Pushdown Processes: Games and Model Checking

Igor Walukiewicz
BRICS[1,2]
Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Aarhus C, Denmark
e-mail: igw@mimuw.edu.pl

(Extended abstract)

Abstract

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

1 Introduction

Pushdown processes are, at least in this paper, just another name for pushdown automata. The different name is used to underline the fact that we are mainly interested in the graph of configurations of a pushdown process and not in the language it recognises. This graph can be considered as a transition system. In general such a transition system may not be regular, i.e., may not be an unwinding of a finite transition system. Given a priority function mapping states of the automaton to natural numbers, such a transition system defines a two player parity game. In the game moves of the players alternate. In a move a player picks a configuration reachable from the current one. The result of a game is a finite or an infinite path. The path is finite if one of the players cannot make a move; in this case the other player wins. If the path is infinite we find the smallest priority such that a state of this priority appears infinitely often on the path. Player I wins if this priority is even.

[1]Basic Research in Computer Science,Centre of the Danish National Research Foundation.
[2]On leave from: Institute of Informatics, Warsaw University,
Banacha 2, 02-097 Warsaw, POLAND

Pushdown processes are a generalisation of regular process which correspond to finite automata or regular transition systems. It is stated in [6] that the extra expressive power of pushdown processes may be of use for describing hierarchically structured systems, such as multi-level cashes, or wide area networks. Considering pushdown games is interesting at least for two reasons. First, as we will show here, there is a connection with model checking. The second reason is the problem of synthesis of correct programs (see for example [11]). The conditions of a game may be seen as a specification, and the two players as a program and environment respectively. In this approach a winning strategy is identified with a reactive program satisfying the specification. Hence it is important to know whether a strategy can be implemented as, for example, a regular or pushdown process.

Pushdown processes are a strict generalisation of processes from so called basic process algebra BPA (see [5] for a short survey). The decidability of the model checking for pushdown processes and the propositional μ-calculus follows from [14]. An elementary model checking procedure for the alternation free fragment of the μ-calculus was given in [3]. We are not aware of any such elementary decision procedure for the whole μ-calculus. BPA is a subclass of process algebra (PA) [1]. For the other interesting subclass of PA, namely, basic parallel processes, the model checking problem is undecidable [9]. The question whether pushdown games have pushdown strategies was posed in [16].

The main results of this paper are the following.

1. We show that if there is a winning strategy in a pushdown game G then there is a pushdown winning strategy in G.

2. We give a model checking algorithm for pushdown processes and the whole μ-calculus which runs in time $\mathcal{O}(2^{cn^3m})$ where m is the size of a pushdown process, n the size of a formula and c is some constant.

3. We show that there exists a formula α such that the model checking problem for pushdown processes and this particular formula α is DEXPTIME–hard.

Let us mention that the restriction to parity games is not essential for the result 1 to hold. One can use standard methods of translating Muller, Rabin or Streett conditions into parity conditions to obtain appropriate result for these kind of conditions.

The plan of the paper is as follows. We start with a preliminary section where we recall definitions of pushdown automata and the propositional μ-calculus. In the following section we present some facts about games with parity conditions. Next 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. The proofs are omitted in this abstract.

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

2 Preliminaries

2.1 Pushdown processes

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

For a given finite set Σ_s, let $Com(\Sigma_s) = \{pop\} \cup \{push(z) : z \in \Sigma_s\}$ be the set of *stack commands* over Σ_s.

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

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

where Q is a finite set of *states* and Σ_s is 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 $(z, q) \to (z', q')$ if $(z', q') \in \delta(z, q)$. 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.

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

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

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

Remark: In our definition of a pushdown automaton we have assumed that the automaton can put at most one symbol on the stack in one move. This is done only for convenience of the presentation. The main results also hold for the more general form of automata which can push many symbols on the stack in one move. Of course 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. (We are not interested in the language equivalence but in isomorphism of induced pushdown trees.)

Remark: The assumption that automata do not have an input alphabet is not essential as in the problems we will consider we will allow states to have "properties" which can be used to simulate behaviour of an automaton with input alphabet.

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 $\rho : Prop \to \mathcal{P}(S)$ is a function assigning to each propositional constant a set of states where this constant holds.

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

$$\| p \|_V^{\mathcal{M}} = \rho(p) \qquad \| \neg p \|_V^{\mathcal{M}} = S - \rho(p) \qquad \| X \|_V^{\mathcal{M}} = V(X)$$

$$\| \alpha \vee \beta \|_V^{\mathcal{M}} = \| \alpha \|_V^{\mathcal{M}} \cup \| \beta \|_V^{\mathcal{M}} \qquad \| \alpha \wedge \beta \|_V^{\mathcal{M}} = \| \alpha \|_V^{\mathcal{M}} \cap \| \beta \|_V^{\mathcal{M}}$$

$$\| \langle\,\rangle \alpha \|_V^{\mathcal{M}} = \{ s : \exists s'.R(s,s') \wedge s' \in \| \alpha \|_V^{\mathcal{M}} \}$$

$$\| [\,] \alpha \|_V^{\mathcal{M}} = \{ s : \forall s'.R(s,s') \Rightarrow s' \in \| \alpha \|_V^{\mathcal{M}} \}$$

$$\| \mu X.\alpha(X) \|_V^{\mathcal{M}} = \bigcap \{ S' \subseteq S : \| \alpha \|_{V[S'/X]}^{\mathcal{M}} \subseteq S' \}$$

$$\| \nu X.\alpha(X) \|_V^{\mathcal{M}} = \bigcup \{ S' \subseteq S : S' \subseteq \| \alpha \|_{V[S'/X]}^{\mathcal{M}} \}$$

here $V[S'/X]$ is the assignment such that, $V[S'/X](X) = S'$ and $V[S'/X](Y) = V(Y)$ for $Y \neq X$. We will write $\mathcal{M}, s, V \models \varphi$ when $s \in \| \varphi \|_V^{\mathcal{M}}$. We will write $\mathcal{M}, s \models \varphi$ if for every assignment V we have $\mathcal{M}, s, V \models \varphi$.

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

3 Parity games and canonical strategies

In this section we recall the notion of parity games and give an explicit description of winning strategies in such games. It turns out that a strategy in such a game induces an assignment of tuples of ordinals to nodes of the game. We call these tuples of ordinals *signatures*. In this way we have means to compare different strategies by comparing signatures they induce. It turns out that there exists canonical, or the least possible, signature assignment.

Most of the material presented here comes from [17]. The notion of signature was proposed by Streett and Emerson [15]. The proof of the existence of memoryless strategies in parity games was given independently by Mostowski [13] and by Emerson and Jutla [7]. Klarlund [10] proves a more general fact that a player has a memoryless strategy in a game if the has a strategy and his winning conditions are given as a Rabin condition.

Let $G = \langle V = V_I \cup V_{II}, E, \Omega : V \to Ind \rangle$ be a bipartite graph with vertices labeled by *priorities* from *Ind* which is a finite subset of natural numbers. A game from some vertex $v_1 \in V_I$ is played as follows: first player I chooses a vertex $v_2 \in V_{II}$, s.t. $E(v_1, v_2)$ then player II chooses a vertex $v_3 \in V_I$, s.t. $E(v_2, v_3)$ and so on ad infinitum unless one of the players cannot make a move. If a player cannot make a move he looses. The result of an infinite play is an infinite path $v_1, v_2, v_3, \ldots$ This path is *winning* for player I if in the sequence $\Omega(v_1), \Omega(v_2), \Omega(v_3), \ldots$ the smallest priority appearing infinitely often is even. The play from vertices of V_{II} is defined similarly but this time player II starts.

Strategy σ for player I is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex from V_I a vertex $\sigma(\vec{v}) \in V_{II}$ such that $E(v, \sigma(\vec{v}))$. A strategy is called *memoryless* iff $\sigma(\vec{v}) = \sigma(\vec{v}')$ whenever $\vec{v}$ and $\vec{v}'$ 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.

Suppose we have a propositional constant I which holds in the vertices from which player I is to move, i.e., in vertices from V_I. Let us assume that the range of Ω is $\{1, \ldots, n\}$ and suppose that for every $i \in \{1, \ldots, n\}$ we have the propositional constant i which holds in the vertices of priority i. Consider the formula:

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

We will be interested in the set:

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

(μ is used to close variables with odd indices and ν is used for even indices).

Definition 2 When applied to n-tuples of ordinals symbols $=$, $<$, $\leq$ stand for corresponding relations in the lexicographical ordering. For every $i \in \{1, \ldots, n\}$ we use $=_i$ to mean that both arguments are defined and when truncated to first i positions the two vectors are equal; similarly for $<_i$ and $\leq_i$.

Definition 3 (Signature) A signature is a n-tuple of ordinals. An assignment $\mathcal{S}$ of signatures to nodes in some set $S \subseteq T$ will be called *consistent* if for every $v \in S \cap V_I$ there is a son $w \in S$ such that:

$$\mathcal{S}(w) \leq_{\Omega(v)} \mathcal{S}(v) \text{ and it is strictly smaller if } \Omega(v) \text{ is odd.} \tag{2}$$

similarly if $v \in S \cap V_{II}$ then for all w such that $E(v, w)$ we have $w \in S$ and the condition (2) holds.

Definition 4 (Canonical signatures) 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 $\| \mu Z.\alpha(Z) \|_V^{\mathcal{M}} = \bigcup_\tau \| \mu^\tau Z.\alpha(Z) \|_V^{\mathcal{M}}$.

We define a notion of the *canonical signature*, $Sig(v)$, of a vertex $v \in \mathcal{S}_I$ (we will write $Sig_G(v)$ if the game is not clear from the context). This is the smallest in the lexicographical ordering sequence of ordinals $(\tau_1, \ldots, \tau_n)$ such that:

$$v \in \| \varphi_I(P_0, \ldots, P_n) \|^G$$

where:

$$P_i = \nu Z_i.\mu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_0, \ldots, P_{i-1}, Z_i, \ldots, Z_n) \quad \text{for } i \text{ even}$$
$$P_i = \mu^{\tau_i} Z_i.\nu Z_{i+1} \ldots \nu Z_n.\varphi_I(P_0, \ldots, P_{i-1}, Z_i, \ldots, Z_n) \quad \text{for } i \text{ odd}$$

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 5 Canonical signature assignment is the least consistent signature assignment. That is, for every consistent signature assignment $\mathcal{S}$, whenever for some node v, $\mathcal{S}(v)$ is defined then $Sig(v)$ is defined and $Sig(v) \leq \mathcal{S}(v)$.

Definition 6 (Canonical strategy) A *canonical strategy* is a strategy taking for each node $v \in \mathcal{W}_I \cap V_I$ a son which has the smallest possible canonical signature.

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

Fact 7 Suppose w is a node reached from v when player I uses a canonical strategy and let p be the minimum of priorities of states appearing in the labels between v and w (not including w). We have that $Sig(w) \leq_p Sig(v)$ and it is strictly smaller if p is odd.

Theorem 8
The set $\mathcal{W}_I$ is the set of nodes from which player I has a winning strategy. A canonical strategy is winning and memoryless.

4 Pushdown strategies in pushdown games

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

The automaton $\mathcal{A}$ defines a pushdown tree $T_{\mathcal{A}}$ which we will take as a graph of the game. To have a game we will also need a priority function. It is an important point to decide which priority functions to allow. If we allowed arbitrary such

functions then the whole advantage of the fact that the graph is generated by a pushdown automaton would be gone. It seems that a reasonable choice is to allow only functions associated with states of the automaton. That is, we start by giving a priority function $\Omega : Q \to \mathbb{N}$ and then for every vertex v of T we consider the state q appearing in the label of v and let $\Omega(v) = \Omega(q)$. This choice of the method of assigning priorities is motivated by the fact that we are interested in the winning conditions definable in S1S.

Next we should clarify what we mean by a pushdown strategy. We would like to say that a pushdown strategy is a strategy realised by a pushdown automaton in a sense that this automaton reads moves of player *II* and outputs moves of player *I*. Such an automaton must have the property that while reading a (possibly infinite) sequence of moves of player *II* it outputs a sequence of moves of player *I* such that the path of $T_{\mathcal{A}}$ designated by these moves is winning for player *I*. We will not formalise this notion of pushdown strategy here as it would require several definitions for which we have no space. We will content ourselves with a weaker definition given in the theorem below. Let us just remark that the strategy automaton given in the proof can be used to construct a strategy automaton as defined above.

Theorem 9
If there is a winning strategy for player I in $T_{\mathcal{A}}$ then there is a winning pushdown strategy, i.e., there is a pushdown automaton $\mathcal{B}$ such that $T_{\mathcal{B}}$ is isomorphic to a winning strategy in $T_{\mathcal{A}}$.

Let us try to explain an idea of the construction of the pushdown strategy for player *I*. In some sense one may consider a pushdown strategy as a strategy operating with a stack of strategies for regular graphs. Whenever a new element is pushed on a stack, player *I* is suspended and a new player *I* is started which has only partial information about the history of the play up to this point. Suppose we are in some position (s, q) and player *I* decides that the best move for him would be to push z' on the stack and change the state to q'. At this moment this player *I* is suspended and a new player *I* starts to play. He will play until z' is taken out from the stack. The main question is what the new player *I* should know about the current position of the play. Because the canonical strategy is memoryless it would be enough for him to know only how the arena of the game looks from his current position. In turn this is determined by the label of the node, which is (sz', q'). Unfortunately we cannot afford to let the player know so much because the size of the stack is potentially unbounded. On the other hand the new payer *I* will play only until z' is popped and the stack becomes s again. Hence the part of the tree where the new player *I* is playing does not depend on s but only on the letter z' on the top of the stack and the current state q'. What depends on s is the rest of the play when the new player is finished. Hence it should be enough for the new player *I* if the old player *I* told him which states are safe. In other words what are the states such that if the new player *I* finishes in one of them then the old player *I* is able to carry on and win. This set of states should depend on the lowest priority of a state met

from the moment the old player I was suspended. So we will not have just one set of states but a vector $\vec{A} = \{A^p\}_{p\in\{1,\ldots,n\}}$ of sets of states. Each A^p is a set of states in which the new player I can finish provided p is the smallest priority of a state from the moment when the old player I was suspended. Apart from $\vec{A}$ the new player should also know the current state q' and the current symbol z' on the top of the stack. We will also use a variable θ to store the lowest priority of a state we came across. This amount of information is bounded so we have a basis for construction of a pushdown automaton realizing the strategy.

Let us now start with the formal definitions. As in the previous section we assume that $\{1,\ldots,n\}$ is the range of Ω. We will use $\vec{A}$ to range over n element vectors of sets of states and θ to range over $\{1,\ldots,n\}$. We also use z to range over stack symbols and q to range over states.

Definition 10 (Sub-game) For every $\vec{A}, z, \theta, q$ we define the game $G(\vec{A}, z, \theta, q)$ as follows. The arena of the game is a subtree of $T_{\mathcal{A}}$ starting from a node with a configuration $(\bot z, q)$. Every node labeled with a configuration $(\bot, q')$, for some q', is marked winning or loosing. We mark the node *winning* if $q' \in A^{\min(p,\theta)}$, where p is the lowest priority of a state appearing on the path to the node (counting q but not q'). Otherwise we mark the node loosing. Whenever a play reaches a marked node, player I wins if this node is marked winning otherwise player II is the winner. If a play is infinite, player I wins iff the obtained path is winning (as defined at the beginning of Section 3).

Remark: In our definition of the game we did not have the concept of marking but we allowed vertices with no sons, and had the rule that a 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.

Definition 11 (Signature, Hint) Suppose that player I has a winning strategy in a game $G(\vec{A}, z, \theta, q)$. Define $Sig(\vec{A}, z, \theta, q)$ to be the canonical signature of the root of the game.

If $q \in Q_I$ then let v be a son of the root which has the smallest canonical signature (if there is more than one such son then fix one arbitrary). If v is labeled by $(\bot, q')$ then let $Hint(\vec{A}, z, \theta, q) = (pop, q')$ otherwise v is labeled by $(\bot z z', q')$ and let $Hint(\vec{A}, z, \theta, q) = (push(z'), q')$.

Definition 12 (Update function) Define $Up(\vec{A}, z, q, \theta)$ to be the sequence of sets $\vec{A}_1 = \{A_1^p\}_{p\in\{1,\ldots,n\}}$, where each A_1^p is the set of states q' such that:

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

in 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.

Definition 13 (Strategy automaton) Let

$$\mathcal{B} = \langle Q, \mathcal{P}(Q)^n \times \Sigma_s \times \{1, \ldots, n\}, q_0, (\emptyset, \ldots, \emptyset, \bot, n), \delta_\beta \rangle$$

Before defining the transition relation let us introduce an abbreviation. We introduce new automata operation $repmin(\theta')$ which means: if on the top of the stack there is some triple $\vec{A}z\theta$, replace it with $\vec{A}z\theta_1$ where $\theta_1 = \min(\theta, \theta')$. We also introduce a semicolon operation, so $\delta_\mathcal{B}(\vec{A}z\theta, q, a) = (pop, q'); repmin(\theta')$ means that first $\vec{A}z\theta$ is removed from the stack and the state is changed to q'; then, possibly, the third component of the triple currently at the top of the stack is changed. Hence if we had a configuration $(s\vec{A}_1 z_1 \theta_1 \vec{A}z\theta, q)$ then after this operation we obtain the configuration $(s\vec{A}_1 z_1 \min(\theta_1, \theta'), q')$.

Let us now proceed with the definition of $\delta_\mathcal{B}$:

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

Theorem 9 follows from the following lemmas:

Lemma 14 If player I can win in $G(\vec{A}, z, \theta, q)$ and

$$repmin(\Omega(q)); push(\vec{A}_1 z_1 n, q_1) \in \delta_\mathcal{B}(\vec{A}z\theta, q)$$

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.

Lemma 15 Let $(sz\vec{A}\theta, q)$ be a configuration reachable from the initial one. Suppose $(sz\vec{A}\theta, q) \rightarrow^+ (sz\vec{A}\theta', q')$ and $sz\vec{A}$ is always in the stack during this derivation. Let p be the minimum of the priorities of the states appearing in the derivation (not counting the last one). We have: (i) $\theta' = \min(p, \theta)$ and (ii) $Sig(\vec{A}, z, \theta', q') \leq_p Sig(\vec{A}, z, \theta, q)$ and it is strictly smaller if p is odd.

Lemma 16 The strategy defined by the automaton $\mathcal{B}$ is winning.

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

5 Model checking for pushdown trees

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

The reduction of the model checking to establishing existence of a winning strategy follows from a fairly standard arguments [8]. In that paper Emerson, Jutla and Sistla show how to reduce the model checking problem over finite transition systems to establishing existence of a winning strategy in a finite game. In our case the argument is essentially the same but we must also observe that in the resulting game the priority function Ω depends only on the states in the current configuration.

Theorem 17
For a given pushdown automaton $\mathcal{A}$ and a μ-calculus formula φ one can construct a pushdown automaton $\mathcal{C}$ and a priority function Ω, such that: $T_{\mathcal{A}} \models \varphi$ iff there is a winning strategy for player I in the game $T_{\mathcal{C}}$ with the priority function Ω. The size of $\mathcal{C}$ is linear in sizes of both $\mathcal{A}$ and φ.

5.1 Establishing existence of winning strategies

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

Before we begin let us try to give some intuitions behind the construction of a finite game $\mathcal{M}_{\mathcal{A}}$. For every $\vec{A}, z, \theta, q$ and $p \in \{1, \ldots, n\}$ we will have in $\mathcal{M}_{\mathcal{A}}$ a node $Check(\vec{A}, z, \theta, p, q)$. There will be strategy for player I from this node iff there is a strategy for player I in the game $G(\vec{A}, z, \theta, q)$ (see Definition 10); we will explain the role of p later. If $pop(q')$ move is possible from (z, q) then for it to be a good choice for player I it should be the case that $q' \in A^{\min(\Omega(q),\theta)}$. If $(push(z_1), q_1)$ is possible then the checking is more complicated as we do not have a stack. We will use universal branching instead. We will have a node $Move((\vec{A}, z, \theta, q), (?, z_1, q_1))$ with the intended meaning that the next planed move is $(push(z_1), q_1)$ and that one has to guess $\vec{A}_1$. We will also have nodes $Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))$ where $\vec{A}_1$ is already guessed and from which it is necessary to check whether it was guessed correctly. We divide the future play into two parts which we consider separately. We check what happens until z_1 is popped from the stack and simultaneously we check what happens after this event. The first task is started from the node $Push(\vec{A}_1, z_1, n, q_1)$ the

other one from nodes $Check(\vec{A}, z, \min(\theta, p''), p'', q'')$ where p'' intuitively represents the lowest priority which was met while z_1 was on the stack and q'' is a state from $A_1^{p''}$.

Definition 18 (Game $\mathcal{M}_\mathcal{A}$) Let $\mathcal{M}_\mathcal{A}$ be a game on a finite graph defined as follows. For every $\vec{A}, \vec{A}_1$, z, z_1, θ, q, q_1 and $p \in \{1, \ldots, n\}$ we have nodes:

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

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

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

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

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

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

$$\begin{array}{l} Check(\vec{A}, z, \theta, p, q) \text{ and } Push(\vec{A}, z, \theta, q) \quad \text{for } q \in Q_I \\ Move((\vec{A}, z, \theta, q), (?, z_1, q_1)) \quad \text{for arbitrary } q \in Q_I \cup Q_{II} \end{array}$$

In the remaining nodes player II makes a move. Priority function Ω_M is defined by:

$$\Omega_M(Check(\vec{A}, z, \theta, p, q)) = p \qquad \Omega_M(Push(\vec{A}, z, \theta, q)) = \Omega(q)$$
$$\Omega_M(Move((\vec{A}, z, \theta, q), (?, z_1, q_1))) = \Omega_M(Move((\vec{A}, z, \theta, q), (\vec{A}_1, z_1, q_1))) = n + 1$$

Player I wins in the game $\mathcal{M}_\mathcal{A}$ if either: (i) after finitely many steps player II cannot make a move or a node labeled $Pop(q)$, for some q, is reached; or (ii) the game is infinite and the infinite path $\mathcal{P}$ which is the result of the play is winning for I. Otherwise player II is the winner.

Theorem 19
Player I has a winning strategy in the game $T_\mathcal{A}$ iff he has a winning strategy in the game $\mathcal{M}_\mathcal{A}$ from the node $Check((\emptyset, \ldots, \emptyset), \bot, n, n, q_0)$.

Let us remark here that the theorem does not imply that there is a finite strategy on a pushdown tree. In order to use the strategy in $\mathcal{M}_\mathcal{A}$ to play in $T_\mathcal{A}$ we need a stack.

Finally let us put Theorems 17 and 19 together and calculate the complexity of the model checking algorithm. The size of the game $\mathcal{M}_{\mathcal{A}}$ is $\mathcal{O}(k2^{cmn})$ where: k is the size of the stack alphabet, m is the number of states of $\mathcal{A}$, n is the cardinality of 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 whether the specific μ-calculus formula holds. Hence any model checking algorithm will solve the problem. Using currently known algorithms we obtain that the whole problem can be solved in time $\mathcal{O}((k2^{cmn})^n)$ (or $\mathcal{O}((k2^{cmn})^{1+n/2})$ if using [12]). This is the estimation only for the problem of establishing existence of a winning strategy. Putting it together with the reduction from the previous subsection we obtain that for a given automaton with m states and k stack symbols and a formula of size n_1 with alternation depth n_2 we have an algorithm working in time $\mathcal{O}((k2^{cmn_1n_2})^{n_2})$.

5.2 The lower bound

Finally we show a deterministic exponential time lower bound on the model checking problem for pushdown automata and (non alternating) μ-calculus. It follows from a quite standard reduction by simulating alternating linear space bounded Turing machines. The simulating automaton is very similar to the one described by Chandra, Kozen and Stockmeyer in [4]. Given an alternating linear space bounded machine M and a word w we construct a pushdown automaton which acts as follows. First it puts the initial configuration of M on the stack. If the initial state is existential, player I chooses which possible move of M to simulate, otherwise player II chooses the move. Simulating the move is done by putting a new configuration by player I on the stack. Proceeding this way, the game eventually arrives to a point when a configuration with an accepting state is pushed on the stack. At the same moment we have also all the preceding configurations on the stack. In this position player II is allowed to make a guess about correctness of this sequence of configurations. He may try to show that player I cheated and there are two subsequent configurations on the stack such that one is not reachable from the other in the move of M which was chosen at that point. Player I wins if player II is not able to do this. We have the following:

Fact 20 There exists a formula α (without alternations) such that the problem "given a pushdown automaton $\mathcal{A}$, is α satisfied in the root of $T_{\mathcal{A}}$" is DEXPTIME–hard.

Remark: This argument does not work for BPA processes. Indeed the complexity result from [2] shows that the the model checking problem for the μ-calculus without alternations is polynomial when a formula is fixed.
Remark: We conjecture that model checking is exponential also in the second parameter. That is, there exists a fixed pushdown process $\mathcal{A}$ such that the problem: "given a formula α, is α satisfied in the root of $T_{\mathcal{A}}$" is DEXPTIME–hard.

References

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

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

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

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

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

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

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

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

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

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

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