# Regular Strategies In Pushdown Reachability Games

A. Carayol and M. Hague

LIGM, Université Paris-Est & CNRS and Royal Holloway University of London

**Abstract.** We show that positional winning strategies in pushdown reachability games can be implemented by deterministic finite state automata of exponential size. Such automata read the stack and control state of a given pushdown configuration and output the set of winning moves playable from that position.

This result can originally be attributed to Kupferman, Piterman and Vardi using an approach based on two-way tree automata. We present a more direct approach that builds upon the popular saturation technique. Saturation for analysing pushdown systems has been successfully implemented by Moped and WALi. Thus, our approach has the potential for practical applications to controller-synthesis problems.

## 1 Introduction

Pushdown systems are well-studied in the software verification community. Their stack mirrors the call stack of a first-order recursive program, and, as such, the control flow of such programs (for instance C and Java programs) can be accurately modelled [10]. These models have been a major part of the automata-theoretic approach to software model checking and considerable progress has been made in the implementation of scalable model checkers of pushdown systems. These tools (e.g. Bebop [2] and Moped [7, 13, 17, 18, 16]) are an essential back-end components of high-profile model checkers such as SLAM [1].

Verification instances are often simple reachability properties. That is, is there a path in the system leading to some designated "error" state? A richer model is that of *games* where two players (Éloise and Abelard) compete to meet a certain goal. Often these players model the system (Éloise) running in an antagonistic environment (Abelard). In a reachability game, one might ask whether it's possible for the system to eventually reach a desired state, regardless of the environmental input. More complex winning conditions, such as Büchi or parity conditions, allow games equivalent to verification against expressive temporal logics such as $\mu$LTL or the modal $\mu$-calculus (e.g. [6]).

In a seminal paper [20], Walukiewicz showed that determining the winner of a pushdown parity game is EXPTIME-complete. Cachat [5] and Serre [14] have independently generalised Walukiewicz's algorithm to compute the winning regions of these games. That is, the set of all positions in the game where a given player can force a win. They use Walukiewicz's algorithm as an oracle to guide the construction of a finite-state automaton recognising the winning region. Another approach, introduced by Piterman and Vardi [12], uses two-way alternating tree automata to navigate a tree representing all possible stacks: after several reductions, including the complementation of Büchi automata, an automaton accepting the winning region can be constructed.

An alternative approach, *saturation*, was popularised as a model-checking algorithm for pushdown systems by Bouajjani *et al.* [3] and independently by Finkel *et al.* [8]. The algorithm was extended to constructing the winning regions of pushdown reachability games by Bouajjani *et al.* [3], Büchi games by Cachat [4], and parity games by Hague and Ong [9].

As well as constructing the winning region, one may also wish to construct a representation of a player's *winning strategy.* A winning strategy monitors the progression of the play of a game and, when a state is in the player's winning region, advises which of a range of possible moves should be played in order to win the game. When the players are the system and the environment, a winning strategy describes *how* to control the system to ensure correctness. This is the *controller-synthesis* problem: given a system and a specification, construct a *controller* of the system that behaves according to the specification.

In the case of pushdown reachability, Büchi, or parity games, it is known that the players have *positional* winning strategies. That is, in order to prescribe the next winning move, a strategy needs only to have access to the current state of the game (as opposed to the entire history of play) [21].

Cachat has given two realisations of Éloise's winning strategy in a pushdown reachability game [4]. The first is a positional strategy, constructed via the saturation technique, that requires space linear in the size of the stack to compute the possible next moves. Alternatively, Cachat presents a strategy implemented by a pushdown automaton that tracks the moves of Abelard and recommends moves to Éloise. Since the automaton tracks the game, the strategy is not positional. However, prescribing the next move requires only constant time. Cachat also argues that similar strategies can be computed for Abelard for positions in his winning region [5].

In the case of Büchi games Cachat also showed that it is possible to construct a linear space positional strategy and a constant time (though not positional) pushdown strategy for Éloise. However, Cachat also observes that adopting his techniques for computing strategies for Abelard is not clear [5]. However, it is known that, even for the full case of parity games, a pushdown strategy exists using different techniques due to Walukiewicz [20] and Serre [15].

The above results use relatively complex systems to define winning strategies. One of the simplest representations of a positional winning strategy over a pushdown game is a regular strategy. In this case, the stack and control state of the current position in the game are read by a finite-state automaton which then outputs the next possible winning moves.

It can be shown that a positional strategy for pushdown parity games can be defined as a regular automaton, exponential in size. Kupferman *et al.* [11] obtain this result from Piterman and Vardi [12]. Essentially, the two-way tree automaton can be reduced to a one-way tree automaton of exponential size, and from this a deterministic automaton reading each branch of the tree (where each branch represents a stack) and recommending next moves can be derived.

However, as mentioned above, this tree automaton approach requires several involved reductions and it is unclear how such a technique may be implemented in practice. The saturation algorithm, however, lends itself readily to implementations computing the winning regions (e.g. Moped [7, 13, 17] and WALi [19] for single-player and Moped for two-player [18] reachability games).

In this work we show how regular positional strategies can be constructed for Éloise in a pushdown reachability game. In Cachat's technique, weights are assigned to runs of the winning region automaton. Éloise's strategy is to take the minimal accepting run of the current configuration and play the move associated to its first transition. Following this strategy the reachability goal will eventually be satisfied. However, this does not provide a regular positional strategy because the weights require space linear in the size of the run to compute. We show that a more subtle method of assigning weights allows different runs to be compared with constant space requirements. Thus, we construct a deterministic regular automaton implementing a positional winning strategy.

Like Piterman and Vardi's technique, our automaton is also exponential in size. However, we believe our construction to be more direct and more likely to be practicable. Indeed, the first step of the algorithm (the construction of the winning region) has already been successfully implemented, whereas Piterman and Vardi's has not.

## 2 Preliminaries

### 2.1 Pushdown Games

A *pushdown reachability game* $\mathcal{G}$ is a given by a tuple $(\mathcal{P}, \Sigma, \mathcal{R}, \mathcal{C}_F)$ where $\mathcal{P} = \mathcal{P}_A \uplus \mathcal{P}_E$ is a finite set of control states partitioned into Abelard and Éloise states respectively, $\Sigma$ is the finite stack alphabet, $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma) \times \left(\mathcal{P} \times \Sigma^{\leq 2}\right)$ is the set of transitions and $\mathcal{C}_F$ is a set of target configurations, where a *configuration* is a tuple $(p, w)$ with $p$ being a control state in $\mathcal{P}$ and $w$ a stack in $\Sigma^*$.

We write $(p, a) \hookrightarrow (p', w)$ for the transition $((p, a), (p', w))$. In the configuration $\alpha = (p, aw)$, the pushdown system can apply the transition $(p, a) \hookrightarrow (p', u)$ to go to the configuration $\alpha' = (p', uw)$.

In the following, for technical convenience, we will assume for each $p \in \mathcal{P}$ and $a \in \Sigma$ there exists some $(p, a) \hookrightarrow (p', w) \in \mathcal{R}$. Furthermore, we will assume a bottom-of-stack symbol $\perp$ that is neither pushed onto nor popped from the stack. These two conditions together ensure that from a configuration $(p, w\perp)$ it is not possible for the system to become stuck; that is, reach a configuration with no successor.

A *play* of a pushdown game is a sequence $(p_0, w_0), (p_1, w_1), \ldots$ where $(p_0, w_0)$ is some starting configuration and $(p_{i+1}, w_{i+1})$ is obtained from $(p_i, w_i)$ via some $(p_i, a) \hookrightarrow (p_{i+1}, w) \in \mathcal{R}$. In the case where $p_i \in \mathcal{P}_E$ it is Éloise who chooses the transition to apply, otherwise Abelard chooses the transition.

The *winner* of a play $(p_0, w_0), (p_1, w_1), \ldots$ is Éloise if there exists some $i$ such that $(p_i, w_i) \in \mathcal{C}_F$; otherwise, Abelard wins the play. The *winning region*

$\mathcal{W}$ of a pushdown game is the set of all configurations from which Éloise can always win all plays, regardless of the transitions chosen by Abelard.

### 2.2 Alternating $\mathcal{P}$-Automata

To recognise sets of configurations, we use *alternating $\mathcal{P}$-automata.* These were first used by Bouajjani *et al.* [3].

An alternating $\mathcal{P}$ automaton is a tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \mathcal{F}, \delta)$ where $\mathcal{Q}$ is a finite set of states such that $\mathcal{P} \subseteq \mathcal{Q}$, $\Sigma$ is a finite alphabet, $\mathcal{F} \subseteq \mathcal{Q}$ is the set of accepting states, and $\delta \subseteq \mathcal{Q} \times \Sigma \times 2^{\mathcal{Q}}$ is a transition relation. We denote a transition $(q, a, Q)$ as $q \xrightarrow{a} Q$.

A *run over a word* $a_1 \ldots a_n \in \Sigma^*$ from a state $q_0$ is a sequence

$$Q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_n} Q_{n+1}$$

where each $Q_i$ is a set of states such that $Q_1 = \{q_0\}$, and for each $1 \leq i \leq n$ we have

$$Q_i = \{q_1, \ldots, q_m\} \quad \text{and} \quad Q_{i+1} = \bigcup_{1 \leq j \leq m} P_j$$

where for each $1 \leq j \leq m$ we have $q_j \xrightarrow{w} P_j$. The run is accepting if $Q_{n+1} \subseteq \mathcal{F}$. Thus, for a given state $q$, we define $\mathcal{L}_q(\mathcal{A})$ to be the set of words over which there is an accepting run of $\mathcal{A}$ from $\{q\}$. Finally, we define

$$\mathcal{L}(\mathcal{A}) = \{(p, w) \mid p \in \mathcal{P} \text{ and } w \in \mathcal{L}_p(\mathcal{A})\} \ .$$

When $Q_i$ is a singleton set, we will often omit the set notation. For example, the run above could be written $q_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} Q_{n+1}$. Furthermore, when $w = a_1 \ldots a_n$ we will write $q \xrightarrow{w} Q$ as shorthand for a run from $q$ to $Q$. In particular, we always have $q \xrightarrow{\varepsilon} q$ for any $q \in \mathcal{Q}$.

### 2.3 Constructing the Winning Region

We recall the saturation technique for computing Éloise's winning region of a pushdown reachability game. The algorithm was introduced by Bouajjani *et al.* [3] and is essentially an accelerated backwards fixpoint computation beginning with the target set of configurations and then computing all configurations that may reach it. We adapt the algorithm slightly by annotating each added transition with the number of iterations of the algorithm required to add the transition to the automaton. A similar, though more complex annotation scheme was used by Cachat to give a positional (though non-regular) winning strategy for Éloise [4].

Fix a pushdown reachability game $\mathcal{G} = (\mathcal{P}, \Sigma, \mathcal{R}, \mathcal{C}_F)$ such that $\mathcal{C}_F$ is represented by an alternating $\mathcal{P}$-automata $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{C}_F$. We will show how to construct an automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}) = \mathcal{W}$, where $\mathcal{W}$ is Éloise's winning region of $\mathcal{G}$.

Without loss of generality, we assume that there are no incoming transitions to any state $p \in \mathcal{P}$ of $\mathcal{A}$. The saturation algorithm constructs the automaton $\mathcal{B}$ that is the least fixed point of the sequence of automata $\mathcal{A}_0, \mathcal{A}_1, \ldots$ defined below. We simultaneously construct the sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots$ and two annotation functions $I$ and $R$ that annotate each transition $t \in \mathcal{Q} \times \Sigma \times 2^{\mathcal{Q}}$. The $I$ function assigns to each rule its *birthdate* : a natural number which is intuitively the number of iterations of the saturation algorithm required to add the transition to $\mathcal{B}$. Since the algorithm is a backwards reachability algorithm, the birthdate broadly gives the number of transitions required to either remove the corresponding stack character or rewrite it to part of a stack in the set of target configurations. The $R$ partial function assigns to each transition starting with a state of Éloise and whose birthdate is not 0 the rule of the pushdown game responsible for the addition of the transition to the automaton. All transitions in $\mathcal{A}_0$ will have the birthdate 0 assigned by $I$.

Initially, let $I(t) = 0$ for each $t \in \delta_0$ and define $\mathcal{A}_0 = \mathcal{A} = (\mathcal{Q}, \Sigma, \delta_0, \mathcal{F})$. Then we define $\mathcal{A}_{i+1} = (\mathcal{Q}, \Sigma, \delta_{i+1}, \mathcal{F})$ where $\delta_{i+1}$ is the smallest set of transitions such that

1. $\delta_i \subseteq \delta_{i+1}$, and
2. for each $p \in \mathcal{P}_E$, if $r = (p, a) \hookrightarrow (p', w) \in \mathcal{R}$ and $p' \xrightarrow{w} Q$ is a run of $\mathcal{A}_i$, then
$$t = p \xrightarrow{a} Q \in \delta_{i+1}$$
and if $t \notin \delta_i$ then set $I(t) = i + 1$ and $R(t) = r$, and
3. for each $p \in \mathcal{P}_A$ and $a \in \Sigma$ we have
$$t = p \xrightarrow{a} Q \in \delta_{i+1}$$
where, letting
$$\{(p_1, w_1), \ldots, (p_m, w_m)\} = \{(p', w) \mid (p, a) \hookrightarrow (p', w) \in \mathcal{R}\}$$
we have $Q = \bigcup_{1 \le j \le m} Q_j$ where for each $1 \le j \le m$, $p_j \xrightarrow{w_j} Q_j$ is a run of $\mathcal{A}_i$. Furthermore, if $t \notin \delta_i$ then set $I(t) = i + 1$.

One can prove that $\mathcal{L}(\mathcal{B}) = \mathcal{W}$. Since the maximum number of transitions of an alternating automaton is exponential in the number of states (and we do not add any new states), we have that $\mathcal{B}$ is constructible in exponential time.

**Theorem 1 ([3]).** *The winning region of a pushdown reachability game is regular and constructible in exponential time.*

Before proceeding with the construction of the strategy, we briefly discuss why $\mathcal{L}(\mathcal{B}) = \mathcal{W}$. It is well known that the winning region for Éloise is the smallest set $\mathcal{W}$ such that $\mathcal{L}(\mathcal{A}_0) \subseteq \mathcal{W}$ and $\mathcal{W} = \mathrm{Pre}(\mathcal{W})$ where for any set of configuration $C$,

$$\begin{aligned}\mathrm{Pre}(C) = \{&c' \text{ of Éloise} \mid \exists c \in C, c' \to c\} \\ &\cup \{c' \text{ of Abelard} \mid \forall c \in C, c' \to c \Rightarrow c \in C\}\end{aligned}$$

The key property of the algorithm is that it ensures that $\mathcal{L}(\mathcal{B})$ is closed under the Pre operation (*i.e.,* $\mathrm{Pre}(\mathcal{L}(\mathcal{B})) = \mathcal{L}(\mathcal{B})$). More precisely, it ensures that for all

$i \geq 0$, $\mathrm{Pre}(\mathcal{L}(\mathcal{A}_i)) \subseteq \mathcal{L}(\mathcal{A}_{i+1})$. Hence as $\mathcal{B}$ is by definition equal to $\mathcal{A}_N = \mathcal{A}_{N+1}$, we have that $\mathrm{Pre}(\mathcal{L}(\mathcal{B})) = \mathcal{L}(\mathcal{B})$. As $\mathcal{L}(\mathcal{B})$ contains $\mathcal{L}(\mathcal{A}_0)$, it follows that $\mathcal{L}(\mathcal{B})$ contains the winning region of Éloise.

For the converse inclusion, it is necessary to show that every configuration accepted by $\mathcal{B}$ belongs to the winning region of Éloise. For this we need to fix a strategy for Éloise that is winning from every configuration in $\mathcal{L}(\mathcal{B})$.

The strategies of Éloise considered in this article consist of associating to every run $\rho$ a weight $\Omega(\rho)$ and a well-founded ordering $<$ on weights. The strategy consists in picking the successor of a configuration of Éloise in $\mathcal{L}(\mathcal{B}) \backslash \mathcal{A}_0$ accepted by a run of $\mathcal{B}$ with the smallest possible weight. The weight is defined such that along any play following this strategy the weight of the smallest accepting run strictly decreases. As the ordering is assumed to be well-founded this ensures that a configuration in $\mathcal{L}(\mathcal{A})$ is eventually reached.

The key property here is that the algorithm ensures that for every configuration $c$ of Éloise accepted by a run $\rho$ of $\mathcal{B}$ which does not belong to $\mathcal{L}(\mathcal{A}_0)$, there exists a configuration $c'$ accepted by a run $\rho'$ of $\mathcal{B}$ such that $c \to c'$ . Moreover $\rho'$ is obtained by replacing the topmost transition of $\rho$ by several transitions which are *younger*. Similarly for a configuration $c$ of Abelard accepted by some run $\rho$ of $\mathcal{B}$, we have that any configuration $c'$ such that $c \to c'$ is accepted by a run $\rho'$ of $\mathcal{B}$ which is obtained by replacing the topmost transition of $\rho$ by several transitions which are *younger*.

A possible weight for a run $\rho$ is hence a tuple $(n_N, \ldots, n_0) \in \mathbb{N}$ where $N$ is the maximum birthdate of a transition appearing in the automaton $\mathcal{B}$ and, for all $i \geq 0$, $n_i$ is the number of transitions of birthdate $i$. The ordering is here the lexicographic ordering. Sadly this notion of weight cannot be handled by a finite state automaton which is the goal of this article. In the following section, we define a notion of weight that is compatible with finite state automata.

# 3 Regular Strategies

## 3.1 Runs As Trees

A run of $\mathcal{B}$ over a word $a_1 \ldots a_n \in \Sigma^*$ from a state $q_0$ can be represented by an unordered, unranked tree of depth $n$ such that,

1. the root node is labelled $q_0$.
2. for each node $\eta$ at depth $0 \leq i < n$ of the tree labelled $q$ there is a transition $t = q \xrightarrow{t} \{q_1, \ldots, q_m\}$ such that $\eta$ has children $\eta_1, \ldots, \eta_m$ labelled $q_1, \ldots, q_m$ respectively and each edge $(\eta, \eta_j)$ for all $1 \leq j \leq m$ is labelled by $t$.

A run, represented as a tree, gives rise to a set of sequences of transitions $t_1, \ldots, t_n$ that are the labellings of the edges of each complete branch of the tree (that is, running from the root node to some leaf node). Given a run $\rho$, let $\mathrm{Branches}(\rho)$ be the set of sequences of labels on the branches of $\rho$.

## 3.2 Ordering On Runs

To define strategies, we first introduce an ordering between runs of the saturated automaton. To do so, we assign to each branch of the run a weight and take the weight of the run to be the maximum weight of all of its branches. The runs are then (pre-)ordered by comparing their weights.

*Weights.* Let $N$ be the number of iterations required for the saturation to reach a fixed point. That is $N$ is the smallest number such that for all $t$ we have $I(t) \leq N$. Note that $N$ is fixed for a given $\mathcal{B}$. The weights are tuples in $\mathbb{N}^{N+1}$ where $\mathbb{N}$ denotes the set of natural numbers. The weights are compared using the reverse lexicographic-ordering

$$(i_0, \ldots, i_N) \prec (i'_0, \ldots, i'_N)$$

whenever there exists $N \geq j \geq 0$ such that $i_j < i'_j$ and for all $N \geq k > j$ we have $i_k = i'_k$. Similarly, we write $\preceq$ to denote $\prec \cup =$. Moreover we write

$$(i_0, \ldots, i_N) \prec_j (i'_0, \ldots, i'_N)$$

whenever $i_j < i'_j$ and for all $N \geq k > j$ we have $i_k = i'_k$.

*Weight of a branch.* Fix a branch $\beta = t_n, \ldots, t_1$ of the run which reads the stack from top to bottom (thus $t_n$ reads the topmost character and $t_1$ the bottommost character). For all $0 \leq j \leq N$, we take $\mathrm{lft}_j$ to be the position from the bottom of stack of the left-most transition of birthdate $j$ and 0 if no such transition exists, *i.e.* $\mathrm{lft}_j = \max\{i \mid I(t_i) = j\}$ (with $\max \emptyset = 0$). Intuitively we first take into account the position (from the bottom) of the transition of birthdate $N$ that is the furthest from the bottom. The greater this position is the greater the weight. Then we look at the position of the transition of age $N-1$ that is the furthest from the bottom. We only take it into account if it is after the previous position. This restriction is only here to ensure that the order can be implemented by an automaton with an exponential number of states. And so on. . . .

The weight of the branch $\beta$ is defined to be

$$\Omega(\beta) := (i_0, \ldots, i_N)$$

where $i_j = \mathrm{lft}_j$ if $\mathrm{lft}_j > \max\{\mathrm{lft}_{j+1}, \ldots, \mathrm{lft}_N\}$ and 0 otherwise.

For example, consider a branch $t_1, t_2, t_3, t_4, t_5, t_6$ with a corresponding sequence of birthdates $1, 4, 2, 2, 5, 1$ and assume that $N = 5$. We have $\mathrm{lft}_5 = 2$, $\mathrm{lft}_4 = 5$, $\mathrm{lft}_3 = 0$, $\mathrm{lft}_2 = 4$,$\mathrm{lft}_1 = 6$ and $\mathrm{lft}_0 = 0$. The weight of this branch is hence $(0, 6, 0, 0, 5, 2)$.

*Weight of a run and of a configuration.* The weight of a run $\rho$ is the maximum weight (for $\prec$) of one of its branches.

$$\Omega(\rho) := \max\{\Omega(\beta) \mid \beta \in \mathrm{Branches}(\rho)\}$$

Finally we assign to any configuration $(p, w)$ accepted by the automaton $\mathcal{B}$ the weight $\Omega((p, w)) = \min\{\Omega(\rho) \mid \rho \text{ accepts } (p, w)\}$ of its smallest accepting run.

The ordering $\prec$ is naturally extended to a total pre-ordering on runs by taking for any two runs $\rho$ and $\rho'$, $\rho \prec \rho'$ if $\Omega(\rho) \prec \Omega(\rho')$. Similarly $\prec$ is extended to configurations accepted by $\mathcal{B}$.

### 3.3 Éloise's Winning Strategy

Given the ordering defined above, we can define a winning strategy for Éloise. Her strategy is a simple one. At any configuration $(p, aw)$ in her winning region, let $\rho$ be a smallest accepting run of $\mathcal{B}$ with respect to $\prec$. Furthermore, let $t = p \xrightarrow{a} Q$ be the first transition of $\rho$. To win the game, Éloise can play the rule $R(t)$. For any configuration $(p, w)$ with $p \in \mathcal{P}_E$, let $\mathrm{Play}_E((p, w))$ be the set of rules $r$ that annotate the first transition of a $\prec$-smallest run of $\mathcal{B}$ over $(p, w)$ whenever $(p, w) \in \mathcal{W} \setminus \mathcal{C}_F$. Otherwise, let $\mathrm{Play}_E((p, w)) = \emptyset$.

**Lemma 1.** *For a given pushdown reachability game $\mathcal{G} = (\mathcal{P}, \Sigma, \mathcal{R}, \mathcal{C}_F)$ with $\mathcal{W}$, $\mathcal{B}$ and $\prec$ being Éloise's winning region, the automaton constructed by saturation and its associated ordering respectively, it is the case that, for all configurations $(p, aw) \in \mathcal{W}$, we have either*

1. *$(p, aw) \in \mathcal{C}_F$, or*
2. *$p \in \mathcal{P}_E$ and for all $(p, a) \hookrightarrow (p', u) \in \mathrm{Play}_E((p, w))$ we have*

$$(p', uw) \prec (p, aw) \quad \textit{with}\, (p', uw) \in \mathcal{W},$$

3. *$p \in \mathcal{P}_A$ and for all $(p, a) \hookrightarrow (p', u) \in \mathcal{R}$ we have*

$$(p', uw) \prec (p, aw) \quad \textit{with}\, (p', uw) \in \mathcal{W}.$$

*Proof.* We only consider Éloise's case as Abelard's case is similar. Let $(p, aw) \in \mathcal{W}$ be a configuration of Éloise. Let $\rho$ be a minimal run accepting for $(p, aw)$ and let $t = p \xrightarrow{a} Q$ be the first transition of $\rho$. Furthermore for all $q \in Q$, let $\rho_q$ be the subrun of $\rho$ accepting $w$ from $q$ Finally assume that $R(t) = (p, a) \hookrightarrow (p', u)$.

By definition of the saturation algorithm, there exists a run $\rho_u$ of the form $p' \xrightarrow{u} Q$ where every transition $t'$ labelling $\rho_u$ is such that $I(t') < I(t)$. Let $\rho'$ be the run obtained by plugging into $\rho_u$ the run $\rho_q$ at each leaf labelled by $q \in Q$ . This runs accepts $(p', uw)$ and hence $(p', uw)$ belongs to $\mathcal{W}$.

Furthermore every branch $\beta'$ of $\rho'$ is obtained from some branch $\beta$ of $\rho$ by replacing the first transition $t$ by a sequence of transitions $t_1, \ldots, t_{|u|}$ where for all $1 \leq j \leq |u|$, $I(t_j) < I(t)$. By definition of the order $\prec$, $\Omega(\beta') \prec_{I(t)} \Omega(\beta)$. Hence $\Omega((p', uw)) \preceq \Omega(\rho') \prec \Omega(\rho) = \Omega((p, aw))$. □

**Theorem 2.** *The positional strategy $\mathrm{Play}_E$ is winning for Éloise from every configuration of her winning region.*

*Proof.* Assume towards a contradiction that there exists an infinite play $c_0, c_1 \ldots$ which starts in the winning region of Éloise and that does not reach a configuration in $\mathcal{C}_F$. From Lemma 1, we immediately obtain that $\Omega(c_0) \succ \Omega(c_1) \succ \cdots$. Being a (reverse) lexicographic ordering built upon well-founded orderings, $\prec$ is a well-founded total ordering on weights which brings the contradiction. □

### 3.4 Regular Winning Strategies

We show that the above strategy can be implemented by a regular automaton. That is, we define a finite deterministic automaton which processes a configuration $(p, w)$ and outputs the set of rules $\text{Play}_E((p, w))$ whenever $p \in \mathcal{P}_E$ and $(p, w) \in \mathcal{W} \setminus \mathcal{C}_F$, and $\emptyset$ otherwise. The automaton reads the stack content $w$ from the bottom of the stack and reaches some state $s$. The output is obtained by applying a mapping $\text{Out}_p$ to $s$. We first formally define strategy automata.

**Definition 1.** *Given a pushdown game $(\mathcal{P}, \Sigma, \mathcal{R}, \mathcal{C}_F)$, a* strategy *automaton $\mathcal{S}$ is a tuple $(S, \Sigma, s_0, \delta, (\text{Out}_p)_{p\in\mathcal{P}_E})$ where $S$ is a finite set of states, $\Sigma$ is an input alphabet, $s_0$ is an initial state and $\delta : S \times \Sigma \mapsto S$ is a transition function and for all $p \in \mathcal{P}_E$, $\text{Out}_p : S \mapsto 2^{\mathcal{R}}$ is an output mapping for the control state $p$.*

As usual, we extend the transition function $\delta$ to words over the input alphabet $\Sigma$. Writing $\tilde{w}$ to denote the mirror of the word $w$, the output $\text{Play}_{\mathcal{S}}((p, w))$ of a strategy automaton $\mathcal{S}$ over a given configuration $(p, w)$ is defined to be

$$\text{Play}_{\mathcal{S}}((p, w)) := \text{Out}_p(\delta(s_0, \tilde{w}))$$

Given a pushdown game $\mathcal{G} = (\mathcal{P}, \Sigma, \mathcal{R}, \mathcal{C}_F)$ as well as an automaton $\mathcal{B} = (\mathcal{Q}, \Sigma, \delta, \mathcal{F})$ representing Éloise's winning region, obtained by saturation, along with its associated ordering $\prec$. We define a strategy automaton $\mathcal{S}_{\mathcal{B}}$ such that for all $(p, w)$ we have

$$\text{Play}_{\mathcal{S}_{\mathcal{B}}}((p, w)) = \text{Play}_E((p, w)) \ .$$

### 3.5 The automaton $\mathcal{S}_{\mathcal{B}}$

**The State-Set** The automaton $\mathcal{S}_{\mathcal{B}}$ will run $\mathcal{B}$ in reverse, starting from the bottom of the stack. Assuming that the automaton has read the word $\tilde{w}$, the state of $\mathcal{S}_{\mathcal{B}}$ will have as a component a mapping $\text{Moves}_w : \mathcal{P}_E \mapsto 2^{\mathcal{R}}$ such that for all state $p \in \mathcal{P}_E$, $\text{Moves}_w(p) = \text{Play}_E((p, w))$. Note that this mapping is only defined for states belonging to Éloise as those are the only states for which she is required to make a decision. Clearly if the automaton can maintain this information, we have constructed a strategy automaton. In order to update this component while keeping the state set at most exponential, the automaton will maintain two additional pieces of information.

- the set of states $\text{Acc}_w \in 2^{\mathcal{Q}}$ from which $\mathcal{B}$ admits an accepting run on $w$,
- a partial mapping in $f_w : \mathcal{Q} \times \mathcal{Q} \to \{\prec_\iota, \text{EQ}, \succ_\iota \mid 0 \le \iota \le N\}$ which when applied to two states $q_1$ and $q_2 \in \text{Acc}_w$ compares the minimal runs of $\mathcal{B}$ starting in state $q_1$ and $q_2$ respectively.

Intuitively, for the automaton to update $\text{Moves}_{wa}$, it is only necessary to know the transitions that can start a minimal accepting run for any state of Éloise. We will see below that this information can be computed only using the comparison provided by $f_w$.

More formally, let $w$ be a stack content. The set $\text{Acc}_w \subseteq \mathcal{Q}$ is the set of states $\mathcal{B}$ from which $\mathcal{B}$ has an accepting run, that is

$$\text{Acc}_w := \left\{ q \in \mathcal{Q} \;\middle|\; q \xrightarrow{w} F \subseteq \mathcal{F} \right\} \ .$$

The partial mapping $f_w : \mathcal{Q} \times \mathcal{Q} \to \{\prec_\iota, \mathrm{EQ}, \succ_\iota \mid 0 \leq \iota \leq N\}$ is defined for all states $q_1$ and $q_2 \in \mathrm{Acc}_w$ by taking

$$f_w(q_1, q_2) := \begin{cases} \prec_\iota \text{ if } \Omega(\rho_1) \prec_\iota \Omega(\rho_2) \\ \succ_\iota \text{ if } \Omega(\rho_1) \succ_\iota \Omega(\rho_2) \\ \mathrm{EQ} \text{ if } \Omega(\rho_1) = \Omega(\rho_2) \end{cases}$$

where $\rho_1$ and $\rho_2$ are $\prec$-minimal runs accepting $w$ from $q_1$ and $q_2$ respectively.

**The Transition function** To define the transition function of the strategy automaton, it remains to show how to compute $\mathrm{Acc}_{aw}$, $f_{aw}$ and $\mathrm{Moves}_{aw}$ using only $a$, $\mathrm{Acc}_w$ and $f_w$. To do this we will define three functions $\mathrm{Up}_{\mathrm{Acc}}$, $\mathrm{Up}_f$, and $\mathrm{Up}_{\mathrm{Moves}}$ that perform the updates for their respective components.

We define $\mathrm{Up}_{\mathrm{Acc}}$ following the standard membership algorithm for alternating automata, and obtain the following lemma.

**Definition 2** **($\mathrm{Up}_{\mathrm{Acc}}$).** *We define*

$$\mathrm{Up}_{\mathrm{Acc}}(a, \mathrm{Acc}_w) := \left\{ q \;\middle|\; q \xrightarrow{a} Q \in \delta \wedge Q \subseteq \mathrm{Acc}_w \right\} .$$

**Lemma 2.** *For all $w \in \Sigma^*$ and all $a \in \Sigma$ we have $\mathrm{Acc}_{aw} = \mathrm{Up}_{\mathrm{Acc}}(a, \mathrm{Acc}_w)$.*

Computing $f_{aw}$ is more involved and requires some preliminary notations.

First observe that the mapping $f_w$ induces a total pre-ordering on the set $\mathrm{Acc}_w$. For all subsets $Q \subseteq \mathrm{Acc}_w$, we denote by $\max(Q)$ the set of all maximal elements for this ordering. We write $f_w(\max(Q_1), \max(Q_2))$ for the value $f_w(q, q')$ for any $q \in \max(Q_1)$ and $q' \in \max(Q_1)$. As all the elements of $\max(Q_1)$ (resp. $\max(Q_2)$) are equal for the ordering, the choice of $q$ and $q'$ is irrelevant.

As a first step, we use the information of $f_w$ to compare the weights of minimal runs on $aw$ starting with two given transitions $t_1$ and $t_2$. Take any two transitions $t_1 = q_1 \xrightarrow{a} Q_1$ and $t_2 = q_2 \xrightarrow{a} Q_2$ with $Q_1 \subseteq \mathrm{Acc}_w$ and $Q_2 \subseteq \mathrm{Acc}_w$ and $I(t_1) = \gamma_1$ and $I(t_2) = \gamma_2$. There are two cases to comparing runs starting with $t_1$ and $t_2$. In the first case, the minimal runs from $Q_1$ and $Q_2$ differ on some weight $\iota > \gamma_1, \gamma_2$. In this case the ordering is dominated by $\iota$ and remains unchanged. If, however, $\gamma_1 \geq \iota$ or $\gamma_2 \geq \iota$, then the relative ordering of the runs is decided by $t_1$ and $t_2$. More formally, we write

$$t_1 \prec_\iota t_2$$

if either

1. the ordering between the minimal runs is not decided by $t_1$ and $t_2$, that is
   (a) $f_w(\max(Q_1), \max(Q_2)) = \prec_\iota$, and
   (b) $\iota > \gamma_1, \gamma_2$.
2. the ordering is decided by $t_1$ and $t_2$, that is
   (a) $\gamma_1 < \gamma_2$ and $\iota = \gamma_2$ and
   (b) $f_w(\max(Q_1), \max(Q_2)) \notin \{\prec_{\iota'}, \succ_{\iota'} \mid \gamma_2 < \iota' \leq N\}$, or

In addition, we write $t_1 \succ_\iota t_2$ when $t_2 \prec_\iota t_1$. We also write $t_1 \,\mathrm{EQ}\, t_2$ when

1. $\gamma_1 = \gamma_2$, and
2. $f_w(\max(Q_1), \max(Q_2))$ belongs to $\{\mathrm{EQ}, \prec_\iota, \succ_\iota \mid 0 \le \iota \le \gamma_1 = \gamma_2\}$.

**Lemma 3.** *For any two transitions $t_1 = q_1 \xrightarrow{a} Q_1$ and $t_2 = q_2 \xrightarrow{a} Q_2$ with $Q_1 \subseteq \mathrm{Acc}_w$ and $Q_2 \subseteq \mathrm{Acc}_w$, $t_1 \prec_\iota t_2$ (resp. $t_1 \succ_\iota t_2$, resp. $t_1 \,\mathrm{EQ}\, t_2$ ) if and only if $\Omega(\rho_1) \prec_\iota \Omega(\rho_2)$ (resp. $\Omega(\rho_1) \succ_\iota \Omega(\rho_2)$, resp. $\Omega(\rho_1) = \Omega(\rho_2)$) where $\rho_1$ and $\rho_2$ are the minimal runs accepting $aw$ and starting with $t_1$ and $t_2$ respectively.*

*Proof.* Let $I(t_1) = \gamma_1$ and $I(t_2) = \gamma_2$. We first argue that a minimal run beginning with $t_1$ (resp. $t_2$) can be constructed from $t_1$ and a minimal run from $Q_1$ (resp. $Q_2$). Let $\rho'_1$ be a minimal run from $Q_1$. Let $\rho_1 = t_1\rho''_1 \prec_\iota t_1\rho'_1$. If $\gamma_1 \ge \iota$, then $\Omega(t_1\rho''_1) = \Omega(t_1\rho'_1)$ and thus $t_1\rho'_1$ is also a minimal run. Otherwise $\gamma_1 < \iota$ and $\rho_1 = t_1\rho''_1 \prec_\iota t_1\rho'_1$ implies $\rho''_1 \prec_\iota \rho'_1$, contradicting the minimality of $\rho'_1$.

Now, let $\rho_1 = t_1\rho'_1$ and $\rho_2 = t_2\rho'_2$. Suppose $t_1 \prec_\iota t_2$. There are two cases. When $f_w(\max(Q_1), \max(Q_2)) = \prec_\iota$ (implying $\rho'_1 \prec_\iota \rho'_2$) and $\iota > \gamma_1, \gamma_2$ then we conclude $t_1\rho'_1 \prec_\iota t_2\rho'_2$. Otherwise $\gamma_1 < \gamma_2 = \iota$ and $f_w(\max(Q_1), \max(Q_2))$ is not $\prec_{\iota'}$ or $\succ_{\iota'}$ for some $\iota' > \gamma_2$. From the last condition, we know $\rho'_1$ and $\rho'_2$ are equal or differ only on some weight $\iota' \le \gamma_2$. Thus we know $t_1\rho'_1 \prec_{\gamma_2=\iota} t_2\rho'_2$.

In the other direction, suppose $t_1\rho'_1 \prec_\iota t_2\rho'_2$. If $\iota > \gamma_1, \gamma_2$, then we have $\rho'_1 \prec_\iota \rho'_2$ and thus $f_w(\max(Q_1), \max(Q_2)) = \prec_\iota$. We then have $t_1 \prec_\iota t_2$ as required. If $\gamma_1 \ge \iota$ or $\gamma_2 \ge \iota$, then for $t_1\rho'_1$ to be smaller than $t_2\rho'_2$ we must have $\gamma_1 < \gamma_2 = \iota$ and moreover $\rho'_1$ and $\rho'_2$ must have the same weight, or differ on some $\iota' \le \iota$. Thus, $f_w(\max(Q_1), \max(Q_2)) \notin \{\prec_{\iota'}, \succ_{\iota'} \mid \gamma_2 < \iota' \le N\}$ and $t_1 \prec_\iota t_2$.

The case for $\succ_\iota$ is symmetric, hence it only remains to consider EQ. We have $t_1 \,\mathrm{EQ}\, t_2$ if and only if $\gamma_1 = \gamma_2$ and $f_w(\max(Q_1), \max(Q_2))$ belongs to $\{\mathrm{EQ}, \prec_\iota, \succ_\iota \mid 0 \le \iota \le \gamma_1 = \gamma_2\}$. We have this iff $\rho'_1$ and $\rho'_2$ have equal weights or differ only on some $\iota \le \gamma_1, \gamma_2$ and, thus, iff $\Omega(t_1\rho'_1) = \Omega(t_2\rho'_2)$. □

As a consequence of the above lemma, we have defined a total pre-order on the set of $a$-transitions. For any set of $a$-transitions $T$, we denote by $\min(T)$ the set of minimal elements for this order. We are now ready to define $\mathrm{Up}_f$.

**Definition 3 ($\mathrm{Up}_f$).** *We define $\mathrm{Up}_f(a, \mathrm{Acc}_w, f_w)$ as the mapping $g$ defined for all states $q_1$ and $q_2 \in \mathrm{Acc}_{aw}$ by*

$$g(q_1, q_2) := f_w(\min(T_{q_1}), \min(T_{q_2}))$$

*where $T_{q_1} = \{q_1 \xrightarrow{a} Q_1 \mid Q_1 \subseteq \mathrm{Acc}_w\}$ and $T_{q_2} = \{q_2 \xrightarrow{a} Q_2 \mid Q_2 \subseteq \mathrm{Acc}_w\}$.*

It directly follows from Lemma 3 that:

**Lemma 4.** *For all $w \in \Sigma^*$ and all $a \in \Sigma$, we have $f_{aw} = \mathrm{Up}_f(a, \mathrm{Acc}_w, f_w)$.*

Finally, we define $\mathrm{Up}_{\mathrm{Moves}}$.

**Definition 4 ($\mathrm{Up}_{\mathrm{Moves}}$).** *We define $\mathrm{Up}_{\mathrm{Moves}}(a, \mathrm{Acc}_w, f_w)$ to be the mapping associating to any control state $p \in \mathcal{P}_E$ the set $\{R(t) \mid t \in \min(T_p)\}$ where $T_p = \left\{p \xrightarrow{a} Q \mid Q \subseteq \mathrm{Acc}_w\right\}$.*

**Lemma 5.** *For all $w \in \Sigma^*$ and all $a \in \Sigma$, we have*

$$\mathrm{Moves}_{aw} = \mathrm{Up}_{\mathrm{Moves}}(a, \mathrm{Acc}_w, f_w)\ .$$

**The Definition of $\mathcal{S}_\mathcal{B}$** We bring together the above discussion and define $\mathcal{S}_\mathcal{B}$.

**Definition 5.** *Given a pushdown game $\mathcal{G} = (\mathcal{P}, \Sigma, \mathcal{R}, \mathcal{C}_F)$ as well as an annotated automaton $\mathcal{B} = (\mathcal{Q}, \Sigma, \delta, \mathcal{F})$ constructed by saturation in $N$ steps, we define $\mathcal{S}_\mathcal{B}$ to be the strategy automaton $(S, \Sigma, s_0, \delta, (\mathrm{Out}_p)_{p \in \mathcal{P}_E})$ where*

$$S = 2^{\mathcal{Q}} \times (\mathcal{Q} \times \mathcal{Q} \to \{\prec_\iota, \mathrm{EQ}, \succ_\iota \mid 0 \leq \iota \leq N\}) \times \left(\mathcal{P}_E \mapsto 2^{\mathcal{R}}\right)$$

*and $s_0 = (\mathcal{F}, f_0, \mathrm{Moves}_0)$ where we have $f_0(q_1, q_2) = \mathrm{EQ}$ for all $q_1, q_2 \in \mathcal{F}$ and $\mathrm{Moves}_0(p) = \emptyset$ for all $p \in \mathcal{P}_E$, and*

$$\delta(a, (\mathrm{Acc}, f, \mathrm{Moves})) = \left(\mathrm{Up}_{\mathrm{Acc}}(a, \mathrm{Acc}), \mathrm{Up}_f(a, \mathrm{Acc}, f), \mathrm{Up}_{\mathrm{Moves}}(a, \mathrm{Acc}, f)\right)$$

*and finally, for all $p \in \mathcal{P}_E$, $\mathrm{Out}_p(\mathrm{Acc}, f, \mathrm{Moves}) = \mathrm{Moves}(p)$.*

The size of the automaton $\mathcal{S}_\mathcal{B}$ is exponential in the size of pushdown game.

**Theorem 3.** *Given a $\mathcal{P}$-automaton $\mathcal{B} = (\mathcal{Q}, \Sigma, \delta, \mathcal{F})$ constructed by saturation and a strategy automaton $\mathcal{S}_\mathcal{B}$ constructed as above, we have*

$$\mathrm{Play}_{\mathcal{S}_\mathcal{B}}((p, w)) = \mathrm{Play}_E((p, w)) \ .$$

*Proof.* Take a configuration $(p, w)$. By induction on the length of $w$, we show that upon reading $\tilde{w}$ the automaton $\mathcal{S}_\mathcal{B}$ reaches the state $(\mathrm{Acc}_w, f_w, \mathrm{Moves}_w)$.

In the base case, the initial state $s_0$ is by definition $(\mathrm{Acc}_\varepsilon, f_\varepsilon, \mathrm{Moves}_\varepsilon)$. The induction step immediately follows from Lemma 2, 4 and 5. The output is therefore $\mathrm{Moves}_w(p)$ which is by definition equal to $\mathrm{Play}_E((p, w))$. □

## 4 Conclusion

We gave the construction of a regular postional strategy for Éloise in a pushdown reachability game. The strategy automaton is a deterministic automaton of exponential size in the size of the pushdown game.

To define a similar strategy for Abelard, observe that any strategy of Abelard consisting in picking a move that stays outside of the winning region of Éloise is winning. A deterministic strategy automaton implementing such a strategy has states in $\Sigma \times 2^{\mathcal{Q}}$. After reading a stack content $\tilde{aw}$, the automaton reaches the state $(a, \mathrm{Acc}_w)$. For all $p \in \mathcal{P}_A$, the output mapping $\mathrm{Out}_p$ associates to a state $(a, \mathrm{Acc}_w)$ the set of rules $(p, a) \hookrightarrow (p', u) \in \mathcal{R}$ such there are no runs of $\mathcal{B}$ of the form $p' \xrightarrow{u} Q' \subseteq \mathrm{Acc}_w$.

If we consider pushdown Büchi reachability games, computing a regular positional strategy for Éloise can be reduced to the reachability case. Let $\mathcal{W}$ be the winning region of Éloise in the Büchi game and $\mathcal{C}_F$ be a regular set of final configurations. Consider the positional strategy consisting of playing any move that stays in $\mathcal{W}$ for configurations in $\mathcal{W} \cap \mathcal{C}_F$ and for configurations in $\mathcal{W} \setminus \mathcal{C}_F$ plays the regular positional strategy for the reachability game to $\mathcal{W} \cap \mathcal{C}_F$. As $\mathcal{W}$ is regular, the resulting strategy is regular and can be implemented by strategy automaton of exponential size.

Abelard's strategy is more complex and leads to the open problem of extending our approach to pushdown parity games. The saturation method underlying our approach was extended to these settings in [9]. The challenge is to define an ordering on runs of the saturated automaton that can be implemented by a finite state automaton of size at most exponential in that of the pushdown game.

*Acknowledgments* This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1] and the Labex Bézout as part of the program "Investissements d'Avenir" (ANR-10-LABX-58).

## References

1. T. Ball, V. Levin, and S. K. Rajamani. A decade of software model checking with slam. *Commun. ACM*, 54(7):68–76, 2011.
2. Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN*, pages 113–130, 2000.
3. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.
4. T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, pages 704–715, 2002.
5. T. Cachat. *Games on Pushdown Graphs and Extensions*. PhD thesis, RWTH Aachen, 2003.
6. E. Allen Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377, 1991.
7. J. Esparza and S. Schwoon. A bdd-based model checker for recursive programs. In *CAV*, pages 324–336, 2001.
8. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, volume 9, pages 27–37, 1997.
9. M. Hague and C.-H. Luke Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR*, pages 384–398, 2009.
10. N. D. Jones and S. S. Muchnick. Even simple programs are hard to analyze. *J. ACM*, 24:338–350, April 1977.
11. O. Kupferman, N. Piterman, and M. Y. Vardi. An automata-theoretic approach to infinite-state systems. In *Essays in Memory of Amir Pnueli*, pages 202–259, 2010.
12. N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In *CAV*, pages 387–400, 2004.
13. S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.
14. O. Serre. Note on winning positions on pushdown games with [omega]-regular conditions. *Inf. Process. Lett.*, 85(6):285–291, 2003.
15. O. Serre. *Contribution à létude des jeux sur des graphes de processus à pile*. PhD thesis, Université Paris 7 – Denis Diderot, UFR dinformatique, 2004.
16. D. Suwimonteerabuth, F. Berger, S. Schwoon, and J. Esparza. jmoped: A test environment for java programs. In *CAV*, pages 164–167, 2007.
17. D. Suwimonteerabuth, S. Schwoon, and J. Esparza. jmoped: A java bytecode checker based on moped. In *TACAS*, pages 541–545, 2005.
18. D. Suwimonteerabuth, S. Schwoon, and J. Esparza. Efficient algorithms for alternating pushdown systems with an application to the computation of certificate chains. In *ATVA*, pages 141–153, 2006.
19. WALi. https://research.cs.wisc.edu/wpis/wpds/download.php.
20. I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.
21. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.