Marking shortest paths on pushdown graphs does not preserve MSO decidability

Arnaud Carayol, Olivier Serre

Marking Shortest Paths On Pushdown Graphs Does Not Preserve MSO Decidability

Arnaud Carayol*[1] and Olivier Serre†[2]

[1]LIGM (CNRS & Université Paris Est)
[2]IRIF (CNRS & Université Paris Diderot – Paris 7)

Abstract

In this paper we consider pushdown graphs, *i.e.* infinite graphs that can be described as transition graphs of deterministic real-time pushdown automata. We consider the case where some vertices are designated as being final and we build, in a breadth-first manner, a marking of edges that lead to such vertices (*i.e.*, for every vertex that can reach a final one, we mark all out-going edges laying on some shortest path to a final vertex).

Our main result is that the edge-marked version of a pushdown graph may itself no longer be a pushdown graph, as we prove that the MSO theory of this enriched graph may be undecidable.

*Arnaud.Carayol@univ-mlv.fr
†Olivier.Serre@cnrs.fr

1 Introduction

The original motivation of this paper comes from the following work plan: design algorithms working on *infinite* graphs and computing classical objects from graph theory. One firstly targeted set of algorithms are naturally those computing spanning trees, *e.g.* spanning trees built by performing a breadth-first search or the ones built by performing a depth-first search. In particular, breadth-first search seems to be a good candidate as it can easily be defined by a smallest fixpoint computation.

Of course, to ensure termination one has to identify reasonable classes of infinite graphs: obviously such a class should provide finite description of its elements and the graphs should have some good decidability properties. The simplest such class is the class of transition graphs of pushdown automata: they are finitely described by the underlying pushdown automata and they enjoy many good properties, in particular with respect to logic and games (see *e.g.* [Muller and Schupp(1985), Walukiewicz(2001)]). In particular, monadic second-order logic (MSO) is decidable for any pushdown graph.

Another expected property of our algorithm is that it should be reflective[1] in the following sense: the produced outputs should belong to the same class of structures as the inputs. Equivalently, in the setting of pushdown graphs, it means that we want to design an algorithm that takes as an input a pushdown graph and produces as an output another pushdown graph that is an isomorphic copy of the input graph enriched with a marking of some edges that corresponds to a breadth-first search spanning tree.

The main result of this paper is that such an algorithm does not exist, *i.e.* there is no algorithm that takes as an input a pushdown graphs and returns a copy of it marked with a breadth-first search spanning tree. The roadmap to prove this result is to exhibit a pushdown graph such that when marked with a breadth-first search spanning tree leads to a graph with an undecidable MSO theory: as pushdown graphs enjoy decidable MSO theories it directly permits to conclude.

The paper starts by introducing in Section 2 the classical objects and formally defines the problem under study. Our main results are proven in Section 3 while we briefly discuss some consequences in Section 4.

2 Preliminaries

An ***alphabet*** A is a finite set of letters. In the sequel A^* denotes the set of finite words over A and the ***empty word*** is written ε. The length of

[1] In programming languages, reflection is the process by which a computer program can observe and dynamically modify its own structure and behaviour. See [Broadbent et al.(2010)Broadbent, Carayol, Ong, and Serre] for an example of reflection in the richer setting of collapsible pushdown automata and recursion schemes.

a word u is denoted by $|u|$ and for any $k \geqslant 0$, we let $A^{\leqslant k} = \{u \mid |u| \leqslant k\}$. Let u and v be two finite words. Then $u \cdot v$ (or simply uv) denotes the ***concatenation*** of u and v.

Let A be an alphabet. An A-labeled (oriented) ***graph*** is a pair $G = (V, E)$ where V is a (possibly infinite) set of vertices and $E \subseteq V \times A \times V$ is a (possibly infinite) set of edges. In the sequel we write $v \xrightarrow{a} v'$ to denote that $(v, a, v') \in E$.

A vertex v' is ***reachable*** from a vertex v if there is a sequence $v_1, \ldots, v_\ell$ of vertices together with a sequence of letters $a_1, \ldots, a_{\ell-1}$ such that $v_1 = v$, $v_\ell = v'$ and $v_i \xrightarrow{a_i} v_{i+1}$ for every $i = 1, \ldots, \ell - 1$.

2.1 Pushdown Graphs

A ***deterministic real-time pushdown automaton*** is defined as a tuple $\mathcal{P} = (Q, A, \Gamma, \bot, q_{in}, Q_{fin}, \delta)$ where Q is a finite set of control states, A is a finite input alphabet, Γ is a finite stack alphabet, $\bot \in \Gamma$ is a bottom-of-stack symbol, $q_{in} \in Q$ is an initial state, $Q_{fin} \subseteq Q$ is a set of final states and $\delta : Q \times \Gamma \times A \to Q \times \Gamma^{\leqslant 2}$ is a *partial* transition function such that

- $\delta(q, \bot, a) = (q', u) \Rightarrow u \in (\Gamma \backslash \{\bot\})\bot \cup \{\bot\}$, ***i.e.*** $\bot$ cannot be removed.
- $\delta(q, \gamma, a) = (q', u)$ and $\gamma \neq \bot \Rightarrow u \in (\Gamma \backslash \{\bot\})^{\leqslant 2}$, ***i.e.*** $\bot$ cannot be pushed.

A ***configuration*** of $\mathcal{P}$ is a pair $(q, \sigma) \in Q \times (\Gamma \backslash \{\bot\})^* \bot$ consisting of a control state and a well-formed stack content. The ***initial configuration*** of $\mathcal{P}$ is $(q_{in}, \bot)$ and the ***final configurations*** of $\mathcal{P}$ are those of the form $(q_{fin}, \bot)$ with $q_{fin} \in Q_{fin}$.

Let (q, σ) and (q', σ') be two configurations, and let $a \in A$ be a letter. Then, there is an ***a-labelled transition*** from (q, σ) to (q', σ'), denoted $(q, \sigma) \xrightarrow{a} (q', \sigma')$, if and only if one has $\delta(q, \gamma, a) = (q', u)$ where $\sigma = \gamma\sigma''$ and $\sigma' = u\sigma''$, ***i.e.*** σ' is obtained from σ by replacing its top symbol γ by u.

The ***configuration graph*** of $\mathcal{P}$ is the A-labeled graph $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ where $V_{\mathcal{P}}$ is the set of configurations of $\mathcal{P}$ and where $E_{\mathcal{P}}$ is the transition relation defined by $\mathcal{P}$. A graph isomorphic to a graph $G_{\mathcal{P}}$ is called a ***pushdown graph***.

Example 1. *As a running example, consider the following pushdown automaton $\mathcal{P} = (Q, A, \Gamma, \bot, q_{in}, \{q_{fin}\}, \delta)$ where one lets $Q = \{q_{in}, q_{fin}, q_\sharp\}$, $A = \{a, b, \sharp\}$, $\Gamma = \{a, b, \bot\}$ and δ be as follows:*

- *$\delta(q_{in}, \gamma, a) = (q_{in}, a\gamma)$ and $\delta(q_{in}, \gamma, b) = (q_{in}, b\gamma)$: in the initial state on reading a symbol in $\{a, b\}$ it is copied on top of the stack.*
- *$\delta(q_{in}, \gamma, \sharp) = (q_\sharp, \gamma)$: in the initial state on reading symbol $\sharp$ the state is switched to $q_\sharp$.*

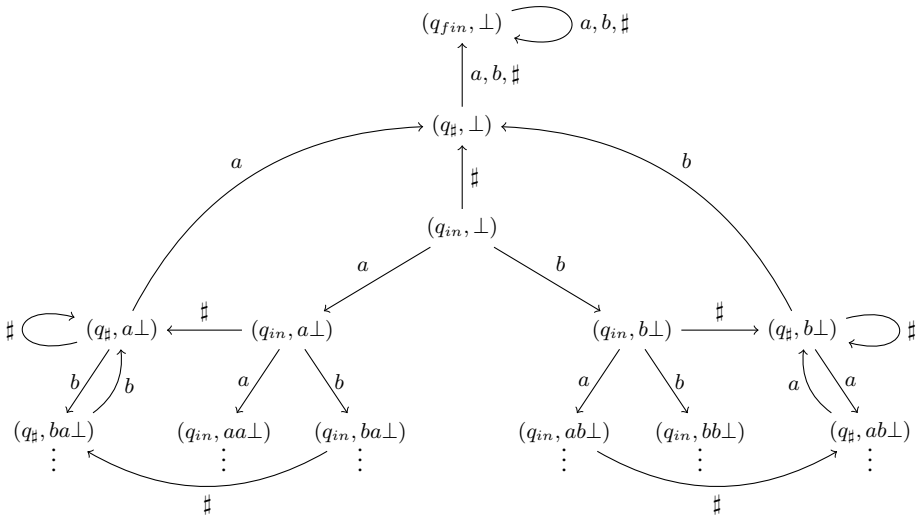

Figure 1: The configuration graph $G_{\mathcal{P}}$ of the pushdown automaton of Example 1

- *For $x \in \{a, b\}$ and $\gamma \neq \bot$, $\delta(q_\sharp, \gamma, x) = (q_\sharp, \varepsilon)$ if $\gamma = x$ and $\delta(q_\sharp, \gamma, x) = (q_\sharp, x\gamma)$ if $\gamma \neq x$; and $\delta(q_\sharp, \gamma, \sharp) = (q_\sharp, \gamma)$: in the state $q_\sharp$ an input letter $\sharp$ does not change the configuration while for an input letter in $\{a, b\}$ the top symbol is popped if it is the same as the input symbol otherwise the letter is copied on top of the stack.*
- *$\delta(q_\sharp, \bot, x) = (q_{fin}, \bot)$ for any $x \in A$: once the stack is emptied in the state $q_\sharp$ one goes to the state q_{fin}.*
- *$\delta(q_{fin}, \bot, x) = (q_{fin}, \bot)$ for any $x \in A$: once the configuration $(q_{fin}, \bot)$ is reached it stays in forever.*

The graph $G_{\mathcal{P}}$ is depicted in Figure 1.

We are interested in defining, in a breadth-first search manner, the set of configurations from which one can reach a final configuration. For this consider the following increasing sequence $(W_i)_{i \geqslant 0}$ of configurations of $\mathcal{P}$ and call its limit W.

- $W_0 = \{(q_{fin}, \bot) \mid q_{fin} \in Q_{fin}\}$ consists only of the final configurations.
- $W_{i+1} = W_i \cup \{(q, \sigma) \mid \exists (q', \sigma') \in W_i$ and $a \in A$ s.t. $(q, \sigma) \xrightarrow{a} (q', \sigma')\}$.

Obviously, W is the set of all configurations from which a final configuration is reachable. Define for every configuration (q, σ) its rank $rk((q, \sigma))$ to be the smallest i such that $(q, \sigma) \in W_i$ when exists and to be ∞ otherwise.

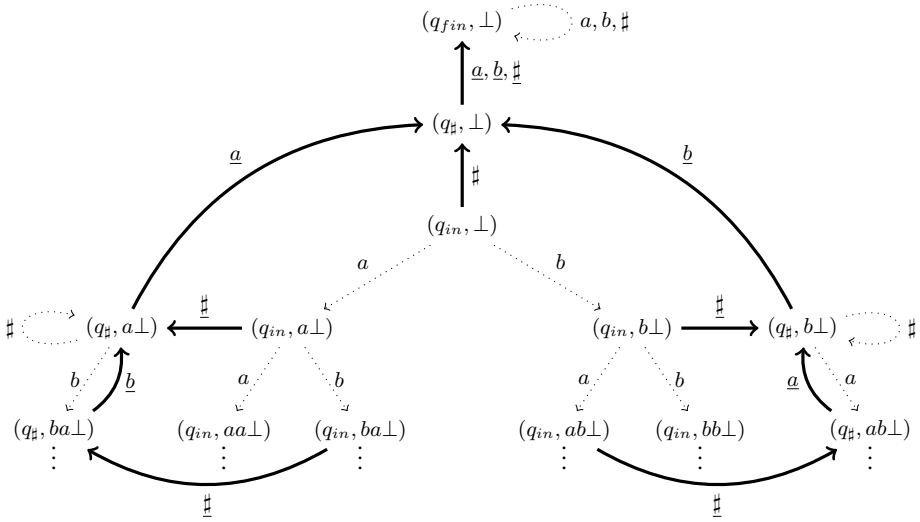

Figure 2: The graph $\widetilde{G}_{\mathcal{P}}$ of the pushdown automaton of Example 1

We now define a new graph $\widetilde{G}_{\mathcal{P}}$ obtained from $G_{\mathcal{P}}$ by marking those edges that go from a configuration to one with a strictly smaller rank (equivalently that decrease the rank by 1). First we let $\widetilde{A} = A \cup \{\underline{a} \mid a \in A\}$ consists of A together with a marked copy of each of its elements. Then we let $\widetilde{G}_{\mathcal{P}}$ be the $\widetilde{A}$-labelled graph $(V_{\mathcal{P}}, \widetilde{E}_{\mathcal{P}})$ where

- $((q,\sigma), a, (q',\sigma')) \in \widetilde{E}_{\mathcal{P}}$ if $((q,\sigma), a, (q',\sigma')) \in E_{\mathcal{P}}$ and $rk((q',\sigma')) \geqslant rk((q,\sigma))$;
- $((q,\sigma), \underline{a}, (q',\sigma')) \in \widetilde{E}_{\mathcal{P}}$ if $((q,\sigma), a, (q',\sigma')) \in E_{\mathcal{P}}$ and $rk((q',\sigma')) < rk((q,\sigma))$.

Coming back to Example 1, the graph $\widetilde{G}_{\mathcal{P}}$ is depicted in Figure 2.

Finally, one can consider a graph built out of $G_{\mathcal{P}}$ by marking only some (but at least one) shortest paths to a final configuration (the extreme case being when the marked paths form a spanning tree). More precisely, a ***well-formed marking*** of $G_{\mathcal{P}}$ is an $\widetilde{A}$-labelled graph $G = (V_{\mathcal{P}}, E)$ such that:

- For every edge $((q,\sigma), \underline{a}, (q',\sigma')) \in E_{\mathcal{P}}$, one has either $((q,\sigma), a, (q',\sigma')) \in E$ or $((q,\sigma), \underline{a}, (q',\sigma')) \in E$.
- For every edge $((q,\sigma), a, (q',\sigma')) \in E_{\mathcal{P}}$, one has $((q,\sigma), a, (q',\sigma')) \in E$.
- For every edge $((q,\sigma), a, (q',\sigma')) \in E$, one has either $((q,\sigma), a, (q',\sigma')) \in E_{\mathcal{P}}$ or $((q,\sigma), \underline{a}, (q',\sigma')) \in E_{\mathcal{P}}$.
- For every edge $((q,\sigma), \underline{a}, (q',\sigma')) \in E$, one has $((q,\sigma), \underline{a}, (q',\sigma')) \in E_{\mathcal{P}}$.
- For every configuration $(q,\sigma) \in W$ one has at least one edge of the form $((q,\sigma), \underline{a}, (q',\sigma')) \in E$.

2.2 Monadic Second Order Logic

Monadic Second Order Logic (MSO) is a classical logical formalism (extending First Order Logic) to express properties of a relational structure. In this framework, formulas are built from atomic formulas using Boolean connectives (negation, disjunction, conjunction) and quantifiers. Atomic formulas in first-order logic have either the form $x_1 = x_2$ or $x_1 \xrightarrow{a} x_2$ where x_1 and x_2 are first-order variables (that each stands for an element in the domain, *i.e.* for a vertex); in monadic second-order logic there are additional atomic formulas of the form $x \in X$ where x is a first-order variable and X is a second-order variable (that stands for a subset of the domain, *i.e.* for a subset of vertices). First-order (resp. second-order) variables are introduced thanks to existential and universal quantifications.

Whether a formula (without free variable) holds in a structure is defined as usual, and to keep this article short we refer the reader *e.g.* to [Thomas(1997)] for examples and formal definitions. When a formula has free variables it permits to express whether a structure together with an assignation of the free variables satisfy the formula: in particular if one has a single first-order free variable, it permits to express a property of an element inside a structure.

We say that a structure has a ***decidable MSO theory*** in case the following problem is decidable: does the input formula φ hold in the structure?

The following is a classical result due to Muller and Schupp [Muller and Schupp(1985)]

Theorem 1. *For any pushdown automaton $\mathcal{P}$ the graph $G_{\mathcal{P}}$ has a decidable MSO theory.*

3 Main Results

We are now ready to state our first result.

Theorem 2. *There is a pushdown automaton $\mathcal{P}$ such that the graph $\widetilde{G}_{\mathcal{P}}$ has an undecidable MSO theory.*

Proof. The idea is to design a pushdown automaton $\mathcal{P}$ such that, for any 2-counter machine with tests for 0, one can build an MSO formula that is true in $\widetilde{G}_{\mathcal{P}}$ if and only if the 2-counter machine halts from its initial configuration. As the halting problem is undecidable for 2-counter machine with tests for 0, it follows that $\widetilde{G}_{\mathcal{P}}$ has an undecidable MSO theory.

We define a pushdown automaton over the stack alphabet $\Gamma = (\{1,2,3\} \times \{1,2,3\} \times \{2\}) \cup \{\bot\}$: with any element over $\Gamma^*\bot$ one can associate a triple (k_1, k_2, k_3) where k_i is obtained by taking the sum along the i-th component of the elements in the stack (ignoring $\bot$). With such a

triple (k_1, k_2, k_3) we can associate a pair of counters $(k_1 - k_3, k_2 - k_3) \in \mathbb{Z} \times \mathbb{Z}$. Of course several stack contents may encode the same counters values but this is not problematic later on.

The pushdown automaton $\mathcal{P}$ works in two modes (the mode being indicated by the control state; we omit the states in this description to help readability). In the first mode, one can push any symbol in $\{1, 2, 3\} \times \{1, 2, 3\} \times \{2\}$ (namely there is a dedicated input letter for any symbol to be pushed), which allows to increment/decrement/keep unchanged the two counters: *e.g.* to increment counter $k_1 - k_3$ and decrement the counter $k_2 - k_3$ one pushes $(3, 1, 2)$; more generally to modify the counter $k_1 - k_3$ by ι_1 and to modify the counter $k_2 - k_3$ by ι_2 one pushes $(2 + \iota_1, 2 + \iota_2, 2)$. In the first mode, one can switch to a second mode (thanks to a special input letter) that comes with three variants depending on the control state (call those states p_1, p_2 or p_3). In the state p_i one pops on the stack and the "popping speed" depends on the i-th component of the current top symbol: if the top stack symbol has 1 as its i-th component, one simply pops, if it has $x > 1$ one goes first to $x - 1$ intermediate states before popping. Hence, in a configuration with associated triple (k_1, k_2, k_3) it takes k_i steps before emptying the stack starting in the state p_i. Once the stack is emptied, the (unique) final state is reached (that is from a configuration with top symbol $\perp$ there is only one possible transition that goes to the final state and this is the only way to reach it).

Now, consider the graph $\widetilde{G}_{\mathcal{P}}$ and let us explain how the extra information carried by this graph permits to compare k_1, k_2 and k_3. This in turn will allow us to test if the counters $k_1 - k_3$ and $k_2 - k_3$ are equal to 0 when simulating a 2-counter machine. In the following, we refer to an edge in $\widetilde{G}_{\mathcal{P}}$ as being marked if it corresponds to an underlined input symbol.

Consider a configuration (in the first mode) with an associated triple (k_1, k_2, k_3) and assume that one wants to check whether $k_1 = k_3$. For this one looks at the marked outgoing edges in the present configuration: if there is one that goes to a configuration (in the second mode) with the state p_1 and another one that goes to a configuration with the state p_3 (remember that the marked edges indicate the fastest choices to the final configuration) one directly concludes that $k_1 = k_3$, hence that $k_1 - k_3 = 0$; however one may have $k_1 = k_3$ without being in the previous situation namely if $k_2 < k_1, k_3$. In this latter case the trick is to increase all three components: the k_1 and the k_3 components by the same value and the k_2 component by a much bigger value, leading to a configuration (k_1', k_2', k_3') with $k_2' > k_1', k_3'$ and $k_1' = k_3'$ if and only if $k_1 = k_3$. The latter can easily be achieved by performing a sequence of actions pushing $(2, 3, 2)$. Hence, if one wants to check whether $k_1 = k_3$ it suffices to check whether the following holds: "either the marked edges indicate to go both to p_1 and p_3 or there exists a path along which only pushing $(2, 3, 2)$ are performed

and that leads to a configuration where the marked edges indicate to go both to p_1 and p_3". As the latter can easily be stated in MSO logic it follows that one can write an MSO formula (with one order-1 free variable) that holds exactly in those configurations (in the first mode) where $k_1 = k_3$. Similarly, one can design a formula to check whether $k_2 = k_3$.

Now for any 2-counter machine with tests for 0 one can easily write an MSO formula that builds on the previous formulas and checks whether there is a run of the machine that is halting: the states of the machine are directly handled in the MSO formula while the counter is managed thanks to the underlying structure of $\widetilde{G}_{\mathcal{P}}$.

As the halting problem for 2-counter machine with tests for 0 is undecidable, it concludes the proof. □

We now state our second result which is an analog of Theorem 2 but for well-formed marking of $G_{\mathcal{P}}$ (hence, it encompasses Theorem 2 as $\widetilde{G}_{\mathcal{P}}$ is a well-formed marking of $G_{\mathcal{P}}$).

Theorem 3. *There is a pushdown automaton $\mathcal{P}$ such that any well-formed marking of $G_{\mathcal{P}}$ has an undecidable MSO theory.*

Proof. The proof used to establish Theorem 2 no longer works because *e.g.* we could have $k_1 = k_3$ but only the edge leading to the configuration with the state p_1 is marked. However, the same pushdown automaton is working but an extra trick is required.

The trick is that we will only be interested in triples (k_1, k_2, k_3) where k_1, k_2 and k_3 are even (and these can be easily filtered out by an MSO sentence, *e.g.* considering the path from the initial configuration corresponding to $(0, 0, 0)$ and performing a modulo 2 computation). We call these configurations ***even*** configurations. With an even configuration and the corresponding triple (k_1, k_2, k_3) we can associate a pair of counters $(\frac{k_1-k_3}{2}, \frac{k_2-k_3}{2}) \in \mathbb{Z} \times \mathbb{Z}$. We take the same pushdown automaton as previously but now in the first mode to simulate changes on the counter one must do two identical successive transitions: *e.g.* to increment counter $k_1 - k_3$ and decrement the counter $k_2 - k_3$ one pushes $(3, 1, 2)$ twice.

Fix a well-formed marking H of $G_{\mathcal{P}}$. Assume we are in an even configuration with an associated triple (k_1, k_2, k_3) and that we want to check whether $k_1 = k_3$. For this one looks at the marked outgoing edges in the present configurations: obviously, if both the one going to the state p_1 and the one going to the state p_3 are marked, one directly concludes that $k_1 = k_3$, hence that $k_1 - k_3 = 0$; however one may have $k_1 = k_3$ without being in the previous situation namely if $k_2 < k_1, k_3$ (as in the proof of Theorem 2) or if $k_2 > k_1, k_3$, $k_1 = k_3$ but only the edge to p_1 is marked (or symmetrically the one to p_3). We handle the case where $k_2 < k_1, k_3$ as previously, *i.e.* by looking at a path (of even length) along which only pushing $(2, 3, 2)$ are performed and that leads to a configuration with

some property. The latter property is either that both the outgoing edge to p_1 and to p_3 are marked or that only the outgoing edge to p_1 is marked (*resp.* p_3) and in the non-even configuration obtained by pushing $(3,3,2)$ (*resp.* $(1,3,2)$) the edge to p_3 (*resp.* p_1) is marked: this means that $k_1 \leqslant k_3$ (*resp.* $k_3 \leqslant k_1$) and that $k_1 + 3 \geqslant k_3 + 2$ (*resp.* $k_3 + 2 \geqslant k_1 + 1$) hence that $k_1 = k_3$ as both are even.

As the existence of a path (of even length) along which only pushing $(2,3,2)$ are performed and that leads to a configuration with the previous property, can easily be expressed as an MSO formula, and as being an even configuration can also be stated in MSO, it follows that one can write an MSO formula (with one order-1- free variable) that holds exactly in those even configurations (in the first mode) where $k_1 = k_3$. Similarly, one can design a formula to check whether $k_2 = k_3$.

Then, as in the proof of Theorem 2, it follows that, for any 2-counter machine with tests for 0, one can build an MSO formula that is true in H if and only if the 2-counter machine halts from its initial configuration. As the halting problem is undecidable for 2-counter machine with tests for 0, it follows that H has an undecidable MSO theory. □

4 Consequences

We now briefly mention some implications of our results. As we know (by Theorem 1) that any pushdown graph has a decidable MSO theory, one directly gets the following.

Corollary 1. *There is a pushdown automaton $\mathcal{P}$ such that no well-formed marking of it is a pushdown automaton.*

A generalisation of reachability properties are given by the notion of pushdown reachability games. In the setting of pushdown graph (that for simplicity we assume without dead-end), one considers a partition of the control states of the underlying pushdown automaton between two players: Eve and Adam. A play consists in moving a pebble starting from an initial configuration as follows: at any stage of the game the player owning (the control state of) the current configuration chooses a successor and moves the pebble to it and so on forever. The play is won by Eve if the pebble eventually reaches a final configuration. A strategy for a player is a function mapping every prefix of plays to a valid move; a player respects a strategy along a play if he systematically plays the move indicated by the strategy; and a strategy is winning for a player from an initial vertex if the player wins any play starting from that vertex when he respects the strategy.

It is a classical result that the winning region for Eve (*i.e.* the set of configurations from which she has a winning strategy) can be defined in a fixpoint manner leading to an object called ***attractor*** (see *e.g.* [Zielonka(1998)]): this construction is a direct adaptation of the

definition (in Section 2.1) of the sequence $(W_i)_{i\geqslant 0}$ to the alternating setting. In particular, if Eve owns all the control states, the attractor coincides with W. Of course, the attractor constructions also come with a corresponding (positional) strategy (namely a strategy that only depend on the current configuration and that ensures to get closer to a final configuration). Hence, one has the following immediate consequence of Theorem 2 and Theorem 3.

Corollary 2. *There is a pushdown reachability game such that any marking of the underlying pushdown graph by an attractor strategy (or any sub-strategy of it) leads to a graph with an undecidable MSO theory.*

This result is interesting because it implies that the constructions in [Walukiewicz(2001), Serre(2004)] for pushdown games build a winning strategy that is not an attractor strategy. Indeed, the constructed strategies can be used to define a well-formed marking of the input pushdown graph which remains a pushdown graph.

Acknowledgements

The authors would like to thank Didier Caucal for numerous helpful discussions and for proof-reading an early version of this manuscript. They would also like to thanks the reviewers for all their valuable comments.

References

[Broadbent et al.(2010)Broadbent, Carayol, Ong, and Serre] C. Broadbent, A. Carayol, C.-H. L. Ong, O. Serre, Recursion Schemes and Logical Reflexion, in: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LiCS 2010), IEEE Computer Society, 120–129, 2010.

[Muller and Schupp(1985)] D. E. Muller, P. E. Schupp, The Theory of Ends, Pushdown Automata, and Second-Order Logic, Theoretical Computer Science 37 (1985) 51–75.

[Serre(2004)] O. Serre, Contribution à l'étude des jeux sur des graphes de processus à pile, Ph.D. thesis, Université Paris 7, 2004.

[Thomas(1997)] W. Thomas, Languages, Automata, and Logic, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Language Theory, vol. III, Springer-Verlag, 389–455, 1997.

[Walukiewicz(2001)] I. Walukiewicz, Pushdown processes: games and model-checking, Information and Computation 157 (2001) 234–263.

[Zielonka(1998)] W. Zielonka, Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees, Theoretical Computer Science 200 (1-2) (1998) 135–183.