mathcal{S}(v) >_k \mathcal{S}(w')$. This gives $Sig(w') \ge_{\Omega(w)} \mathcal{S}(v) >_k \mathcal{S}(w')$. 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, Q_{II}) and 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 (q', pop), $(q', skip)$ or $(q', push(z'))$ is in $\delta(z, q)$ then: $q \in Q_I$ iff $q' \in Q_{II}$.

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

Definition 13 (Pushdown game) An automaton $\mathcal{A}$ together with a partition of states Q_I, Q_{II} and a priority function Ω define the pushdown game $G_{\mathcal{A}} = \langle V, V_I, V_{II}, E, \Omega : V \to \{0, \dots, n\}\rangle$ where $\langle V, E\rangle$ is a pushdown tree $\mathcal{T}_{\mathcal{A}}$ (see Definition 1) 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: Assuming that the initial state belongs to Q_{II}, from our postulate about partition of the states of $\mathcal{A}$ 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. We have made this choice because we are interested in the winning conditions definable in $S1S$ or in the μ-calculus.

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:

$$\begin{aligned} \mathcal{B} = \langle & Q_{\mathcal{B}}, \Sigma_i, \Sigma_o, \Sigma_{s,\mathcal{B}}, q_0, \perp, \\ & \delta_{\mathcal{B}} : Q_{\mathcal{B}} \times \Sigma_{s,\mathcal{B}} \times (\Sigma_i \cup \{\tau\}) \to Q_{\mathcal{B}} \times Com(\Sigma_{s,\mathcal{B}}) \times (\Sigma_o \cup \{\tau\}) \rangle \end{aligned} \tag{9}$$

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 $\perp$ 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.: whenever $m_1, n_1, \dots, m_{k-1}, n_{k-1}, m_k$ is a valid sequence of moves with $m_1, \dots, m_k$ being the moves read by $\mathcal{B}$ and $n_1, \dots, n_{k-1}$ being the moves written by $\mathcal{B}$ then $\mathcal{B}$ should output some move n_k such that $m_1, n_1, \dots, m_k, n_k$ 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 *winning 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 14
If there is a winning strategy for player I in $G_{\mathcal{A}}$ then there is a winning pushdown strategy.

Of course the result holds also for other winning conditions as far as they are defined in terms of the set of states of $\mathcal{A}$ appearing infinitely often in the play. To see this it is enough to use a translation of, the most general, Muller conditions into parity conditions [16].

Let us now try to explain the idea of the construction of the strategy. First, we know that if there is a strategy for player I then there is a minimizing 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 $\mathcal{A}$.

Let us try to describe what kind of information about the stack we need. Consider a run of $\mathcal{A}$. Suppose that in a configuration (s, q) 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, instead of knowing the whole s, just knew what states he can reach when taking z from the stack. In general he will need a sequence of sets of states $\vec{A} = \{A^p\}_{p=1,\dots,n}$, 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 formalizes this intuition in the notion of sub-game, $G(\vec{A}, z, \theta, q)$. The additional parameter θ is used to remember the smallest priority seen since we have put the current top symbol on the stack.
Notation: We assume that $\{1, \dots, n\}$ is the range of Ω. We use $\vec{A}$ to range over n element vectors of sets of states and θ to range over $\{1, \dots, n\}$. We also use z to range over stack symbols and q to range over states.

Definition 15 (Sub-game) For every quadruple $\vec{A}$, z, θ, q we define the sub-game $G(\vec{A}, z, \theta, q)$ as follows. The graph of the game is a tree of configurations of $\mathcal{A}$ started in the configuration $(\perp z, q)$. Every node of this tree 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 (not counting 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 then player I wins iff the obtained path is winning (cf. page 6)

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.

Summarizing, 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 he is allowed to reach when popping the current top 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 16 (Signature, Hint) Suppose that player I has a winning strategy in a sub-game $G(\vec{A}, z, \theta, q)$. Define $Sig(\vec{A}, z, \theta, q)$ to be the canonical signature of the root of this 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 the label of v is $(\perp, q')$ then let $Hint(\vec{A}, z, \theta, q) = (q', pop)$. If the label of v is $(\perp z, q')$ then let $Hint(\vec{A}, z, \theta, q) = (q', skip)$. Otherwise the label of v is $(\perp z z', q')$ 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 17 (Update function) Define $Up(\vec{A}, z, \theta, q)$ to be the sequence of sets $\vec{B} = \{B^p\}_{p=1,\dots,n}$, where each B^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 needed to define the strategy automaton.

Definition 18 (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., $\Sigma_i = \Sigma_o = Q \times Com(\Sigma_s)$. 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 $repmin(\theta')$ that 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 $(s\vec{A}_1z_1\theta_1\vec{A}z\theta, q)$ then after this operation we obtain the configuration $(s\vec{A}_1z_1\min(\theta_1,\theta'), q')$. Let us proceed with the definition of $\delta_{\mathcal{B}}$: If $q \in Q_I$ then:

$$\begin{aligned}
\delta_{\mathcal{B}}(q,\vec{A}z\theta,\tau) =&(q', repmin(\Omega(q)), \text{“}(q', skip)\text{”})\\
&\text{if } Hint(\vec{A},z,\theta,q) = (q', skip)\\
\delta_{\mathcal{B}}(q,\vec{A}z\theta,\tau) =&(q', pop; repmin(\min(\theta,\Omega(q))), \text{“}(q', pop)\text{”})\\
&\text{if } Hint(\vec{A},z,\theta,q) = (q', pop)\\
\delta_{\mathcal{B}}(\vec{A}z\theta,q,\tau) =&(q', repmin(\Omega(q)); push(\vec{A}'z'n), \text{“}(q', push(z'))\text{”})\\
&\text{if } Hint(\vec{A},z,\theta,q) = (q', push(z')) \text{ and } \vec{A}' = Up(\vec{A},z,\theta,q)
\end{aligned}$$

If $q \in Q_{II}$ then:

$$\begin{aligned}
\delta_{\mathcal{B}}(q,\vec{A}z\theta, \text{“}(q', skip)\text{”}) =&(q', repmin(\Omega(q)), \tau)\\
&\text{if } (q', skip) \in \delta_{\mathcal{A}}(q,z)\\
\delta_{\mathcal{B}}(q,\vec{A}z\theta, \text{“}(q', pop)\text{”}) =&(q', pop; repmin(\min(\theta,\Omega(q))), \tau)\\
&\text{if } (q', pop) \in \delta_{\mathcal{A}}(q,z)\\
\delta_{\mathcal{B}}(q,\vec{A}z\theta, \text{“}(q', push(z'))\text{”}) =&(q', repmin(\Omega(q)); push(\vec{A}'z'n), \tau)\\
&\text{if } (q', push(z')) \in \delta_{\mathcal{A}}(q,z) \text{ and } \vec{A}' = Up(\vec{A},z,\theta,q)
\end{aligned}$$

The first lemma shows that the definition of $Up(\vec{A},z,\theta,q)$ has good properties.

Lemma 19 Suppose that player I can win in $G(\vec{A},z,\theta,q)$. If

$$\delta_{\mathcal{B}}(q,\vec{A}z\theta,\tau) = (q_1, repmin(\Omega(q)); push(\vec{A}_1z_1n), \text{“}(q, push(z_1))\text{”})$$

or

$$\delta_{\mathcal{B}}(q,\vec{A}z\theta, \text{“}(q_1, push(z_1))\text{”}) = (q_1, repmin(\Omega(q)); push(\vec{A}_1z_1n), \tau)$$

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 $\mathcal{F}((\perp s', q') \cdots (\perp s'', q'')) = (\perp z s', q') \cdots (\perp z s'', q'')$, 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 $(\perp z z_1, q_1)$. This node is a son of the root of G.

Let σ denote a minimizing strategy in G. Because $(\perp z z_1, q_1) \in \sigma$ we can use the function $\mathcal{F}$ to obtain the 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 $(\perp z z_1, q_1)$ to $\mathcal{F}(w)$. The whole situation is presented in Figure 1

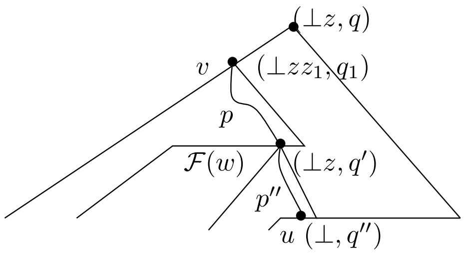

Figure 1: Proof of Lemma 19

According to Definition 15 the node w is marked winning if $q' \in A_1^p$. By the definition of the automaton $\mathcal{B}$ we know that $\vec{A}_1 = Up(\vec{A}, z, \theta, q)$. Hence we have to show that:

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

and that the inequality is strict if $\min(\Omega(q), p)$ is odd.

Let us denote $(\perp z z_1, q_1)$ by v and use the subscript G in $Sig_G(x)$ to stress that this is the canonical signature of x in the game G.

Claim 19.1 $Sig_G(\mathcal{F}(w)) \leq_{\min(\Omega(q),p)} Sig(\vec{A}, z, \theta, q)$ 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 $\mathcal{F}(w)$, by consistency of canonical signatures (Fact 7) we have $Sig_G(v) \leq_{\Omega(q)} Sig(\vec{A}, z, \theta, q)$ and is strictly smaller if $\Omega(q)$ is odd. By the same fact $Sig_G(\mathcal{F}(w)) \leq_p Sig_G(v)$ and it is strictly smaller if p is odd. $\square$

Claim 19.2 $Sig_G(\mathcal{F}(w)) = Sig_G(\vec{A}, z, \min(\Omega(q), p, \theta), q')$

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 $q'' \in A^{\min(p'',\min(\Omega(q),p,\theta))}$. $\square$

Knowing that σ_1 is winning in G_1 we can define a signature assignment by $\mathcal{S}(x) = Sig(\mathcal{F}(x))$ for every x reachable in a play of G_1 when player I uses σ_1. This is a consistent signature assignment, hence by Proposition 11 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. $\square$

With a help of Lemma 19, by induction on the length of the derivation we obtain:

Lemma 20 If a configuration $(s\vec{A}z\theta, q)$ of $\mathcal{B}$ is reachable from the initial configuration then $Sig(\vec{A}, z, \theta, q)$ is defined.

The next lemma describes the main property of $\mathcal{B}$ with respect to signatures.

Lemma 21 Suppose that player I can win the game $G_{\mathcal{A}}$. Let $(s\vec{A}z\theta, q)$ 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 a configuration $(s\vec{A}z\theta, q)$ to a configuration $(s\vec{A}z\theta', q')$ and $s\vec{A}z$ is always on the stack during this derivation. Let p be the minimum of the priorities of the states appearing in the derivation (not counting q'). We have:

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

Proof
The proof proceeds by induction on the length of derivation. We will use $c \to c'$ to mean that the configuration c' can be obtained from c in one step of the computation of $\mathcal{B}$ (we do not indicate what inputs and outputs occurred in the move).

The case $(s\vec{A}z\theta, q) \to (s\vec{A}z\theta', q')$ follows directly from the construction of the automaton.

Suppose that a derivation has the form:

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

and $\vec{A}_1 z_1$ was not popped in between. By induction assumption, θ_1 is the minimum of priorities of states that 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 20 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:

$$Sig(\vec{A}, z, \min(\Omega(q), p_1, \theta), q'') \leq_{\min(\Omega(q), p_1)} Sig(\vec{A}, z, \theta, q)$$

and the inequality is strict if $\min(\Omega(q), p_1)$ is odd. It is easy to see that $\theta'' = \min(\Omega(q), p_1, \theta)$.

The remaining case is when the derivation can be divided into two sequences:

$$(s\vec{A}z\theta, q) \to \cdots \to (s\vec{A}z\theta', q') \to \cdots \to (sAz\theta'', q'')$$

This case follows directly from two applications of the induction assumption.
□

Lemma 22 $\mathcal{B}$ is a strategy automaton.

Proof
The automaton $\mathcal{B}$ is constructed in such a way that from a 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. The construction of $\mathcal{B}$ also guarantees that $\mathcal{B}$ outputs only valid moves. By this we mean that if $\mathcal{B}$ has read $m_1, \ldots, m_i$, has written $n_1, \ldots, n_i$ and $m_1, n_1, \ldots, m_i$ 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 suppose that the sequence $m_1, n_1, \ldots$ determines a loosing (for player I) path $\mathcal{P}$ 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, a contradiction with our assumption. Hence $q \in Q_I$. From Lemma 20 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 an infinite 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) the configuration at the position x_0 is $(sz\vec{A}\theta, q)$ 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 21 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 to 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 19 and 21 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 ad infinitum 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, as the memoryless determinacy theorem shows, no memory is need.

An example of a game that has only big strategy automata 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. The graph of such a game can be defined by a pushdown automaton of size $\mathcal{O}(n^2)$. 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 $\mathcal{T}_\mathcal{A}$ satisfies a given formula φ of the propositional μ-calculus. First, we will construct a pushdown game such that player I has a winning strategy in this game iff $\mathcal{T}_\mathcal{A}$ satisfies φ. Next we will show how to reduce the problem of finding a winner in a pushdown game to a problem of finding a winner in a finite game. This will give an EXPTIME algorithm for the model-checking problem. Finally, we will show the EXPTIME lower bound on the complexity of the model-checking problem

5.1 Reduction to games

Take a pushdown automaton $\mathcal{A}$ and a μ-calculus formula φ. In this subsection we will construct a pushdown automaton $\mathcal{C}$ (depending on $\mathcal{A}$ and φ) together with a partition of its states $Q_I, Q_{II} \subseteq Q_\mathcal{C}$ and a function $\Omega : Q_\mathcal{C} \to \mathbb{N}$. These will define a pushdown game in which player I can win iff formula φ is satisfied in the root of $\mathcal{T}_\mathcal{A}$.

Let us start with some technical definitions concerning μ-calculus formulas. These will facilitate the description of the reduction.

Definition 23 (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 $\nu X.\beta(X)$, 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 is called the *binding function* associated with φ.

Definition 24 (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 say that a bound variable Y depends on a bound variable X in φ when $X \leq_\varphi Y$.

Definition 25 (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 closed μ-calculus formula. Without a loss of generality we may assume that it is well named. Let $X_1, \dots, X_n$ 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:

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

be a pushdown automaton as in (1). This automaton defines a pushdown tree $\mathcal{T}_\mathcal{A}$ (see Definition 1). We want to check if the root of this tree satisfies a given μ-calculus formula φ. To make it easier to talk about the properties of such tree we will have in the μ-calculus a proposition P_q for every state $q \in Q$. This proposition holds in a node of $\mathcal{T}_\mathcal{A}$ iff q is in the label of the node.

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:

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

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

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

To define the game $G_{\mathcal{C}}$ 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 with the label containing a state: $(q, P_{q'})$, $(q, \alpha \vee \beta)$, $(q, \mu X.\alpha(X))$, $(q, \nu X.\alpha(X))$, (q, X), or $(q, \langle\ \rangle\alpha)$; for some formulas α, β and some $q, q' \in Q$ with $q \neq q'$. 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 26
$\mathcal{T}_{\mathcal{A}} \models \varphi$ iff there is a winning strategy for player I in the game $G_{\mathcal{C}}$ described above.

For finite transition systems a very similar theorem was shown by Emerson, Jutla and Sistla [9]. To prove the theorem in the left to right direction one can use signatures of φ. For the right to left implication assume conversely and show that player II has a winning strategy. See for example [24] or [21] for similar arguments.

5.2 Establishing existence of winning strategies

Consider a pushdown automaton:

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

Let $Q_I, Q_{II} \subseteq Q$ be a partition of Q and let $\Omega : Q \to \{1, \ldots, n\}$ be an indexing function. These define the pushdown game $G_{\mathcal{A}}$ (see Definition 13). In this subsection we are concerned with the problem: given $\mathcal{A}$, Q_I, Q_{II}, 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 $\mathcal{A}$, Q_I, Q_{II}, and Ω be fixed for the rest of this subsection.

We will show a method of establishing a winner in all the games $G(\vec{A}, z, \theta, q)$. Please recall that the game $G_{\mathcal{A}}$ is $G((\emptyset, \dots, \emptyset), \perp, n, q_0)$ in this notation. Suppose we want to show that we can win in a game $G(\vec{A}, z, \theta, q)$ for some $q \in Q_I$. Immediately from the definition of the game it follows that we are done if $(q', \mathit{pop}) \in \delta(q, z)$ and $q' \in A^{\min(\theta, \Omega(q))}$. We are also done if $(q', \mathit{skip}) \in \delta(q, z)$ and we somehow know that we can win $G(\vec{A}, z, \min(\theta, \Omega(q)), q')$. We will see later that "somehow know" will be strongly related with the signature of $G(\vec{A}, z, \min(\theta, \Omega(q)), q')$ being smaller (or not bigger) than the signature of $G(\vec{A}, z, \theta, q)$.

Finally we need to deal with the most difficult case of *push* operation. Suppose that the best first move of player I in $G(\vec{A}, z, \theta, q)$ is take the transition $(q', \mathit{push}(z'))$. The play in $G(\vec{A}, z, \theta, q)$ looks as follows. It starts from the root $(\perp z, q)$ of the game. Then it proceeds to $(\perp z z', q')$. Then it either never pops z' from the stack or finally does so and reaches a configuration $(\perp z, q'')$. In the first case we can forget about z as it will never influence the play. The second case is more interesting. We cannot afford to keep z in stack memory but we can use alternation instead. We will guess the set B of all the states q'' as above and start "in parallel" checking of what happens from positions $(\perp z, q'')$. Because we have priorities around we will divide B into $B^1, \dots, B^n$ with B^p being the set of states such that $(\perp z, q'')$ can be reached in some play from $(\perp z, q)$ with p being the smallest priority seen in this play. We check in parallel if we can win in $G(\vec{B}, z', n, q')$ and $G(\vec{A}, z, \min(\theta, p), q'')$ for all $p \leq \Omega(q)$ and $q'' \in B^p$. This way we have reduced the task of checking that we can win in $G(\vec{A}, z, \theta, q)$ to checking that we can win in several other games.

We will construct a finite game $\mathcal{M}_{\mathcal{A}}$ having a node $\mathit{Check}(\vec{A}, z, \theta, p, q)$ for all possible $\vec{A}, z, \theta, q$ and $p \in \{1, \dots, n\}$. The additional parameter p is used to tell what happened between push and pop exactly as in the above description. We will also have nodes $\mathit{Push}(\vec{A}, z, \theta, q)$ that will have the same meaning as *Check* nodes; a different name will help in the correctness proof. A node $\mathit{Move}((\vec{A}, z, \theta, q), (?, z', q'))$ will be used to "implement" our handling of *push* operation. From this node Player I has to guess a tuple of sets of states $\vec{B}$ as in the above description. From a node $\mathit{Move}((\vec{A}, z, \theta, q), (\vec{B}, z', q'))$ player II has the opportunity to ask player I for an evidence to arbitrary of the *Check* nodes reachable from it. We describe the details in the definition below.

Definition 27 (Game $\mathcal{M}_{\mathcal{A}}$) We define a finite game $\mathcal{M}_{\mathcal{A}} = \langle S_M, \to, \Omega_M \rangle$

as follows. For every $\vec{A},\vec{B}\in Q^n$; $z,z'\in\Sigma_s$; $q,q'\in Q$; and $\theta,p\in\{1,\ldots,n\}$ we have nodes:

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

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

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

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

$$\begin{array}{l}
\mathit{Move}((\vec{A},z,\theta,q),(?,z',q'))\to\mathit{Move}((\vec{A},z,\theta,q),(\vec{B},z',q')) \quad \vec{B} \text{ arbitrary}\\
\mathit{Move}((\vec{A},z,\theta,q),(\vec{B},z',q'))\to\mathit{Push}(\vec{B},z',n,q')\\
\mathit{Move}((\vec{A},z,\theta,q),(\vec{B},z',q'))\to\mathit{Check}(\vec{A},z,\min(\theta,p),p,q'')\\
\hfill \text{if } p\leq\Omega(q) \text{ and } q''\in B^p
\end{array}$$

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

$$\mathit{Check}(\vec{A},z,\theta,p,q),\quad \mathit{Push}(\vec{A},z,\theta,q),\quad \mathit{Move}((\vec{A},z,\theta,q''),(?,z',q'))$$
for $q\in Q_I$ and arbitrary $\vec{A},z,z',\theta,p,q',q''$.

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

$$\mathit{Check}(\vec{A},z,\theta,p,q),\quad \mathit{Push}(\vec{A},z,\theta,q),\quad \mathit{Move}((\vec{A},z,\theta,q''),(\vec{B},z',q'))$$
for $q\in Q_{II}$ and arbitrary $\vec{A},\vec{B},z,z',\theta,p,q',q''$.

Priority function Ω_M is defined by:

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

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 $\mathit{Pop}(q)$, for some q, is reached.

- The game is infinite and the obtained infinite path $\mathcal{P}$ is winning for player I. Recall that this means that the minimal priority of states appearing infinitely often on $\mathcal{P}$ is even.

Theorem 28
Player I has a winning strategy in the game $G_{\mathcal{A}}$ (from the root node) iff he has a winning strategy in the game $\mathcal{M}_{\mathcal{A}}$ from the node $Check((\emptyset, \ldots, \emptyset), \perp, n, n, q_0)$.

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 $Hint(\vec{A}, z, \theta, q) = (q', skip)$ then:
$$\sigma_M(Check(\vec{A}, z, \theta, p, q)) = Check(\vec{A}, z, \min(\theta, \Omega(q)), \Omega(q), q')$$
$$\sigma_M(Push(\vec{A}, z, \theta, q)) = Push(\vec{A}, z, \min(\theta, \Omega(q)), q')$$

- If $q \in Q_I$, $Sig(\vec{A}, z, \theta, q)$ is defined and $Hint(\vec{A}, z, \theta, q) = (q', pop)$ then:
$$\sigma_M(Check(\vec{A}, z, \theta, p, q)) = \sigma_M(Push(\vec{A}, z, \theta, q)) = Pop(q')$$

- If $q \in Q_I$, $Sig(\vec{A}, z, \theta, q)$ is defined and $Hint(\vec{A}, z, \theta, q) = (q', push(z'))$ then:
$$\sigma_M(Check(\vec{A}, z, \theta, p, q)) = \sigma_M(Push(\vec{A}, z, \theta, q)) = Move((\vec{A}, z, \theta, q), (?, z', q'))$$

- If $q \in Q_I \cup Q_{II}$, $Sig(\vec{A}, z, \theta, q)$ is defined and $\vec{B} = Up(\vec{A}, z, \theta, q)$ then let:
$$\sigma_M(Move((\vec{A}, z, \theta, q), (?, z', q'))) = Move((\vec{A}, z, \theta, q), (\vec{B}, z', q'))$$

Let $\rightarrow_{\sigma_M}$ denote the subset of the transition relation of $\mathcal{M}_{\mathcal{A}}$ defined by the strategy σ_M, i.e., $\rightarrow_{\sigma_M} = \{(u, v) : \text{if } u \in V_I \text{ then } \sigma_M(u) = v\} \cap \rightarrow$. We will show that every path along $\rightarrow_{\sigma_M}$ starting in $Check((\emptyset, \ldots, \emptyset), \perp, n, n, q_0)$ is winning for player I.

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

- If $Check(\vec{A}, z, \theta, p, q) \rightarrow_{\sigma_M} Check(\vec{A}, z, \min(\theta, \Omega(q)), \Omega(q), q')$ then by Definitions 16 and 18 we have that: $Sig(\vec{A}, z, \min(\theta, \Omega(q)), q') \leq_{\Omega(q)} Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\Omega(q)$ is odd. Similarly for *Push* instead of *Check*.

- If $\mathit{Move}((\vec{A}, z, \theta, q), (\vec{B}, z', q')) \to_{\sigma_M} \mathit{Push}(\vec{B}, z', n, q')$ then by Lemma 19 we have: $\mathit{Sig}(\vec{B}, z', n, q') \leq_{\Omega(q)} \mathit{Sig}(\vec{A}, z, \theta, q)$ and it is strictly smaller if $\Omega(q)$ is odd.

- If $\mathit{Move}((\vec{A}, z, \theta, q), (\vec{B}, z', q')) \to_{\sigma_M} \mathit{Check}(\vec{A}, z, \theta'', p, q'')$ then by Definition 17 we have: $\mathit{Sig}(\vec{A}, z, \theta'', q'') \leq_p \mathit{Sig}(\vec{A}, z, \theta, q)$ and it is strictly smaller if p is odd.

- There is always $\mathit{Pop}(q_1)$ as required in the second clause of the definition of $\to_{\sigma_M}$.

- There is no $\to_{\sigma_M}$ transition to $\mathit{Err}(q_1)$, for arbitrary q_1.

With these properties it is quite easy to show that σ_M is a winning strategy. Let $\mathcal{P}$ be a play in $\mathcal{M}$ when player I uses σ_M. From the above observations it follows that whenever a node $\mathit{Check}(\vec{A}, z, \theta, p, q)$ appears on the path then $\mathit{Sig}(\vec{A}, z, \theta, q)$ is defined. Similarly for *Push* nodes and *Move* nodes. This means that player I can always make a move from these nodes. Hence if the play is finite then it ends in a $\mathit{Pop}(q)$ node and player I wins. Suppose the play $\mathcal{P}$ is infinite and the smallest priority met infinitely often is an odd number p. Let $\{(\vec{A}_i, z_i, \theta_i, q_i)\}_{i\in\mathbb{N}}$ be the sequence of tuples from *Check* or *Push* nodes in $\mathcal{P}$. Looking at the sequence of signatures: $\{\mathit{Sig}(\vec{A}_i, z_i, \theta_i, q_i)\}_{i\in\mathbb{N}}$ we obtain that from some moment the signatures never increase on the positions up to p and decrease infinitely often. This is impossible as the signatures are well order. Hence if the play is infinite then player I wins.

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

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

where S_M is the set of nodes of $\mathcal{M}_\mathcal{A}$ and $Q_\mathcal{C}$ is some set of auxiliary states needed to "implement" the necessary behaviour of δ_C that we describe below. The automaton will work in macro steps. In each macro step it will read or write one move of $\mathcal{A}$ and push or pop some nodes of $\mathcal{M}_\mathcal{A}$. Each macro step will start and finish 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 a *Check* or *Push* node on the top of the stack.

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

If $q \in Q_I$ then there is exactly one transition $m \to_{\sigma_M} u$. We add the following transitions to $\delta(q_0, m, \tau)$:

- 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 $q' \in A^{\min(\theta,\Omega(q))}$. Pop elements from the stack until a *Push* node is popped. At this moment the current node on the top of the stack must be of the form $Move((\vec{A}_1, z_1, \theta_1, q_1), (\vec{A}, z, q_2))$. Push the node $Check(\vec{A}_1, z_1, \min(\theta_1, p), p, q')$ where $p = \min(\theta, \Omega(q))$. Output "$(q', pop)$".
- If u is $Move((\vec{A}, z, \theta, q), (?, z', q'))$ then there are nodes u', u'' such that $u \to_{\sigma_M} u' \to_{\sigma_M} u''$, where u' is $Move((\vec{A}, z, \theta, q), (\vec{B}, z', q'))$ and u'' is $Push(\vec{B}, z', n, q')$. Add to $\delta(q_0, m, \tau)$ operations that push u' and u'' on the stack and output "$(q', push(z'))$"

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

- If u is $Check(\vec{A}, z, \theta, p, q')$ or $Push(\vec{A}, z, \theta, q')$ then read "$(q', skip)$" and replace m by u on the top of the stack. Do not produce any output.
- If u is $Pop(q')$ then $q' \in A^{\min(\theta,\Omega(q))}$. Read "$(q', Pop)''$ and then pop elements from the stack until a *Push* node is popped. The current top node of the stack must be of the form $Move((\vec{A}_1, z_1, \theta_1, q_1), (\vec{A}, z, q_2))$. Push the node $Check(\vec{A}_1, z_1, \min(\theta_1, p), p, q')$ where $p = \min(\theta, \Omega(q))$. Do not produce any output.
- If u is $Move((\vec{A}, z, \theta, q), (?, z', q')$ then there are nodes u', u'' such that $u \to_{\sigma_M} u' \to_{\sigma_M} u''$, where u' is $Move((\vec{A}, z, \theta, q), (\vec{B}, z', q'))$ and u'' is $Push(\vec{B}, z', n, q')$. Read "$(q', Push(z'))''$ and then push u' and u'' on the stack. Do not produce any output.

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

The following observation shows that $\mathcal{C}$ is a strategy automaton.

Observation 28.1 If $\mathcal{C}$ reads a sequence of valid moves of $\mathcal{A}$ then it outputs a sequence of valid moves of $\mathcal{A}$. The content of the stack of $\mathcal{C}$ forms a path in $\mathcal{M}_{\mathcal{A}}$ along $\to_{\sigma_{\mathcal{M}}}$. If $\mathcal{C}$ pops some elements from the stack and then pushes a $Check(\vec{A}, z, \min(\theta, p), q)$ node then p is the smallest priority of the nodes popped from the stack.

□

The size of the transition system $\mathcal{M}_{\mathcal{A}}$ is $\mathcal{O}(k2^{cmn})$ where k is the size of the stack alphabet, m is the number of states of $\mathcal{A}$, $\{1, \ldots, n\}$ is the range of the priority function Ω, and c is a constant. The task of establishing existence of a winning strategy in $\mathcal{M}_{\mathcal{A}}$ 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, 23] we obtain that the whole problem can be solved in time $\mathcal{O}((k2^{cmn})^n)$.

Corollary 29 Suppose we have a pushdown automaton $\mathcal{A}$ with a partition of its states Q_I, Q_{II} and a function Ω defining the priorities of the states. These define a pushdown game $G_{\mathcal{A}}$. We can establish the winner in $G_{\mathcal{A}}$ in time $\mathcal{O}((k2^{cmn})^n)$ where k is the size of the stack alphabet, m the number of states of $\mathcal{A}$, and n is the size of the range of Ω.

It may be interesting to note that taking other winning conditions than parity conditions does not essentially change the complexity of deciding the winner. To translate, the most general, Muller conditions to parity conditions we need to use LAR's [16, 26]. These are of size $2^{m\log(m)}$. These LAR's will take place of priorities p in $\mathit{Check}(\vec{A}, z, \theta, p, q)$ nodes of the above construction. Hence the size of $\mathcal{M}_{\mathcal{A}}$ will increase by the factor $2^{m\log(m)}$.

Corollary 29 gives 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 30 For a given automaton $\mathcal{A}$ with m states and k stack symbols and a formula φ of the size n_1 and of the alternation depth n_2 there is an algorithm deciding in time $\mathcal{O}((k2^{cmn_1n_2})^{n_2})$ 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 obtained 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 wqw' is of length n.

For a given word w we construct a pushdown automaton $\mathcal{A}$ with the states partitioned into Q_I, Q_{II}, such that player I has a strategy to reach a leaf in the game $\mathcal{T}_\mathcal{A}$ iff w is accepted by M.

Let us describe the behaviour of $\mathcal{A}$. Initially, being in states from Q_I, it guesses n letters from $Q \cup \Gamma$ and pushes them on the stack. Among them there should be exactly one letter $q \in Q$. The automaton remembers this letter and the letter exactly after it. After pushing these n letters, $\mathcal{A}$ arrives at a state form Q_I, if q is an existential state, and at a state from Q_{II} otherwise. In this state, $\mathcal{A}$ consults the transition function of $\mathcal{M}$ and has a choice of pushing on the stack one of the triples describing a legal move of M. At this point the whole process repeats and $\mathcal{A}$ pushes new n elements on the stack. The automaton finishes this first stage when in the last cycle of pushing letters it pushed an accepting state. At this point $\mathcal{A}$ goes to a state *Check* $\in Q_{II}$ and on the stack we have a sequence:

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

where $c_0, \dots, c_k$ are configurations of M and $\{(q_i, a_i, d_i)\}_{i=1,\dots,k}$ are moves of M.

From this position there will be a strategy for player I iff this sequence is a valid sequence of configurations for the choices $\{(q_i, a_i, d_i)\}_{i=1,\dots,k}$ of M. In the state *Check* player II can decide to either check that c_k comes from c_{k-1} in the move (q_k, a_k, d_k), or to take c_k from the stack without checking and check some configurations below. If finally player II decides to check that some c_i comes from c_{i-1} in the move (q_i, a_i, d_i) then it enters a state Check_1. In this state player II can either check that the last letter of c_i is correct or it can take the letter from the stack and choose some subsequent letter. To check that some letter of c_i is correct the automaton remembers this letter in its finite control, takes $n+1$ letters from stack (remembering $(q_i, a_i, d_i$ on the way). Then it remembers the next four letters from the stack and checks the consistency of the five letters it remembers with the move (q_i, a_i, d_i). If the test succeeds $\mathcal{A}$ stops, otherwise it goes into an infinite loop.

If the play reaches the configuration c_0 without player II deciding to check consistency two consecutive configurations then player II checks whether the first configuration is an initial one. If the test succeeds $\mathcal{A}$ stops, otherwise it goes into an infinite loop.

It is not difficult to construct a polynomial size automaton $\mathcal{A}$ defining the game described above.

Proposition 31 There exists a formula α (without alternations) such that the problem "given a pushdown automaton $\mathcal{A}$, is α satisfied in the root of $\mathcal{T}_{\mathcal{A}}$" is DEXPTIME-hard. (Formula α expresses the fact that player I can reach a final state no matter what player II does.)

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 the size n_1 and the 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 $\mathcal{T}_{\mathcal{A}}$" is DEXPTIME hard.

Acknowledgements

I would like to thank Damian Niwinski for his helpful comments. I am grateful to the anonymous referees for their valuable remarks.

References

[1] J. Bergstra and J. Klop. Process theory based on bisimulation semantics. In de Bakker, de Rover, and Rozemberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, 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 R. 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] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graphs and unfoldings of transition systems. University of Aarhus BRICS report RS-95-44. Presented at CSL'95, To appear in Jounal of Pure and Applied Logic.

[8] E. A. Emerson. Automata, tableaux and temporal logic. In *Colledge Conference on Logic of Programs*, volume 193 of *LNCS*. Springer-Verlag, 1985.

[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*, pages 368–377, 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] Y. Gurevich and L. Harrington. Trees, automata and games. In *14th Symp. on Theory of Computations, ACM*, pages 60–65, 1982.

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

[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. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, editor, *Fifth Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.

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

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

[19] A. Nerode, A. Yakhnis, and S. Yakhnis. Concurent programs as strategies in games. In Y. Moschovakis, editor, *Logic from Computer Science*. Springer-Verlag, 1992.

[20] D. Niwiński. Fixed point characterization of infinite behaviour of finite state systems. *Theoretical Computer Science*, 189:1–69, 1997.

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

[22] S. Seibert. *Effektive Strategiekonstruktion für Gale-Stewart-Spiele auf Transitions Graphen*. PhD thesis, Kiel University, July 1996.

[23] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996.

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

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

[26] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997.