Parity games played on transition graphs of one-counter processes

Olivier Serre

Parity games played on transition graphs of one-counter processes *

Olivier Serre

LIAFA, Université Paris VII & CNRS

Abstract. We consider parity games played on special pushdown graphs, namely those generated by one-counter processes. For parity games on pushdown graphs, it is known from [22] that deciding the winner is an ExpTime-complete problem. An important corollary of this result is that the μ-calculus model checking problem for pushdown processes is ExpTime-complete. As one-counter processes are special cases of pushdown processes, it follows that deciding the winner in a parity game played on the transition graph of a one-counter process can be achieved in ExpTime. Nevertheless the proof for the ExpTime-hardness lower bound of [22] cannot be adapted to that case. Therefore, a natural question is whether the ExpTime upper bound can be improved in this special case. In this paper, we adapt techniques from [11, 4] and provide a PSpace upper bound and a DP-hard lower bound for this problem. We also give two important consequences of this result. First, we improve the best upper bound known for model-checking one-counter processes against μ-calculus. Second, we show how these games can be used to solve pushdown games with winning conditions that are Boolean combinations of a parity condition on the control states with conditions on the stack height.

1 Introduction

Infinite two-player games with perfect information allow us to encode several challenging problems from formal verification, and this is one of the reasons why they are so intensively studied for several years. Several model-checking problems can be expressed as decision problems for games: the most famous example is that the μ-calculus model-checking problem is polynomially equivalent to the solution of a parity game. This correspondence was first proved for finite graphs [6] and later extended to various classes of infinite graphs, *e.g.* pushdown graphs [22, 23]. Two-player games also offer a very convenient framework to represent interaction of a program with some (possibly hostile) environment. In this approach, the first player represents the program while the second player simulates the environment. A winning strategy expresses a property that must hold *whatever* the environment does. Hence, finding a winning strategy for the first player

* This research has been partially supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES), (contract HPRN-CT-2002-00283), see www.games.rwth-aachen.de.

allows to synthesize a controller that restricts the program and ensures that the property expressed by the winning condition always holds [2].

The most standard setting of verification and synthesis only considers finite games. Nevertheless infinite models arise when recursive programs or programs with variable on infinite domains are considered. Therefore, solving games on such infinite objects is a natural question. The special case of pushdown processes, have been intensively studied from the games point of view (see *e.g.* [22, 11, 5, 3, 7, 19]) and the most important consequence for model-checking is that for pushdown processes, the μ-calculus model-checking problem is ExpTime-complete [22].

In this paper, we consider a natural subclass of pushdown processes, namely one-counter processes with zero test. Verification problems for one-counter processes have intensively been studied (see *e.g.* [9, 8]) but, for model-checking problems, most of the complexity results concern lower bounds whereas upper bounds generally follow from known results for pushdown processes. Hence, this frequently yields complexity gaps, as for μ-calculus model-checking where the lower bound is DP-hard [8] whereas the upper bound is ExpTime [22].

We consider parity games played on one-counter graphs and provide a PSpace algorithm to decide the winner in such games. Our procedure relies on a tricky adaptation and a precise analysis of the techniques from [11, 4] that were originally developed for pushdown games. Our result improves the ExpTime upper bound inherited from pushdown games [22]. As a by-product, it improves the best known upper bound for the μ-calculus model-checking for one-counter processes from ExpTime to PSpace.

Another consequence of our main result concerns pushdown games equipped with winning conditions that combine both regular conditions and conditions on how the stack height evolves during a play (*e.g.* unboundedness). Special cases of these games have been studied and shown to be decidable [5, 3, 7]. Here, we capture a larger class of games and we provide a more intuitive construction which generalizes those for parity games [22] and for strict unboundedness [19]. Moreover, our construction is very general, and one does not need to provide a specific construction/proof for each possible kind of winning condition neither to prove preliminary results on the existence of memoryless strategy.

The paper is organized as follows. In Section 2, we give the main definitions. Section 3 provides a PSpace algorithm to solve one-counter parity games and a DP lower-bound is presented for one-counter reachability games. The consequences of these results are presented in the two last sections: Section 4 considers the μ-calculus model-checking problem while Section 5 focuses on pushdown games. Due to the page limit, missing proofs and extra details can be found in the Appendix.

2 Definitions

An *alphabet* A is a finite set of letters. A^* denotes the set of *finite words* on A and A^ω the set of *infinite words* on A. The empty word is denoted by ε.

Infinite two-player games. Let $G = (V, E)$ denote a (possibly infinite) graph with vertices V and edges $E \subseteq V \times V$. Let $V_{\mathbf{E}} \cup V_{\mathbf{A}}$ be a partition of V between two players Eve and Adam. A *game graph* is such a tuple $\mathcal{G} = (V_{\mathbf{E}}, V_{\mathbf{A}}, E)$. An *infinite two-player game* on a game graph $\mathcal{G}$ is a pair $\mathbb{G} = (\mathcal{G}, \Omega)$, where $\Omega \subseteq V^\omega$ is a *winning condition*.

The players, Eve and Adam, play in $\mathbb{G}$ by moving a token between vertices. A *play* from some vertex v_0 proceeds as follows: the player owning v_0 chooses a successor v_1 such that $(v_0, v_1) \in E$. Then the player owning v_1 chooses a successor v_2 and so on, forever. If at some point one of the players cannot move, she/he looses the play. Otherwise, the play is an infinite word $\Lambda \in V^\omega$ and is won by Eve if and only if $\Lambda \in \Omega$. As one can always add loops on dead-end vertices, and slightly modify the winning condition to make looping plays on some dead-end vertex loosing for the player that controls it, we will assume in the sequel that all plays are infinite. A *partial play* is any prefix of a play.

A strategy for Eve is a function assigning, to any partial play ending in some vertex $v \in V_{\mathbf{E}}$, a vertex v' such that $(v, v') \in E$. Eve *respects a strategy* Φ during some play $\Lambda = v_0 v_1 v_2 \cdots$ if $v_{i+1} = \Phi(v_0 \cdots v_i)$, for all $i \geq 0$ such that $v_i \in V_{\mathbf{E}}$. Finally a strategy for Eve is *winning* from some position $v \in V$, if any play starting from v, where Eve respects Φ, is won by her. A vertex $v \in V$ is winning for Eve if she has a winning strategy from v. Symmetrically, one defines strategies and winning positions for Adam.

A game $\mathbb{G}$ is *determined* if, from any position, either Eve or Adam has a winning strategy. For all games considered in this article one can use Martin's Theorem [15] and conclude that they are determined.

For more details and basic results on games, we refer to [20, 25].

Pushdown games. *Pushdown processes* provide a natural model for programs with recursive procedures. They are like nondeterministic pushdown automata except that they have no input (and therefore no initial state neither final state).

More formally, a *pushdown process* is a tuple $\mathcal{P} = \langle Q, \Gamma, \bot, \Delta \rangle$ where Q is a finite set of states, Γ is a finite stack alphabet that contains a special bottom-of-stack symbol $\bot$ and $\Delta : Q \times \Gamma \to 2^{(\{skip(q), pop(q), push(q,\gamma) \mid q \in Q, \gamma \in \Gamma \setminus \{\bot\}\})}$ is the transition relation. We additionally require that, for all $q \in Q$, $\Delta(q, \bot)$ does not contain any element of the form $pop(q')$.

A *stack* is any word in $St = (\Gamma \setminus \{\bot\})^* \cdot \bot$. A configuration of $\mathcal{P}$ is a pair (q, σ) with $q \in Q$ and $\sigma \in St$. Note that the top stack symbol in some configuration (q, σ) is the leftmost symbol of σ.

Any pushdown process $\mathcal{P}$ induces an infinite graph, called *pushdown graph*, denoted $G = (V, E)$, whose vertices are the configurations of $\mathcal{P}$, and edges are defined by the transition relation Δ, i.e., from a vertex $(p, \gamma\sigma)$ one has edges to:

- $(q, \gamma\sigma)$ whenever $skip(q) \in \Delta(p, \gamma)$.
- (q, σ) whenever $pop(q) \in \Delta(p, \gamma)$.
- $(q, \gamma'\gamma\sigma)$ whenever $push(q, \gamma') \in \Delta(p, \gamma)$.

Consider a partition $Q_{\mathbf{E}} \cup Q_{\mathbf{A}}$ of Q between Eve and Adam. It induces a natural partition $V_{\mathbf{E}} \cup V_{\mathbf{A}}$ of V by setting $V_{\mathbf{E}} = Q_{\mathbf{E}} \times St$ and $V_{\mathbf{A}} = Q_{\mathbf{A}} \times St$. The

resulting game graph $\mathcal{G} = (V_{\mathbf{E}}, V_{\mathbf{A}}, E)$ is called a *pushdown game graph*. Finally, a *pushdown game* is a game played on such a game graph.

The *regular winning conditions* on pushdown games are inherited from the standard acceptance condition for automata on infinite words. The simplest one is the *reachability condition*. Let $F \subseteq Q$ be a set of *final states*, and let V_F be the set of configuration which control state is in F. The reachability condition is the winning condition defined by $\Omega_{reach}(F) = \{v_0 v_1 \cdots \mid \exists v_i \in V_F\}$. Using the notion of final states one can define the *Büchi* condition and its dual winning condition the *co-Büchi conditions*: $\Omega_{Buc}(F) = \{v_0 v_1 \cdots \mid \forall i \geq 0\ \exists j \geq i \text{ s.t. } v_i \in V_F\}$ and $\Omega_{co-Buc}(F) = V^\omega \setminus \Omega_{Buc}(F)$.
Let *col* be a coloring function from Q into a finite set of colors $C \subset \mathbb{N}$. This function is easily extended into a function from V into C by setting $col((q,\sigma)) = col(q)$. The parity condition is the winning condition defined by:
$\Omega_{par} = \{v_0 v_1 \cdots \mid \liminf((col(v_i))_{i\geq 0}) \text{ is even}\}$.

For a parity game played on a pushdown graph, the main question is to decide which player has a winning strategy from some given configuration. It is easily seen that this last question can be reduced to decide the winner for configurations with empty stack. For the general case of parity games played on pushdown graph, this last problem has been fully characterized by Walukiewicz [22].

Theorem 1. *[22] Deciding the winner from some configuration of empty stack in a pushdown parity game is an* ExpTime *problem. Moreover, this problem is* ExpTime*-hard even if the winning condition is a reachability one.*

One-counter games. A *one-counter process* is a special case of a pushdown process $\mathcal{P} = \langle Q, \Gamma, \bot, \Delta \rangle$ where $\Gamma = \{1, \bot\}$ consists of a single stack symbol 1 together with the bottom-of-stack symbol. It therefore corresponds to a finite state machine equipped with a counter that can be test to zero. The notions of *one-counter graphs*, *one-counter game graphs* and *one-counter games* are induced by the one for pushdown processes.

Remark 1. Note that our model of one-counter processes can test whether the stack (equivalently the counter value) is empty (equivalently equals 0). This follows from the fact that in the definition of pushdown processes the bottom-of-stack belongs to the stack alphabet, and hence can be checked as top stack symbol when performing an action.

Theorem 1 implies an ExpTime upper bound to decide the winner in a one-counter parity game. The ExpTime-hard lower bound of Theorem 1 is established by coding the computation of an alternating Turing machine using linear space into a reachability pushdown game. The main idea of the reduction is that the pushdown process is built so that its stack is a description of the prefix of a computation of the Turing machine. Therefore this construction strongly relies on the fact that the stack alphabet is large enough to describe configurations of the Turing machine. Hence this proof cannot be adapted to the case of one-counter game.

A natural question is thus to check whether it is possible, in the special case of one-counter games, to improve the ExpTime upper bound. We positively answer this question in Theorem 2 by providing a PSpace algorithm.

3 Deciding the winner in a one-counter parity game

3.1 Upper bound

Intuition In [11], Kupferman and Vardi have proposed a new approach, based on automata, to solve model-checking problem for pushdown graphs. The main idea was to reduce a model-checking problem to an emptiness problem for a class of tree automata, namely alternating two-way parity tree automata. This technique can then be adapted to solve parity pushdown games [4].

Let us first informally recall the construction of [4], and explain how to simplify it in the special case of one-counter parity games. It is rather standard to consider that the complete infinite tree of arity k is a representation of the set of all finite words on an alphabet of cardinality k. Each node in this tree is labeled by the last letter of the word it represents: hence the word associated to some node is obtained by considering the sequence of labels of the nodes on the path from the root to the current one. Using this fact, a play in a pushdown game can be considered as an (infinite) path in such a tree in which a node encodes the stack content while an extra information describes the control state. As there are finitely many control states, and as the possible moves only depend on the control state and on the top stack symbol (that is the label of the current node), this representation of a play can be seen as a path in the run of some tree automaton on the complete infinite tree of arity k, where k is the size of the pushdown stack alphabet without the bottom-of-stack symbol. This tree automaton can go in both directions in the tree: it goes down to simulate a rule that pushes some new symbol, it goes up to simulate a popping rule and it stay in the same node to simulates a skip rule. For each possible move, the control state has to be updated in accordance with the pushdown transition rules. As we aim to simulate a game, the tree automaton needs to be alternating: existential states are those associated to Eve's states while universal states are those associated to Adam's states. Finally, the acceptance condition is inherited from the winning condition, and is therefore a parity condition. The complete infinite tree of arity k is accepted if and only if Eve has a winning strategy in the pushdown game.

The previous tree automaton works on the complete infinite tree and the arity of this tree is the cardinality of the stack alphabet without the bottom-of-stack symbol. Hence, if we restrict ourselves to one-counter processes instead of general pushdown processes, the arity is equal to 1 and instead of a tree we have to consider a simpler model, namely the infinite word $\perp 1^\omega$. Therefore, it follows that to decide the winner in a one-counter parity game it is sufficient to check emptiness for an alternating two-way parity *word* automaton. In Proposition 2 we will show that emptiness for these word automata can be checked in PSpace and hence it will yield a PSpace procedure to decide the winner in a one-counter parity game (Theorem 2).

Definitions Given a set S of variables, we denote by $\mathcal{B}^+(S)$ the set of positive boolean formulas over S with *true* and *false*. A subset $S' \subseteq S$ *satisfies* a formula in $\mathcal{B}^+(S)$ if this formula is satisfied by the valuation assigning true to every variable in S' and false to every variable in $S \setminus S'$.

An *alternating two-way parity word automaton* $\mathcal{A}$ is a tuple $\langle Q, A, q_{\text{in}}, \delta, col\rangle$, where Q is a finite set of control states, A is a finite input alphabet, $q_{\text{in}} \in Q$ is an initial state, δ is a mapping from $Q \times A$ to $\mathcal{B}^+(Q \times \{-1, 0, 1\})$, and col is a mapping from Q to a finite set of colors $C \subset \mathbb{N}$. An *alternating one-way parity word automaton* corresponds to the special case where $\delta : Q \times A \to \mathcal{B}^+(Q \times \{1\})$.

A *run* of $\mathcal{A}$ on an infinite word $u = a_0 a_1 \cdots \in A^\omega$ is an infinite $(Q \times \mathbb{N})$-labeled tree such that its root is labeled by $(q_{\text{in}}, 0)$, and for every vertex x labeled by some (q, n) with sons labeled by $(q_1, n_1), \ldots, (q_k, n_k)$, the set $\{(q_1, n_1 - n), \ldots, (q_k, n_k - n)\} \subset Q \times \{-1, 0, 1\}$ satisfies $\delta(q, a_n)$. A run is *accepting* if and only if for every infinite branch, the smallest infinitely repeated color is even, where the color of a node labeled by some (q, n) is $col(q)$. Finally, an infinite word is *accepted* if there exists an accepting run on it, and we denote by $L(\mathcal{A})$ the set of all words accepted by $\mathcal{A}$.

The construction Let $\mathcal{C} = \langle Q, \{1, \bot\}, \bot, \Delta\rangle$ be a one-counter process equipped with a partition $Q_{\mathbf{E}} \cup Q_{\mathbf{A}}$ of its control states, and with a coloring function $col : Q \to C$. Let $\mathbb{G}$ be the one-counter parity game induced by the preceding partition and the coloring function col. Let q_{in} be some state in Q. We are interesting in deciding whether $(q_{\text{in}}, \bot)$ is winning for Eve in $\mathbb{G}$.

To solve this problem, instead of using the techniques from [22], that would lead to an ExpTime procedure, we adapt the techniques developed in [11, 4], and note that it reduces our problem to the emptiness problem for alternating two-way parity word automaton.

Let us consider the alternating two-way parity word automaton $\mathcal{A} = \langle Q, \{1, \bot\}, q_{\text{in}}, \delta, col\}$ where the transition function δ is defined by:

- for every $q \in Q_{\mathbf{E}}$ and for every $a \in \{1, \bot\}$, $\delta(q, a)$ equals $[\bigvee_{push(q',1)\in\Delta(q,a)}(q', 1)] \vee [\bigvee_{skip(q')\in\Delta(q,a)}(q', 0)] \vee [\bigvee_{pop(q')\in\Delta(q,a)}(q', -1)]$.
- for every $q \in Q_{\mathbf{A}}$ and for every $a \in \{1, \bot\}$, $\delta(q, a)$ equals $[\bigwedge_{push(q',1)\in\Delta(q,a)}(q', 1)] \wedge [\bigwedge_{skip(q')\in\Delta(q,a)}(q', 0)] \wedge [\bigwedge_{pop(q')\in\Delta(q,a)}(q', -1)]$.

We have the following straightforward proposition.

Proposition 1. *[11, 4] The configuration $(q_{in}, \bot)$ is winning for Eve in $\mathbb{G}$ if and only if $\mathcal{A}$ accepts the infinite word $\bot 1^\omega$.*

Checking whether $\mathcal{A}$ accepts the word $\bot 1^\omega$ is closely related to checking emptiness for a language accepted by an alternating two-way parity word automaton. This problem was studied by Vardi in [21] in the more general setting of two-way alternating parity tree automata, and then this construction was adapted for alternating two-way *Büchi* word automata in [17, 10]. In the case of tree automata, checking emptiness is in ExpTime, while in the case of Büchi word automata the problem is in PSpace. The following proposition, extends this last result to the case of parity acceptance condition.

Proposition 2. *Deciding emptiness for a language accepted by an alternating two-way parity word automaton can be achieved in* PSPACE.

Proof. We only give the main ideas and explain how the construction of [17, 10] is extended to our setting. A complete proof can be found in the Appendix.

Let $\mathcal{A}$ be an alternating two-way parity word automaton. The first step is to build an alternating one-way parity automaton $\mathcal{B}$ such that $L(\mathcal{B}) \neq \emptyset$ if and only if $L(\mathcal{A}) \neq \emptyset$. Moreover the number of control states of $\mathcal{B}$ is polynomial in the number of control states of $\mathcal{A}$. However the size of its alphabet is exponential but note that it is not important for the complexity of emptiness checking.The construction of $\mathcal{B}$ directly follows from the ones in [21, 17, 10]. A precise analysis of the structure of $\mathcal{B}$ is given by the following lemma (which is proved in the Appendix).

Lemma 1. *Let $\mathcal{A} = \langle Q, A, q_{in}, \delta, col\rangle$ be an alternating two-way parity word automaton, let $n = |Q|$ and let d be the number of colors involved in the parity condition. Then there exists an alternating one-way parity word automaton $\mathcal{B}$ such that $L(\mathcal{B}) \neq \emptyset$ if and only if $L(\mathcal{A}) \neq \emptyset$. Moreover, $L(\mathcal{B}) = L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$, where $\mathcal{B}_1$ is an alternating one-way automaton without acceptance condition (every run, when exists, is accepting) and has $\mathcal{O}(nd)$ states, and $\mathcal{B}_2$ is a purely universal (its transition function takes value into the boolean formulas made only of conjunctions) one-way parity automaton (with d colors) and has $\mathcal{O}(nd)$ states.*

Let n_1 and n_2 be the respective sizes of the set of control states of $\mathcal{B}_1$ and $\mathcal{B}_2$. As $\mathcal{B}_2$ is purely universal, its dual automaton $\overline{\mathcal{B}_2}$ is a non deterministic parity automaton using d colors and having $\mathcal{O}(n_2)$ states. It is then standard to build a nondeterministic Büchi automaton $\overline{\mathcal{B}'_2}$ that recognizes the same language than $\overline{\mathcal{B}_2}$ (that is the complement of $L(\mathcal{B}_2)$) and having $\mathcal{O}(n_2 d)$ states (see [13] for instance). Dualizing $\overline{\mathcal{B}'_2}$ yields a purely universal co-Büchi automaton $\mathcal{B}'_2$ with $\mathcal{O}(n_2 d)$ states and such that $L(\mathcal{B}'_2) = L(\mathcal{B}_2)$.

Now, the *intersection* of $\mathcal{B}_1$ and $\mathcal{B}'_2$ provides an alternating co-Büchi automaton $\mathcal{B}'$ with $\mathcal{O}(n_2 d + n_1) = \mathcal{O}(n^2 d)$ states that recognizes the language $L(\mathcal{B}_1) \cap L(\mathcal{B}'_2) = L(\mathcal{B}_1) \cap L(\mathcal{B}_2) = L(\mathcal{B})$. As checking emptiness for an alternating co-Büchi automaton can be achieved in PSPACE (see [12] for instance), we conclude that one can check whether $L(\mathcal{A})$ is empty in PSPACE.

□

Propositions 1 and 2 directly imply the following theorem.

Theorem 2. *Deciding the winner in a one-counter parity game can be done in* PSPACE.

3.2 Lower bound

In this section, we give a lower bound for the problem of deciding the winner in a one-counter reachability game. Due to the symmetry of the problem, the lower bound should be robust under complementation: we provide such a lower

bound, namely DP-hardness. Note that DP-hardness is a rather standard lower bound for problems related to one-counter process, *e.g.* the EF model-checking problem for one-counter processes [8].

A language L is in the complexity class DP if and only if there are two languages $L_1 \in$ NP and $L_2 \in$ co-NP such that $L = L_1 \cap L_2$.

The SAT-UNSAT problem is the following one: given two Boolean formulas ψ_1 and ψ_2, both in conjunctive normal form with three literals per clause, decide whether φ_1 is satisfiable and φ_2 is not. It is rather immediate to prove that SAT-UNSAT is DP-complete [16].

Let us first explain how to polynomially reduce 3-SAT to decide the winner in a one-counter reachability game. Let $X = \{x_1, \ldots, x_k\}$ be a set of variables and let ψ be some Boolean formula in conjunctive normal form with 3 literals per clause. Let denote $\psi = C_1 \wedge C_2 \wedge \cdots \wedge C_h$, where $C_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ for all $i = 1, \ldots, h$ with $l_{i,j} \in \{x, \overline{x} \mid x \in X\}$, for $j = 1, 2, 3$.

For every $i \geq 1$, let ρ_i denote the i-th prime number. A valuation of X is a mapping from X into $\{0, 1\}$, that is a tuple in $\{0, 1\}^k$. Let $\tau : \mathbb{N} \to \{0, 1\}^k$ be the function defined by $\tau(n) = (b_1, b_2, \ldots, b_k)$ where $b_j = 0$ if $n = 0 \mod \rho_j$ and $b_j = 1$ otherwise. The Chinese remainder lemma implies that τ is surjective.

Consider now the following informal game. Eve chooses some integer n encoding a valuation that she claims to satisfy ψ. Then Adam picks a clause C_i that he claims not to be satisfied by the preceding valuation. Eve contests by giving a literal of C_i that she claims to be evaluated to true by the preceding valuation. Finally Adam checks whether this literal is evaluated to true: if it is the case, then Eve wins, otherwise Adam does. It is then easily seen that Eve has a winning strategy if and only if ψ is satisfiable.

This game can be encoded into a one-counter reachability game. For the first step, Eve increments the counter until it equals n. For the second step, Adam indicates the clause by changing the control state. In the third step, Eve indicates the literal by changing the control state. Finally, Adam check whether the literal evaluates to true by decrementing the counter while performing a modulo ρ_k counting, where the literal was x_k or $\overline{x_k}$.

Now, if one wants to reduce SAT-UNSAT, it suffices to add a preliminary step to the previous game. Let (ψ_1, ψ_2) be the instance of SAT-UNSAT. First Adam picks ψ_1 or ψ_2. In the first case Eve and Adam play the previous game. In the second case, they play the dual game where Adam is now the one that has to provide a valuation for ψ_2, and where Eve wins if and only if ψ_2 is not satisfiable. Eve wins the main game if and only if she can win both sub-games, that is if and only if ψ_1 is satisfiable while ψ_2 is not. Hence, we have the following result (a detailed proof can be found in the appendix).

Theorem 3. *Deciding the winner in a one-counter reachability game is a* DP*-hard problem.*

Remark 2. An alternative proof for this result is the following one: consider the EF model-checking problem for one-counter automata. In [8] this problem is shown to be DP-hard. One can then easily reduce it to decide the winner in a

one-counter reachability game. Nevertheless, we think that the proof we gave for Theorem 3 is more intuitive and better here as it is self-contained.

4 Model-checking propositional μ-calculus against one-counter trees

In this section we rephrase Theorem 3 in the framework of propositional μ-calculus model-checking problem for one-counter trees. An important consequence is that it improves from ExpTime to PSpace the best complexity bound known for this problem.

Propositional μ-calculus is a very powerful fix point logic that allows to specify a large class of properties of (non-terminating) systems. Moreover, many important temporal logic were shown to be fragments of μ-calculus. For definitions and results on μ-calculus, we refer to [1].

Models of μ-calculus formulas are transitions systems, that is graphs equipped with functions that assign to any propositional constant the set of vertices where it holds. The μ-calculus model-checking problem is to decide, for a given model $\mathcal{M}$, a state s of $\mathcal{M}$, and a μ-calculus formula φ, whether φ holds in s. In the sequel we are interested in the special case where $\mathcal{M}$ is the unfolding of a one-counter graph, called a *one-counter tree*.

A *standard* technique to solve a μ-calculus model-checking problem is to construct a parity game in which Eve has a winning strategy if and only if the model satisfies the formula. The game graph is obtained by considering the synchronized product of a finite game graph, representing the formula φ, with the model $\mathcal{M}$. This idea was first used in [6] for finite transition systems, and was then adapted in [22] for pushdown trees (see also [24] for a general presentation of the technique). In the case of pushdown trees, an important point to note is that in the synchronized product, the stack alphabet remains unchanged (the product is done in the control states). Hence, using the same construction for one-counter trees reduces the μ-calculus model-checking problem for a one-counter tree to solve a one-counter parity game. Conversely, it follows from [22] that solving a one-counter parity game reduces to a μ-calculus model checking problem. As both reductions are polynomial, we obtain the following consequence of Theorem 2.

Theorem 4. *The propositional μ-calculus model-checking problem for one-counter trees can be solved in* PSpace *and is* DP*-hard.*

Remark 3. Note that the DP-hardness was already known, as it is a consequence of the DP-hardness of the model-checking problem for the branching-time temporal logic EF [8] which is a fragment of the propositional μ-calculus.

5 Application to pushdown games

In section 2 we have defined the regular winning conditions. Nevertheless, when considering pushdown games, non-regular winning conditions arise naturally. In

particular, one can require conditions on how the stack height evolves during the play. For some configuration $v = (q, \sigma\bot)$ in a pushdown graph, let $sh(v) = |\sigma|$ denote the stack height in v. The *unboundedness condition* requires that the stack height is not bounded. Its dual condition is the *boundedness condition*. Both conditions are formally defined as follows:

- $\Omega_{Ubd} = \{v_0 v_1 \cdots \mid \limsup((sh(v_i)_{i\geq 0}) = \omega\}$.
- $\Omega_{Bd} = \{v_0 v_1 \cdots \mid \exists B \geq 0 \text{ s.t. } sh(v_i) < B \ \forall i \geq 0\}$.

If we replace the lim sup by a lim in the definition of the unboundedness condition then we obtain the *strict unboundedness condition* which enforces the stack height to converge to infinity. Its dual version, the *repeating condition* requires that some stack height (equivalently, some vertex) is infinitely often visited. Both conditions are formally defined as follows:

- $\Omega_{StUbd} = \{v_0 v_1 \cdots \mid \lim((sh(v_i))_{i\geq 0}) = \omega\}$.
- $\Omega_{Rep} = \{v_0 v_1 \cdots \mid \exists B \geq 0 \text{ s.t. } \forall j \geq 0 \ \exists i \geq j \text{ s.t. } sh(v_i) = B\}$.

These four winning conditions will be designated as *stack conditions*. Pushdown games with stack conditions are known to be decidable in ExpTime [5, 19, 3, 7, 18]. In the sequel we consider winning conditions that are a Boolean combination of stack conditions with a parity condition. For instance the winning condition $\Omega_{par} \cap \Omega_{Ubd} \cap \Omega_{Rep}$ requires that the smallest infinitely visited color has to be even and that arbitrary large stack height occurs while some level is infinitely repeated. Note that the winning condition $\Omega_{Ubd} \cap \Omega_{Rep}$ was already mentioned in [5] and can be rephrased as: there exists infinitely many vertices that are infinitely often visited during the play. Decidability of pushdown games with this winning condition was open and is a consequence of the main result of this section.

Games equipped with winning conditions that are a Boolean combination of a parity condition and of an unboundedness condition have been shown to be decidable in [3] when restricting to Büchi conditions and in [7] for the general case. For all these games an ExpTime-complete complexity bound has been provided.

The main result of this section is to provide an ExpSpace procedure to solve these games and more generally to solve the ones equipped with a Boolean combination of a parity condition and of stack conditions. Even if the complexity bound may not be optimal here, the results are more general and the presentation and proof techniques are much simpler and unified. Indeed, the construction is a generalization of the one for parity condition, and it *separates* all conditions involved in the Boolean combination, which allows to reason independently on these conditions and leads to a very flexible construction. Moreover, no preliminary result on memoryless strategy is needed, while it was the case in [7].

From now on, we fix a pushdown process $\mathcal{P} = \langle Q, \Gamma, \bot, \Delta \rangle$, a partition $Q_{\mathbf{E}} \cup Q_{\mathbf{A}}$ of its control states and a coloring function $col : Q \to \{0, \ldots, d\}$. Let $\mathcal{G}$ be the corresponding game graph, and let Ω_{par} be the parity condition induced by col.

For an infinite play $\Lambda = v_0 v_1 \cdots$, let $Steps_\Lambda$ be the set of indices of positions where no configuration of strictly smaller stack height is visited later in the play. More formally, $Steps_\Lambda = \{i \in \mathbb{N} \mid \forall j \geq i \ sh(v_j) \geq sh(v_i)\}$. Note that $Steps_\Lambda$ is always infinite and hence induces a factorization of the play Λ into finite pieces.

For all pair $(i,j) \in Steps_\Lambda$, with $i \neq j$ and such that there is no $k \in Steps_\Lambda$ such that $i < k < j$, we define $mcol(i,j) = \min\{col(v_k) \mid i \leq k \leq j\}$ and

$$kind(i,j) = \begin{cases} S & \text{if } sh(v_j) = sh(v_i) + 1 \\ (B,h) & \text{if } sh(v_j) = sh(v_i) \text{ and } h = \max\{sh(v_k) - sh(v_i) \mid i \leq k \leq j\} \end{cases}$$

In the factorization induced by $Steps_\Lambda$, a factor $v_i \cdots v_j$ will be called a *bump* of height h if $kind(i,j) = (B,h)$, and will be called a *Stair* if $kind(i,j) = S$.

For any play Λ with $Steps_\Lambda = \{n_0 < n_1 < \cdots\}$, one can define two sequences $(mcol_i^\Lambda)_{i\geq 0} \in \mathbb{N}^\mathbb{N}$ and $(kind_i^\Lambda)_{i\geq 0} \in (\{S\} \cup (\{B\} \times \mathbb{N}))^\mathbb{N}$ defined by $mcol_i^\Lambda = mcol(n_i, n_{i+1})$ and $kind_i^\Lambda = kind(n_i, n_{i+1})$.

These sequences fully characterize the parity conditions and the stack conditions.

Proposition 3. *For a play Λ the following equivalences hold*

1. *$\Lambda \in \Omega_{par}$ iff $\liminf((mcol_i^\Lambda)_{i\geq 0})$ is even.*
2. *$\Lambda \in \Omega_{Ubd}$ iff either $\{kind_i^\Lambda \mid i \geq 0\}$ contains (B,h) for any $h \geq 0$, or S appears infinitely often in $(kind_i^\Lambda)_{i\geq 0}$.*
3. *$\Lambda \in \Omega_{StUbd}$ iff S appears infinitely often in $(kind_i^\Lambda)_{i\geq 0}$.*

By dualization, one obtains similar characterizations for Ω_{Bd} and Ω_{Rep}.

The main idea used in [22] to solve parity pushdown game is to build a parity game played on an exponentially larger finite graph with the same number of colors. This new game *simulates* the pushdown game, in the sense that the sequences of visited colors during a correct simulation play are exactly the sequences $(mcol_i^\Lambda)_{i\geq 0}$ for plays Λ in the original pushdown game. Moreover, a play in which a player does not correctly simulate the pushdown game is loosing for that player. From this construction follows the ExpTime upper bound.

Let us explain how to extend this technique to handle stack conditions. When considering the strict unboundedness condition, it is sufficient to detect in the simulation game of [22] whether the currently simulated factor is a stair or a bump. Therefore, this construction can be easily adapted to reduce a pushdown games with a strict unboundedness winning condition to a Büchi game played on a finite game graph (the Büchi condition enforcing to simulate an infinite number of stairs) [19, 18]. Nevertheless, for bumps, one cannot express any property on their height.

Consider the unboundedness condition. A play satisfies it either if it satisfies the strict unboundedness condition (which can be encoded by a Büchi condition) or if some stack height is infinitely often repeated and arbitrarily high bumps appear. For this last case, it would be sufficient to detect whether a bump is the highest one since the play is on the current stack level: indeed in a non strictly

unbounded game, this happens infinitely often if and only if arbitrarily high bumps occur during the play. In order to detect this phenomena, we enrich the finite game graph of [22] with a counter that is incremented whenever a bump higher than the counter value is simulated, and that is decremented (mainly for technical reasons) when a stair is simulated: if finitely many stair are simulated, the counter is incremented infinitely often if and only if arbitrarily high bumps occur on some fixed level (the one reached after the last stair). Therefore the unboundedness condition is simulated in this new one-counter game by requiring that either one simulates infinitely many stairs or the counter is infinitely often incremented.

Hence, when considering as winning condition a Boolean combination of a parity condition and of stack conditions, one gets a reduction to a one-counter game equipped with a simple combination of parity, Büchi and co-Büchi condition that can easily be expressed as a parity condition by slightly modifying the underlying one-counter process.

Before providing a description of the one-counter game graph $\widetilde{\mathcal{G}}$, let us consider the following informal description of this simulation game. We aim at simulating a play in the pushdown game from some initial vertex $(p_{in}, \bot)$. In $\widetilde{\mathcal{G}}$ we keep track of only the control state and the top stack symbol of the simulated configuration, and we maintain a counter κ. The interesting case is when it is in a control state p with top stack symbol α, and the player owning p wants to push a letter β onto the stack and change control state to q. For every strategy of Eve, there is a certain set of possible (finite) continuations of the play that will end with popping β from the stack. We require Eve to declare a vector $\overrightarrow{S} = ((S_0^-, S_0^+), \ldots, (S_d^-, S_d^+))$ of $(d+1)$ pairs in $(2^Q)^2$, where S_i^- (*resp.* S_i^+) is the set of all states the game can be in after popping of β along these plays where in addition the stack height in the induced bump is strictly smaller (*resp.* equal or larger) than κ and the smallest visited color while β was on the stack is i.
Adam has two main choices. He can continue the game by pushing β onto the stack and update the state (we call this a *pursue* move). Otherwise, he can pick a set $S_i^\star$ (for $\star = -$ or $+$) and a state $s \in S_i^\star$, and continue the simulation from that state s (we call this a *jump* move). If he does a pursue move, then he remembers the vector $\overrightarrow{S}$ claimed by Eve and the counter κ is decreased; if later on a pop transition is simulated, the play stops and Eve wins if and only if the resulting state is in $S_\theta^\star$ where θ is the smallest color seen in the current level (this information is encoded in the control state, reset after each pursue move and updated after each jump move) and $\star = +$ if $\kappa = 0$ and $\star = -$ otherwise. If Adam does a jump move to a state s in $S_i^\star$, the currently stored value for θ is updated to $\min(\theta, i, col(s))$, which is the smallest color seen since the current stack level was reached, and if $\star = +$, the currently stored vector $\overrightarrow{R} = ((R_0^-, R_0^+), \ldots, (R_d^-, R_d^+))$ is changed to $\overrightarrow{R^+} = ((R_0^+, R_0^+), \ldots, (R_d^+, R_d^+))$ and κ is incremented.

Therefore the main vertices of the one-counter game graph are configurations of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ and they are controlled by the player that control p.

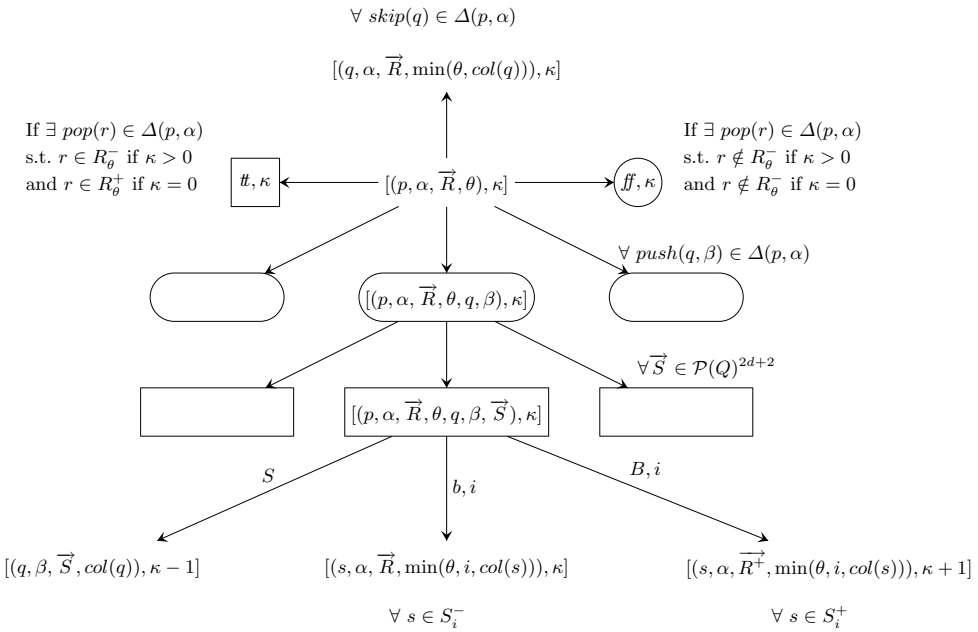

Fig. 1. Local structure of $\widetilde{\mathcal{G}}$.

Intermediate configurations are used to handle the previously described intermediate steps. The local structure is given in Figure 1 (circle vertex are those controlled by Eve). Two special control states *tt* and *ff* are used to simulate pop moves. This game graph is equipped with a coloring function on the vertices and on the edges: vertices of the form $[(p,\alpha,\overrightarrow{R},\theta),\kappa]$ have color $col(p)$, edges leaving from a vertex $[(p,\alpha,\overrightarrow{R},\theta,q,\beta,\overrightarrow{S}),\kappa]$ have two colors, one in $\{S,b,B\}$ (the color is S if the edge simulates a stair, b if it simulates a bump smaller than κ and B otherwise) and one in $\{0,\ldots,d\}$ if it simulates a bump (the color is θ is the bump has color θ). It is easily seen that intermediate control states can be used to have only colors on vertices. A precise description of the graph is given in the Appendix.

The winning condition for the game played on $\widetilde{\mathcal{G}}$ depends on the winning condition considered in the pushdown graph. If the winning condition is of the form $\psi(\Omega_1,\ldots,\Omega_k)$ for a Boolean formula ψ, the winning condition on $\widetilde{\mathcal{G}}$ will be

$\psi(\widetilde{\Omega_1}, \ldots, \widetilde{\Omega_k})$, where

$$\widetilde{\Omega} = \begin{cases} \Omega_{par} & \text{if } \Omega = \Omega_{par} \\ \Omega_{Buc}(\{S, B\}) & \text{if } \Omega = \Omega_{Ubd} \\ \Omega_{co-Buc}(\{S, B\}) & \text{if } \Omega = \Omega_{Bd} \\ \Omega_{Buc}(\{S\}) & \text{if } \Omega = \Omega_{StUbd} \\ \Omega_{co-Buc}(\{S\}) & \text{if } \Omega = \Omega_{Rep} \end{cases}$$

Our main result is the following.

Theorem 5. *A configuration $(p_{in}, \bot)$ is winning for Eve in $\mathbb{G} = (\mathcal{G}, \psi(\Omega_1, \ldots, \Omega_k))$ if and only if $[(p_{in}, \bot, ((\emptyset, \emptyset), \ldots, (\emptyset, \emptyset), col(p_{in})), 0]$ is winning for Eve in $\widetilde{\mathbb{G}} = (\widetilde{\mathcal{G}}, \psi(\widetilde{\Omega_1}, \ldots, \widetilde{\Omega_k}))$. Hence, deciding the winner in such a pushdown game can be done in* ExpSpace.

6 Conclusion

Refining the techniques from [11, 4], we have obtained a PSpace algorithm to decide the winner in a one-counter parity game. As this problem was shown to be DP-hard, a remaining question is whether the complexity gap can be reduced.

As a corollary of our main result, we have improved the best known upper bound for the μ-calculus model-checking problem against one-counter processes.

We have shown how to use one-counter parity games to solve pushdown games equipped with winning conditions requiring both regular properties and stack height properties. We briefly mention here an extension of our result. In [14] pushdown games equipped with visibly pushdown winning conditions were considered. Such winning conditions capture all regular properties and several natural non-regular properties. In this setting, one can express the strict unboundedness condition but not the unboundedness one. The technique to solve these games is similar to the one for parity pushdown games: it uses a reduction to a parity game played on a finite game graph. One can easily show that the techniques of Section 5 can be adapted to solve pushdown games equipped with a winning condition combining a visibly pushdown condition with an unboundedness condition.

Acknowledgments. I would like to acknowledge the anonymous referees for their helpful suggestions and remarks.

References

1. A. Arnold and D. Niwiński. *Rudiments of mu-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2001.
2. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controlers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

3. A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with the unboundedness and regular conditions. In *Proceedings of FST&TCS'03*, volume 2914 of *LNCS*, pages 88–99. Springer, 2003.
4. T. Cachat. Two-way tree automata solving pushdown games. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 303–317. Springer, 2002.
5. T. Cachat, J. Duparc, and W. Thomas. Solving pushdown games with a Σ_3-winning condition. In *Proceedings of CSL'02*, volume 2471 of *LNCS*, pages 322–336. Springer, 2002.
6. E. A. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of mu-calculus. In *Proceedings of CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.
7. H. Gimbert. Parity and exploration games on infinite graphs. In Springer, editor, *Proceedings of CSL'04*, volume 3210 of *LNCS*, pages 56–70, 2004.
8. P. Jančar, A. Kučera, F. Moller, and Zdeněk Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Information and Computation*, 188:1–19, 2004.
9. A. Kucera. Efficient verification algorithms for one-counter processes. In Springer, editor, *Proceedings of ICALP'00*, volume 1853 of *LNCS*, pages 317–328, 2000.
10. O. Kupferman, N. Piterman, and M. Vardi. Extended temporal logic revisited. In Springer, editor, *Proceedings of Concur'01*, volume 2154 of *LNCS*, pages 519–535, 2001.
11. O. Kupferman and M. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proceedings of CAV'00*, volume 1855 of *LNCS*, pages 36–52. Springer, 2000.
12. O. Kupferman and M. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
13. C. Löding. Methods for the transformation of ω-automata: Complexity and connection to second order logic. Diplomata thesis, Christian-Albrechts-University of Kiel, 1998.
14. C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proceedings of FST&TCS'04*, volume 3328 of *LNCS*, pages 408–420. Springer, 2004.
15. D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102(363–371), 1975.
16. C. Papadimitriou. *Complexity Theory*. Addison Wesley, 1994.
17. N. Piterman. Extending temporal logic with ω-automata. Master's thesis, The Weizmann Institute of Science, 2000.
18. O. Serre. *Contribution à l'étude des jeux sur des graphe de processus à pile*. PhD thesis, Université Paris VII, November 2004.
19. O. Serre. Games with winning conditions of high Borel complexity. In *Proceedings of ICALP'04*, volume 3142 of *LNCS*, pages 1150–1162. Springer, 2004.
20. W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of STACS 1995*, volume 900 of *LNCS*, pages 1–13. Springer, 1995.
21. M. Vardi. Reasoning about the past with two-way automata. In *Proceedings of ICALP 1998*, volume 1443 of *LNCS*, pages 628–641. Springer, 1998.
22. I. Walukiewicz. Pushdown processes: games and model checking. In *Proceedings of CAV'96*, volume 1102 of *LNCS*, pages 62–74. Springer, 1996.
23. I. Walukiewicz. Pushdown processes: games and model checking. *Information and Computation*, 157:234–263, 2000.
24. I. Walukiewicz. A landscape with games in the background. In *Proceeding of LICS'04*, pages 356–366. IEEE Computer Society, 2004.

25. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Appendix

A Proofs of Section 3

A.1 Proof of Proposition 1

Proposition 1. *The configuration $(q_{in}, \bot)$ is winning for Eve in $\mathbb{G}$ if and only if $\mathcal{A}$ accepts the infinite word $\bot 1^\omega$.*

Proof. Let us assume that Eve has a winning strategy from $(q_{\text{in}}, \bot)$ in $\mathbb{G}$. This strategy can be seen as an infinite tree where each branch represents a possible play in which Eve follows her winning strategy. It is straightforward to note that this tree is an accepting run of $\mathcal{A}$ on $\bot 1^\omega$.

Conversely, an accepting run of $\mathcal{A}$ provides a winning strategy for Eve from $(q_{\text{in}}, \bot)$ in $\mathbb{G}$. □

A.2 Proof of Lemma 1

Lemma 1. *Let $\mathcal{A} = \langle Q, A, q_{in}, \delta, col\rangle$ be an alternating two-way parity word automaton, let $n = |Q|$ and let d be the number of colors involved in the parity condition. Then there exists an alternating one-way parity word automaton $\mathcal{B}$ such that $L(\mathcal{B}) \neq \emptyset$ if and only if $L(\mathcal{A}) \neq \emptyset$. Moreover, $L(\mathcal{B}) = L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$, where $\mathcal{B}_1$ is an alternating one-way automaton without acceptance condition (every run, when exists, is accepting) and has $\mathcal{O}(nd)$ states, and $\mathcal{B}_2$ is a purely universal (its transition function takes value into the boolean formulas made only of conjunctions) one-way parity automaton (with d colors) and has $\mathcal{O}(nd)$ states.*

Proof. The proof is a simple adaptation of the general one for alternating two-way parity automata on *trees* [21]. It can also be seen as a generalization of the one for alternating two-way *Büchi* automata on words [17, 10] (which was itself a simplification of the one in [21]). The presentation of this proof is based on an adaptation of the one in [17] for the Büchi acceptance condition.

We start with some definitions.

Definition A1 *A* strategy *for $\mathcal{A}$ is a mapping $\tau : \mathbb{N} \to 2^{Q\times\{-1,0,1\}\times Q}$. For each subset $\zeta \subseteq Q \times \{-1, 0, 1\} \times Q$, we set $state(\zeta) = \{q \in Q \mid (q, i, q') \in \zeta,\ q' \in Q,\ i = -1, 0, 1\}$. The strategy τ* is on a word *$u = a_0a_1 \cdots$ if $q_{in} \in state(\tau(0))$, and for all $i \geq 0$ and all state $q \in state(\tau(i))$, $\{(q', c) \mid (q, c, q') \in \tau(i)\}$ satisfies $\delta(q, a_i)$.*

Definition A2 *A* path *in a strategy τ is a finite or infinite sequence $(0, q_{in})$,(i_1, q_1), $(i_2, q_2)\ldots$ of pairs from $\mathbb{N} \times Q$ such that, either the path is infinite and for all $j \geq 0$, there exists $c_j \in \{-1, 0, 1\}$ such that $(q_j, c_j, q_{j+1}) \in \tau(i_j)$ and*

$i_{j+1} = i_j + c_j$; *or the path is a finite sequence* $(0, q_{in}), (i_1, q_1) \dots (i_m, q_m)$ *and for all* $0 \leq j < m$, *there exists* $c_j \in \{-1, 0, 1\}$ *such that* $(q_j, c_j, q_{j+1}) \in \tau(i_j)$ *and* $i_{j+1} = i_j + c_j$, *and* $\delta(q_m, a_m) = true$. *A path* $(0, q_{in}), (i_1, q_1), (i_2, q_2) \dots$ *is* accepting *if it is finite, or it is infinite and satisfies the parity condition,* i.e. $\liminf(col(q_i)_{i \geq 1})$ *is even. Finally, a strategy* τ *is* accepting *if all paths in* τ *are accepting.*

The following proposition was proved by Vardi in [21].

Proposition A3 *[21] An alternating two-way parity automaton accepts a word if and only if it has an accepting strategy on that word.*

We now introduce the notion of annotation that expresses what kind of loops may appears in a path in some strategy τ.

An *annotation* of the strategy τ (on the word u) for $\mathcal{A}$ is a mapping $\eta : \mathbb{N} \to 2^{Q \times \{1,\dots,d\} \times Q}$. The meaning of a tuple (q, c, q') is that there is a loop, respecting τ, that starts with control state q reads some letters and come back to the original letter with state q'. Moreover, the smallest color visited in the meantime (without considering the initial state q) is c. An annotation has to satisfy the following properties for all $i \in \mathbb{N}$.

1. If $(q, c, q') \in \eta(i)$ and $(q', c', q'') \in \eta(i)$, then $(q, \min(c, c'), q'') \in \eta(i)$: if there is a loop of minimal color c followed by a loop of minimal color c' then there is a loop of minimal color $\min(c, c')$.
2. If $(q, 0, q') \in \tau(i)$ then $(q, col(q'), q') \in \eta(i)$: if there is a trivial loop, then its color is the one of the last state.
3. If $i > 0$, $(q, -1, q') \in \tau(i)$, $(q', c, q'') \in \eta(i-1)$ and $(q'', 1, q''') \in \tau(i-1)$, then $(q, \min(col(q'), c, col(q''')), q''') \in \eta(i)$: we go left, loop, and go back right.
4. If $(q, 1, q') \in \tau(i)$, $(q', c, q'') \in \eta(i+1)$ and $(q'', -1, q''') \in \tau(i+1)$, then $(q, \min(col(q'), c, col(q''')), q''') \in \eta(i)$: we go right, loop and go back left.
5. If $i > 0$, $(q, -1, q') \in \tau(i)$ and $(q', 1, q'') \in \tau(i-1)$, then $(q, \min(col(q'), col(q'')), q'') \in \eta(i)$: we go left and go back right immediately.
6. If $(q, 1, q') \in \tau(i)$ and $(q', -1, q'') \in \tau(i+1)$, then $(q, \min(col(q'), col(q'')), q'') \in \eta(i)$: we go right and go back left immediately.

A *downward path* in an annotation η of a strategy τ on a word $u = a_0 a_1 \cdots$ is a sequence of tuples $(i_1, q_1, t_1), (i_2, q_2, t_2), \dots$, where for all $j \geq 1$, $i_j \in \mathbb{N}$, $q_j \in Q$ and t_j is either an element of $\tau(i_j)$ or $\eta(i_j)$, and the following holds.

- If t_j is an element of $\tau(i_j)$, $t_j = (q_j, 1, q_{j+1})$ and $i_{j+1} = i_j + 1$. In this case, we say that the color of (i_j, q_j, t_j) is $col(q_{j+1})$.
- If t_j is an element of $\eta(i_j)$, $t_j = (q_j, c, q_{j+1})$ and $i_{j+1} = i_j$. In this case, we say that the color of (i_j, q_j, t_j) is c.

A downward path can be finite if it ends either by a tuple (i_m, q_m, t_m) such that $t_m = (q, c, q)$ (it ends by a loop) or by a tuple (i_m, q_m, t_m) such that $\delta(q_m, a_{i_m}) = true$. A finite downward path is *accepting* either if we are in the first case with c being even, or if we are in the second case. Finally an infinite

downward path is accepting if it satisfies the parity condition, that is if the smallest color (in the previously defined way) appearing infinitely often is even.

Finally, an annotation η is *accepting* if every downward path in η is accepting. We have the following characterization.

Proposition A4 *[21] An alternating two-way parity automaton accepts a word if and only if it has a strategy on that word and an accepting annotation of the strategy.*

The automaton $\mathcal{B}$ will read a word on the alphabet A together with a strategy and an annotation of it. It accepts if and only if the annotation is a correct one and is accepting.

We introduce two new alphabets: $\Delta_Q^s = 2^{Q\times\{-1,0,1\}\times Q}$ (for strategies) and $\Delta_Q^a = 2^{Q\times\{1,\ldots,d\}\times Q}$ (for annotations). Finally, let us define the alphabet $A' = A \times \Delta_Q^s \times \Delta_Q^a$, and the three natural projections p_1, p_2 and p_3 from A' on A, Δ_Q^s and Δ_Q^a. These projections define morphisms from A'^* on A^*, ${\Delta_Q^s}^*$ and ${\Delta_Q^a}^*$. Finally, the automaton $\mathcal{B}$ is the "intersection" of two automata $\mathcal{B}_1$ and $\mathcal{B}_2$. On some input u, the automaton $\mathcal{B}_1$ checks that $p_3(u)$ is an annotation of the strategy $p_2(u)$, while $\mathcal{B}_2$ checks that all downward paths in it are accepting. A crucial point in the sequel will be that $\mathcal{B}_1$ has no acceptance condition (it can just be blocked to reject), and that $\mathcal{B}_2$ is purely universal.

Description of $\mathcal{B}_1$: In order to check that $p_2(u)$ is a strategy on u, and in order to check the two first conditions for $p_3(u)$ to be an annotation of $p_2(u)$, it suffices to verify local conditions, and the input alphabet A' can thus be restricted in such a way that these local conditions are always satisfied. We assume in the sequel that it is the case and we only deal with the conditions 3 to 6 for $p_3(u)$.

Let $\mathcal{B}_1 = \langle Q_1, A', \{q_{\text{in}}^1\}, \delta_1, col_1\rangle$ where Q_1 consists of the following states.

- Two states $\{q_{\text{in}}^1, q_1^1\}$ that are used to initiate and propagate the verification of various conditions. The state q_{in}^1 is used to verify that q_{in} is in $state(\zeta)$ if the input letter is (a, ζ, μ). The state q_1^1 is used to recursively check the other conditions for the rest of the input word.
- A set $\{C\} \times Q \times \{\in, \notin\}$ of states is used to check the consecution of the strategy, that is if there is a state $(q, 1, q')$ in the current strategy, then there must be a state $(q', \_, \_)$ in the next strategy, and if there is no state of the form $(q, \_, \_)$ in the current strategy, there must not be a state $(q', -1, q)$ in the next strategy.
- A set $\{A\} \times Q \times \{1, \ldots, d\} \times Q \times \{\in, \notin\}$ of states that represent the tuples (q, c, q') of the annotation that must belong $(\in)$ or not $(\notin)$ to the third component of the current input letter.
- A set $\{S\} \times Q \times \{-1\} \times Q \times \{\notin\}$ of states that represent the tuples $(q, -1, q')$ of the strategy that must not be in the second component of the current input letter.

The coloring function here is unused and can thus be defined to be constant and equal to 2 (hence every run, when exists, is accepting).

The transition function δ_1 is defined as follows.

- $\delta_1((C, q, \in), (a, \zeta, \mu))$ equals $true$ if $q \in state(\zeta)$ or if $\delta(q, a) = true$, and equals $false$ otherwise.
- $\delta_1((C, q, \notin), (a, \zeta, \mu))$ equals $false$ if there exists some q' such that $(q', -1, q) \in \zeta$, and equals $true$ otherwise.
- $\delta_1((A, q_1, c, q_2, \in), (a, \zeta, \mu))$ equals $true$ if $(q_1, c, q_2) \in \mu$, and equals $false$ otherwise.
- $\delta_1((A, q_1, c, q_2, \notin), (a, \zeta, \mu))$ equals $true$ if $(q_1, c, q_2) \notin \mu$, and equals $false$ otherwise.
- $\delta_1((S, q_1, i, q_2, \notin), (a, \zeta, \mu))$ equals $true$ if $(q_1, i, q_2) \notin \zeta$, and equals $false$ otherwise.
- Let set $consec(a, \zeta) = \{q \in Q \mid q \notin state(\zeta) \text{ and } \delta(q, a) \neq true\}$, for every $a \in A$ and every $\zeta \in Q \times \{-1, 0, 1\} \times Q$. The consecution of the strategy is expressed by the formula $\Upsilon^{cons}(a, \zeta, \mu) = \bigwedge_{q \in consec(a,\zeta)} (c, q, \notin) \wedge \bigwedge_{(q', 1, q) \in \zeta} (c, q, \in)$.
- To verify condition 3, we set $\varphi_1 = (q, -1, q') \in \tau(i)$, $\varphi_2 = (q', c, q'') \in \eta(i-1)$, $\varphi_3 = (q'', 1, q''') \in \tau(i-1)$ and $\varphi_4 = (q, \min(c, col(q'), col(q''')), q''') \in \eta(i)$. The condition is expressed by the formula $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \Rightarrow \varphi_4$, which is equivalent to $\varphi_2 \wedge \varphi_3 \Rightarrow \varphi_4 \vee \neg\varphi_1$. Thus we have the expression $\Upsilon^3(a, \zeta, \mu) = \bigwedge_{(q', c, q'') \in \mu} \bigwedge_{(q'', 1, q''') \in \zeta} \bigwedge_{q \in Q} [(A, q, c', q''', \in) \vee (S, q, -1, q', \notin)]$, where $c' = \min(c, col(q'), col(q'''))$.
- We do the same thing for condition 4, that is we set $\varphi_1 = (q, 1, q') \in \tau(i)$, $\varphi_2 = (q', c, q'') \in \eta(i+1)$, $\varphi_3 = (q'', -1, q''') \in \tau(i+1)$ and $\varphi_4 = (q, \min(c, col(q'), col(q''')), q''') \in \eta(i)$. The condition is expressed by the formula $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \Rightarrow \varphi_4$, which is equivalent to $\varphi_1 \wedge \neg\varphi_4 \Rightarrow \neg\varphi_2 \vee \neg\varphi_3$. Thus we have the expression $\Upsilon^4(a, \zeta, \mu) = \bigwedge_{(q, 1, q') \in \zeta} \bigwedge_{(q, c', q''') \in \mu} \bigwedge_{q'' \in Q} [(A, q', c, q'', \notin) \vee (S, q'', -1, q''', \notin)]$, where $c' = \min(c, col(q'), col(q'''))$.
- We do the same thing for condition 5, that is we set $\Upsilon^5(a, \zeta, \mu) = \bigwedge_{(q, 1, q'') \in \zeta} \bigwedge_{q \in Q} [(A, q, c, q'', \in) \vee (S, q, -1, q', \notin)]$, where $c = \min(col(q'), col(q''))$.
- We do the same thing for condition 6, that is we set $\Upsilon^5(a, \zeta, \mu) = \bigwedge_{(q, 1, q') \in \zeta} \bigwedge_{(q, c, q'') \notin \mu} (S, q, -1, q'', \notin)$, where $c = \min(col(q'), col(q''))$.
- For the initialization, $\delta_1(q^1_{\text{in}}, (a, \zeta, \mu))$ equals $q^1_1 \wedge \Upsilon^c(a, \zeta) \wedge \Upsilon^3(a, \zeta, \mu) \wedge \Upsilon^4(a, \zeta, \mu) \wedge \Upsilon^5(a, \zeta, \mu) \wedge \Upsilon^6(a, \zeta, \mu)$ if $q_{\text{in}} \in state(\zeta)$, and equals $false$ otherwise.
- Finally, for the propagation, we set $\delta_1(q^1_1, (a, \zeta, \mu)) = q^1_1 \wedge \Upsilon^c(a, \zeta) \wedge \Upsilon^3(a, \zeta, \mu) \wedge \Upsilon^4(a, \zeta, \mu) \wedge \Upsilon^5(a, \zeta, \mu) \wedge \Upsilon^6(a, \zeta, \mu)$.

It is easily verified that a word u is accepted by $\mathcal{B}_1$ if and only if $p_3(u)$ is an annotation of the strategy $p_2(u)$. It is also clear that $|Q_1| = \mathcal{O}(nd)$.

Description of $\mathcal{B}_2$: First recall that $\mathcal{B}_2$ is used to check that all downward path are accepting. It will therefore be equipped with a parity condition and will be purely universal.

More precisely, $\mathcal{B}_2 = \langle Q \times \{1, \ldots, d\}, A', (q_{\text{in}}, col(q_{\text{in}})), \delta_2, col_2 \rangle$, where the following holds.

- For every state (q, c) and every letter (a, ζ, μ), $\delta_2((q, c), (a, \zeta, \mu))$ equals $false$ if there exists some odd color c' such that $(q, c', q) \in \mu$, or if there exists some odd color c', some color c'' and some state $q' \in Q$, such that $(q, c'', q') \in \mu$ and $(q', c', q') \in \mu$. Otherwise,
  $\delta_2((q, c), (a, \zeta, \mu)) = \bigwedge_{(q,1,q') \in \zeta} (q', col(q')) \wedge \bigwedge_{(q,c',q') \in \mu} \bigwedge_{(q',1,q'') \in \zeta} (q'', \min(c', col(q'')))$
- For every state (q, c), $col_2((q, c)) = c$.

It is then easily verified that $\mathcal{B}_2$ accepts a word u if and only if all downward path in $p_3(u)$ are accepting. It is also clear that $|Q_1| = \mathcal{O}(nd)$ and that $\mathcal{B}_2$ is purely universal. □

A.3 Proof of Theorem 2

Theorem 2. *Deciding the winner in a one-counter parity game can be done in* PSPACE.

Proof. It suffices to note that checking whether $\mathcal{A}$ accepts $\bot 1^\omega$ can be reduced to check emptiness for an alternating two-way parity word automaton $\mathcal{A}'$ which is built from $\mathcal{A}'$ by adding one state.

More precisely, $\mathcal{A}' = \langle A \cup \{q_c\}, \{1, \bot\}, q_{\text{in}}, \delta', col' \rangle$, where $col'(q) = col(q)$ if $q \in Q$ and $col'(q_c) = 0$. The transition function δ' is such that $\delta'(q, a) = \delta(q, a)$ for every $q \in Q \setminus \{q_{\text{in}}\}$ and for every $a \in \{1, \bot\}$, $\delta'(q_{\text{in}}, \bot) = \delta(q_{\text{in}}, \bot) \wedge (q_c, 1)$ and $\delta'(q_{\text{in}}, 1) = \delta(q_{\text{in}}, 1)$, and $\delta(q_c, 1) = (q_c, 1)$ and $\delta(q_c, \bot) = false$. Therefore $\mathcal{A}'$ works like $\mathcal{A}$ except that, at the beginning of the run, it starts a computation verifying that the input word is $\bot 1^\omega$. If the input word is $\bot 1^\omega$, $\mathcal{A}'$ accepts if and only if $\mathcal{A}$ does. Hence $\bot 1^\omega$ is accepted by $\mathcal{A}$ if and only if the language accepted by $\mathcal{A}'$ is not empty (equivalently equal to $\{\bot 1^\omega\}$). □

A.4 Proof of Theorem 3

Theorem 3. *Deciding the winner in a one-counter reachability game is a* DP*-hard problem.*

Proof. We first show how to polynomially reduce 3-SAT to the game problem. Let $X = \{x_1, \ldots, x_k\}$ be a set of variables and let ψ be some Boolean formula in conjunctive normal form with 3 literals per clause. Let denote $\psi = C_1 \wedge C_2 \wedge \cdots \wedge C_h$, where $C_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ for all $i = 1, \ldots, h$ with $l_{i,k} \in \{x, \overline{x} \mid x \in X\}$, for $k = 1, 2, 3$.

For every $i \geq 1$, let ρ_i denote the i-th prime number. A valuation of X is a mapping from X into $\{0, 1\}$, that is a vector in $\{0, 1\}^k$. Consider the function

$\tau : \mathbb{N} \to \{0,1\}^k$ defined by $\tau(n) = (b_1, b_2, \ldots, b_k)$ where $b_j = 0$ if $n = 0 \mod \rho_j$ and $b_j = 1$ otherwise. The Chinese remainder lemma implies that τ is surjective.

Let us consider the following game:

1. Eve chooses some integer n encoding a valuation that she claims to satisfy ψ.
2. Adam picks a clause C_i that he claims not to be satisfied by the preceding valuation.
3. Eve gives a literal of C_i that she claims to be evaluated to true by the preceding valuation.
4. Adam checks whether this literal is evaluated to true. If it is the case, then Eve wins, otherwise Adam does.

It is easily seen that Eve has a winning strategy if and only if ψ is satisfiable.

Now it remains to explain how to encode the preceding games into a one-counter reachability graph. The initial configuration $(q_{\text{in}}, \bot)$ is controlled by Eve. From it she can apply the transitions $push(q_{\text{in}}, 1)$ and $skip(q_{cc})$. In this last case (that concludes the first step of the informal game), the play goes in some configuration $(q_{cc}, 1^n\bot)$ encoding the valuation $\tau(n)$, and it is Adam's turn to play. Adam can apply any rule $skip(q_{C_i})$ for $i = 1, \ldots, h$, meaning that he wants to check that the clause C_i is satisfied by $\tau(n)$ (second step of the previous game). From $(q_{C_i}, 1^n\bot)$, Eve plays a transition $skip(q_{l_{i,j}})$ for any $j \in \{1,2,3\}$, claiming that the literal $l_{i,j}$ is evaluated to true by $\tau(n)$ (third step). Then she applies the transition $skip(m_0^{\rho_k})$ if $l_{i,j} = x_k$, or $skip(\overline{m}_0^{\rho_k})$ if $l_{i,j} = \overline{x_k}$. From the configuration $(m_0^{\rho_k}, 1^n\bot)$ Adam pops until the stack gets empty, and in the same time a modulo ρ_k counting is performed: the successive states are $m_1^{\rho_k}, m_2^{\rho_k}, \ldots, m_{\rho_k-1}^{\rho_k}, m_0^{\rho_k}, m_1^{\rho_k} \ldots$ Finally a configuration $(m_l^{\rho_k}, \bot)$ is reached, and there is a unique possible transition rule: $skip(q_w)$ if $l = 0$, or $skip(q_l)$ otherwise. The states q_w and q_l are looping states and q_w is the only final state. In the case where $l_{i,j} = \overline{x_k}$, the play goes the same except that it goes to $(q_w, \bot)$ from $(\overline{m}_l^{\rho_k}, \bot)$ if $l \neq 0$, and to $(q_l, \bot)$ otherwise.

It is clear that $(p_{\text{in}}, \bot)$ is winning for Eve if and only if ψ is satisfiable. It is not difficult to check that one can build in polynomial time the underlying one-counter process (that has $\mathcal{O}(\Sigma_{i=1}^k \rho_i)$ states which is polynomial in k).

Now, if one wants to reduce SAT-UNSAT, it suffices to add a preliminary step to the previous game. Let (ψ_1, ψ_2) be the instance of SAT-UNSAT. First Adam picks ψ_1 or ψ_2. In the first case Eve and Adam play the previous game. In the second case, they play the symmetrical game where Adam is now the one that has to provide a valuation for ψ_2, and where Eve wins if and only if ψ_2 is not satisfiable. Eve wins the main game if and only if she can win both sub-games, that is if and only if ψ_1 is satisfiable while ψ_2 is not. □

B Details for Section 5

B.1 The one-counter game graph $\widetilde{\mathcal{G}}$

Let us first precisely describe the one-counter game graph $\widetilde{\mathcal{G}}$. We refer the reader to Figure 1. For simplicity, we denote a configuration of this one-counter automa-

ton by (p, κ) instead of $(p, 1^\kappa \bot)$, and when decrementing the counter, we have the convention that $\kappa - 1$ designates 0 if $\kappa = 0$.

- The main vertices of $\widetilde{\mathcal{G}}$ are those of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$, where $p \in Q$, $\alpha \in \Gamma$, $\overrightarrow{R} = ((R_0^-, R_0^+) \dots, (R_d^-, R_d^+)) \in \mathcal{P}(Q)^{2d+2}$, $\theta \in \{0, \dots, d\}$ and $\kappa \geq 0$. A vertex $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ is reached when simulating a partial play Λ in $\mathbb{G}$ such that:
  - The last vertex in Λ is $(p, \alpha\sigma)$ for some $\sigma \in \Gamma^*$.
  - Eve claims that she has a strategy to continue Λ in such a way that if α is eventually popped, the control state reached after popping α belongs to $R^\star_m$, where m is the smallest color visited since α was on the stack, and $\star = +$ if the maximal increase of the stack height was at least κ since α was on the stack, and $\star = -$ otherwise.
  - The color θ is the smallest one since α was pushed onto the stack.

  A vertex $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ is controlled by Eve if and only if $p \in Q_{\mathbf{E}}$.
- The control states tt and ff are there to ensure that the vectors $\overrightarrow{R}$ encoded in the main vertices are correct. Vertices of the form $[tt, \kappa]$ are controlled by Adam, whereas vertices of the form $[ff, \kappa]$ belongs to Eve. As these vertices are dead-ends, a play reaching some vertex $[tt, \kappa]$ is won by Eve whereas a play reaching some vertex $[ff, \kappa]$ is won by Adam.

  There is a transition from some vertex $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ to $[tt, \kappa]$, if and only if there exists a transition rule $pop(r) \in \Delta(p, \alpha)$, such that $r \in R^\star_\theta$ with $\star = -$ if $\kappa > 0$ and $\star = +$ otherwise (this means that $\overrightarrow{R}$ is correct with respect to this transition rule). Symmetrically, there is a transition from a vertex $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ to a vertex $[ff, \kappa]$ if and only if there exists a transition rule $pop(r) \in \Delta(p, \alpha)$ such that $r \notin R^\star_\theta$ where $\star = -$ if $\kappa > 0$ and $\star = +$ otherwise (this means that $\overrightarrow{R}$ is not correct with respect to this transition rule).
- To simulate a transition rule $skip(q) \in \Delta(p, \alpha)$, the player that controls $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ goes in $[(q, \alpha, \overrightarrow{R}, \min(\theta, \rho(q))), \kappa]$. Note that the last component of the control state has to be updated as the smallest color seen since α is on the stack is now $\min(\theta, \rho(q))$.
- To simulate a transition rule $push(q, \beta) \in \Delta(p, \alpha)$, the player that controls $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ goes in $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa]$. This vertex is controlled by Eve who has to give a vector $\overrightarrow{S} = ((S_0^-, S_0^+), \dots, (S_d^-, S_d^+)) \in \mathcal{P}(Q)^{2d+2}$ that describes the control states that can be reached if β is eventually popped. To describe this vector, she goes to the corresponding vertex $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa]$. The vertex $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa]$ is controlled by Adam who chooses either to simulate a bump or a stair. In the first case, he additionally chooses whether the simulated bump will have a stack height (strictly) smaller than κ or not. He also has to pick the minimal color in this bump. To simulate a bump of height smaller than κ with minimal color i, he goes to a vertex $[(s, \alpha, \overrightarrow{R}, \min(\theta, i, col(s))), \kappa]$, for some $s \in S_i^-$, through an edge colored by i and b (b stands for small bump).

To simulate a bump of height greater than κ with minimal color i, he goes to a vertex $[(s, \alpha, \overrightarrow{R^+}, \min(\theta, i, col(s))), \kappa+1]$, for some $s \in S_i^+$, through an edge colored by i and B (B stands for big bump). Here $\overrightarrow{R^+} = ((R_0^+, R_0^+), \ldots, (R_d^+, R_d^+))$. Note that the counter is incremented in that case.
Finally to simulate a stair, Adam goes to the vertex $[(q, \beta, \overrightarrow{S}, \rho(q)), \kappa - 1]$, through an edge colored by S (for stair). Note that the counter is decremented in that case.
The last component of the control state (that stores the smallest color seen since the currently simulated stack level was reached) has to be updated in all these cases. After simulating a bump of minimal color i, the minimal color is $\min(\theta, i, col(s))$. After simulating a stair, this color has to be initialized (since a new stack level is simulated). Its value, is therefore $\rho(q)$, which is the unique color since the (new) stack level was reached.
Note that in the simulation of a bump of height greater than κ, the vector $\overrightarrow{R}$ of possible reachable states when popping α is updated: whatever happens now, if α is popped, the increased of the stack on the current level would have been greater than κ and hence the control state reached would have to be in R_m^+ for some m.

The only vertices that are colored are those of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ and the color of such a vertex is $col(p)$. Some edges are also colored. These are the one leaving from some vertex $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa]$: this (possibly multi) coloring is always formed by a color in $\{S, b, B\}$ that can be completed (for bumps) by a color in $\{0, 1, \ldots, d\}$. See Figure 1 for details.

Remark B1 *In the definition of parity games we were requiring to have a total coloring function working only on vertices. One can add extra intermediate states and introduce a new color larger than d to fit the definition without changing the issue of that game.*

B.2 Proof of Theorem 5

This subsection is devoted to the proof Theorem 5 that is recalled bellow.

Theorem 5. *A configuration $(p_{in}, \bot)$ is winning for Eve in $\mathbb{G} = (\mathcal{G}, \psi(\Omega_1, \ldots, \Omega_k))$ if and only if $[(p_{in}, \bot, ((\emptyset, \emptyset), \ldots, (\emptyset, \emptyset), col(p_{in})), 0]$ is winning for Eve in $\widetilde{\mathbb{G}} = (\widetilde{\mathcal{G}}, \psi(\widetilde{\Omega_1}, \ldots, \widetilde{\Omega_k}))$. Hence, deciding the winner in such a pushdown game can be done in* ExpSpace.

Factorization of a play in $\widetilde{\mathbb{G}}$

Recall that in $\widetilde{\mathcal{G}}$ some edges are colored and that we can also have two colors on the same edge. Hence, to represent a play, we have to encode this information

on edge coloring. First, note that the information on colors in $\{S, b, B\}$ is implicitly encoded into the vertices, as they respectively correspond to decrement/do not modify/increment the counter. Therefore we only need to encode the colors in $\{0, \dots, d\}$ that appears when simulating a bump: a play will be represented as a sequence of vertices together with colors in $\{0, \dots, d\}$ that correspond to colors appearing on edges.

For any play in $\widetilde{\mathbb{G}}$, a *round* is a factor between two visits through vertices of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$. We have the following possible forms for a round:

- The round is of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa][(q, \alpha, \overrightarrow{R}, \theta), \kappa]$ and corresponds therefore to the simulation of a skip rule. We designate it as a *bump of height 0.*
- The round is of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa][(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa][(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa] i$ $[(s, \alpha, \overrightarrow{R}, \min(\theta, i, col(s))), \kappa]$ and corresponds therefore to the simulation of a rule pushing β followed by a sequence of moves that ends by popping β. Moreover the stack height does not increase by more than κ in the meantime. We designate it has a *small bump.*
- The round is of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa][(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa][(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa] i$ $[(s, \alpha, \overrightarrow{R^+}, \min(\theta, i, col(s))), \kappa + 1]$ and corresponds therefore to the simulation of a rule pushing β followed by a sequence of moves that ends by popping β. Moreover the stack height increases by more than κ in the meantime. We designate it has a *big bump.*
- The round is of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa][(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa][(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa]$ $[(q, \beta, \overrightarrow{S}, col(q)), \kappa - 1]$ and corresponds therefore to the simulation of a rule pushing a symbol β that will not be removed. We designate it has a *stair.*

For any play $\lambda = v_0 v_1 v_2 \cdots$ in $\widetilde{\mathbb{G}}$, we consider the subset of indices corresponding to vertices of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$. More precisely:

$$Rounds_\lambda = \{n \in \mathbb{N} \mid v_n = [(p, \alpha, \overrightarrow{R}, \theta), \kappa],\ p \in Q,\ \alpha \in \Gamma,\ \overrightarrow{R} \in \mathcal{P}(Q)^{2d+2},\ 0 \leq \theta \leq d,\ \kappa \geq 0\}$$

Therefore, the set $Rounds_\lambda$ induces a natural factorization of λ into rounds.

Definition B2 (Rounds factorisation) *For a (possibly partial) play $\lambda = v_0 v_1 v_2 \cdots$, we call* rounds factorization *of λ, the (possibly finite) sequence $(\lambda_i)_{i \geq 0}$ of rounds λ defined as follows. Let $Rounds_\lambda = \{n_0 < n_1 < n_2 < \cdots\}$, then for all $0 \leq i < |Rounds_\lambda|$, $\lambda_i = v_{n_i} \cdots v_{n_{i+1}}$.*

Therefore, for every $i \geq 0$, the first vertex in λ_{i+1} equals the last one in λ_i. Moreover, $\lambda = \lambda_1 \odot \lambda_2 \odot \lambda_3 \odot \cdots$, where $\lambda_i \odot \lambda_{i+1}$ denotes the concatenation of λ_i with λ_{i+1} without its first vertex.

Finally, the color *of a round is the smallest color in $\{0, \dots, d\}$ appearing in the round.*

In order to prove both implications of Theorem 5, we build from a winning strategy for Eve in one game a winning strategy for her in the other game. The main argument to prove that the new strategy is winning is to prove a correspondence between the factorizations of plays in both games.

Direct implication

Assume that the configuration $(p_{in}, \bot)$ is winning for Eve in $\mathbb{G}$, and let Φ be a corresponding strategy for her.

Using Φ, we define a strategy φ for Eve in $\widetilde{\mathbb{G}}$ from $[(p_{in}, \bot, (\emptyset, \dots, \emptyset), \rho(p_{in})), 0]$. This strategy stores a partial play in $\mathbb{G}$, that is an element in V^* (where V denotes the set of vertices of $\mathcal{G}$). This memory will be denoted by Λ. At the beginning Λ is initialized to the vertex $(p_{in}, \bot)$. We first describe φ, and then we explain how Λ is updated. Both the strategy φ and the update of Λ, are described for a round.

Choice of the move. Assume that the play is in some vertex $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ for $p \in Q_{\mathbf{E}}$. The move given by φ depends on $\Phi(\Lambda)$:

- If $\Phi(\Lambda) = pop(r)$, then Eve goes to $[\sharp, \kappa]$ (Proposition B3 will prove that this move is always possible).
- If $\Phi(\Lambda) = skip(q)$, then Eve goes to $[(q, \alpha, \overrightarrow{R}, \min(\theta, col(q))), \kappa]$.
- If $\Phi(\Lambda) = push(q, \beta)$, then Eve goes to $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa]$.

In this last case, or in the case where $p \in Q_{\mathbf{A}}$ and Adam goes to $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa]$, we also have to explain how Eve behaves from $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa]$. She has to provide a vector $\overrightarrow{S} \in \mathcal{P}(Q)^{2d+2}$ that describes which states can be reached if β is popped, depending on both the increased of the stack compared with κ and on the smallest visited color in the meantime. In order to define $\overrightarrow{S}$, Eve considers the set of all possible continuation of $\Lambda \cdot (q, \beta\alpha\sigma)$ (where $(p, \alpha\sigma)$ denotes the last vertex of Λ) where she respects her strategy Φ. For each such play, she checks whether some configuration of the form $(s, \alpha\sigma)$ is visited after $\Lambda \cdot (q, \beta\alpha\sigma)$, that is if β is eventually popped. If it is the case, she considers the first configuration $(s, \alpha\sigma)$ appearing after $\Lambda \cdot (q, \beta\alpha\sigma)$, the smallest color i and the maximal increase h of the stack height since β was on the stack. For every $i \in \{0, \dots d\}$, S_i^-, is exactly the set of states $s \in Q$ such that the preceding case happens with $h < \kappa - 1$, and S_i^+ is exactly the set of states $s \in Q$ such that the preceding case happens with $h \geq \kappa - 1$. Finally, we set $\overrightarrow{S} = ((S_0^-, S_0^+), \dots, (S_d^-, S_d^+))$ and Eve moves to $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa]$.

Update of Λ. The memory Λ is updated after each visit to a vertex of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$. We have three cases depending on the kind of the round:

- The round is a bump of height 0 and therefore a $skip(q)$ action was simulated. Let $(p, \alpha\sigma)$ be the last vertex in Λ, then the updated memory is $\Lambda \cdot (q, \alpha\sigma)$.
- The round is a bump of non null height, and therefore a bump of color i (where i is the color of the round) starting with some action $push(q, \beta)$ and ending in a state $s \in S_i^- \cup S_i^+$ was simulated. Let $(p, \alpha\sigma)$ be the last vertex in Λ. Then the memory becomes Λ extended by $(q, \beta\alpha\sigma)$ followed by a sequence of moves, where Eve respects Φ, that ends by popping β and reach $(s, \alpha\sigma)$ while having i as smallest color and augmenting the stack height by at least κ, if $s \in S_i^+$ (respectively by at most $\kappa - 1$ if $s \in S_i^-$).

– The round is a stair and therefore we have simulated a $push(q, \beta)$ action. If $(p, \alpha\sigma)$ denotes the last vertex in Λ, then the updated memory is $\Lambda \cdot (q, \beta\alpha\sigma)$.

Therefore, with any partial play λ in $\widetilde{\mathbb{G}}$ in which Eve respects her strategy φ, is associated a partial play Λ in $\mathbb{G}$. An immediate induction shows that Eve respects Φ in Λ. The same arguments works for an infinite play λ, and the corresponding play Λ is therefore infinite, starts from $(p_{in}, \bot)$ and Eve respects Φ in that play. Therefore it is a winning play.

The following proposition is a consequence of how φ was defined.

Proposition B3 *Let λ be a partial play in $\widetilde{\mathbb{G}}$ that starts from $[(p_{in}, \bot, (\emptyset, \dots, \emptyset), \rho(p_{in})), 0]$, ends in a vertex of the form $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$, and where Eve respects φ. Let Λ be the play associated to λ built by the strategy φ. Then the following holds:*

1. *Λ ends in a vertex of the form $(p, \alpha\sigma)$ for some $\sigma \in \Gamma^*$.*
2. *θ is the smallest visited color in Λ since α has been pushed.*
3. *In Λ a configuration of stack height $\kappa + |\sigma|$ has been visited.*
4. *Assumer that Λ is extended, that Eve keeps respecting Φ and that the next move after $(p, \alpha\sigma)$ is to some vertex (r, σ). Then $r \in R_i^\star$, where i is the smallest visited color since α was on the stack (that is $i = \theta$ from point (2)) and $\star = +$ if $\kappa = 0$ and $\star = -$ otherwise.*

Proof. The only difficult part is (4). The fact that $r \in R_i^- \cup R_i^+$ is clear. First note that λ ends by a (possibly empty) sequence of bumps of the form $[(p', \alpha, \overrightarrow{R'}, \theta'), \kappa'] \dots [(p'', \alpha, \overrightarrow{R'^\star}, \theta''), \kappa' + \iota]$. Let $[(p', \alpha, \overrightarrow{R'}, \theta'), \kappa']$ be the first vertex in this sequence In the special case where the sequence is empty, $p' = p$, $\overrightarrow{R'} = \overrightarrow{R}$, $\theta' = \theta$ and $\kappa' = \kappa$. Moreover, one has $\overrightarrow{R} = \overrightarrow{R'}$ and $\kappa' = \kappa$ if all bumps have height smaller than κ', and otherwise $\overrightarrow{R} = \overrightarrow{R'^+}$ and $\kappa' < \kappa$.

In the special case where $\kappa = 0$, $\kappa' = 0$ and by definition of $R_i'^+$, one has $r \in R_i'^+ = R_i^+$. In the case where $\kappa > 0$, if $\kappa' = \kappa$, all bumps before popping α had height less than κ', and by definition of $R_i'^-$, one has $r \in R_i'^- = R_i^-$. If $\kappa' < \kappa$, then a bump higher than κ' was done before popping α, and therefore, by definition of $R_i'^+$, one has $r \in R_i'^+ = R_i^-$.

Remark B4 *Proposition B3 implies that the strategy φ is well defined when it provides a move to $\mathtt{tt}$. Moreover, one can deduce that, if Eve respects φ, no configuration of control state $\mathtt{ff}$ is visited.*

The preceding remark shows in particular that any finite play ends in some vertex $[\mathtt{tt}, \kappa]$ and is therefore won by Eve. For infinite play, using the definitions of $\widetilde{\mathcal{G}}$ and φ, we easily deduce the following proposition.

Proposition B5 *Let λ be an infinite play in $\widetilde{\mathbb{G}}$ that starts from $[(p_{in}, \bot, (\emptyset, \dots, \emptyset), \rho(p_{in})), 0]$, and where Eve respects φ. Let Λ be the associated play build by the strategy φ. Let $(\lambda_i)_{i \geq 0}$ be the round factorization of λ. Then, for every $i \geq 1$ the following hold:*

1. *λ_i is a bump if and only if $kind_i^\Lambda = (B, h)$ for some h. Moreover if κ is the counter value in the first vertex of λ_i, $h \geq \kappa$ if and only if λ_i is a big bump.*
2. *λ_i has color $mcol_i^\Lambda$.*

Proposition B5 implies that for any infinite play λ in $\widetilde{\mathbb{G}}$ starting from $[(p_{in}, \bot, (\emptyset, \dots, \emptyset), \rho(p_{in})), 0]$ where Eve respects φ, the sequence of visited colors in λ is $(mcol_i^\Lambda)_{i\geq 0}$ for the corresponding play Λ in $\mathbb{G}$. Moreover $\lambda \in \Omega_{Buc}(\{S, B\})$ if and only if $(kind_i^\Lambda)_{i\geq 0}$ contains either infinitely many S or (B, h) for every $h \geq 0$ (because if there are finitely many S in λ but infinitely many B, there are higher and higher bumps in some stack level). Finally $\lambda \in \Omega_{Buc}(\{S\})$ if and only if $(kind_i^\Lambda)_{i\geq 0}$ contains infinitely many S.

Using Proposition 3 we conclude that $\lambda \in \psi(\widetilde{\omega_1}, \dots, \widetilde{\Omega_k})$ if and only if $\Lambda \in \psi(\Omega_1, \dots, \Omega_k)$. As Λ is winning for Eve, it follows that λ is also winning for her.

Converse implication

Assume now that Eve has a winning strategy φ in $\widetilde{\mathbb{G}}$ from $[(p_{in}, \bot, (\emptyset, \dots, \emptyset), col(p_{\text{in}})), 0]$. Using φ, we build a strategy Φ for Eve in $\mathbb{G}$ for plays starting from $(p_{in}, \bot)$.

The strategy Φ uses, as memory, a stack Π, to store the complete description of a play in $\widetilde{\mathbb{G}}$. Recall here that a play in $\widetilde{\mathbb{G}}$ is represented as a sequence of vertices together with colors in $\{0, \dots d\}$.

Therefore the stack alphabet of Π is the set of vertices of $\widetilde{\mathcal{G}}$ together with $\{0, \dots, d\}$. Note that this alphabet is infinite. In the following, $top(\Pi)$ will denote the top stack symbol of Π while $StCont(\Pi)$ will be the word obtained by reading Π from bottom to top (without considering the bottom-of-stack symbol of Π). In any play where Eve respects Φ, $StCont(\Pi)$ will be a play in $\widetilde{\mathbb{G}}$ that starts from $[(p_{in}, \bot, (\emptyset, \dots, \emptyset), col(p_{\text{in}})), 0]$ and where Eve respects her winning strategy φ. Moreover, for any play Λ where Eve respects Φ, we will always have that $top(\Pi) = [(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ if and only if the current configuration in Λ is of the form $(p, \alpha\sigma)$. Finally, if Eve keeps respecting Φ, and if α is eventually popped the configuration that is reached will be of the form (r, σ) for some $r \in R_i^\star$, where i is the smallest visited color since α was on the stack, and $\star = +$ if the stack was increased by at least κ symbols in between $(p, \alpha\sigma)$ and (r, σ), and $\star = -$ otherwise. Initially, Π only contains $[(p_{\text{in}}, \bot, (\emptyset, \dots, \emptyset), col(p_{\text{in}})), 0]$.

In order to describe Φ, we assume that we are in some configuration $(p, \alpha\sigma)$ and that $top(\Pi) = [(p, \alpha, \overrightarrow{R}, \theta), \kappa]$. We first describe how Eve plays if $p \in Q_{\mathbf{E}}$, and then we explain how the stack is updated.

- **Choice of the move.** Assume that $p \in Q_{\mathbf{E}}$ and that Eve has to play from some vertex $(p, \alpha\sigma)$. For this, she considers the value of φ on $StCont(\Pi)$.
  If it is a move to $[\sharp t, \kappa]$, Eve plays an action $pop(r)$ for some state $r \in R_\theta^\star$, with $\star = -$ if $\kappa > 0$ and $\star = +$ otherwise. Lemma B6 will prove that such an r always exists.
  If the move given by φ is to go to some vertex $[(q, \alpha, \overrightarrow{R}, \min(\theta, col(q))), \kappa]$, Eve applies the transition $skip(q)$.

If the move given by φ is to go to some vertex $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa]$, then Eve applies the transition $push(q, \beta)$.

– **Update of Π.** Assume that the last move, played by Eve or Adam, was to go from $(p, \alpha\sigma)$ to some configuration (r, σ). The update of Π is illustrated by figure 2 and explained in what follows. Eve pops in Π until she finds some configuration of the form $[(p', \alpha', \overrightarrow{R'}, \theta', p'', \alpha, \overrightarrow{R''}), \kappa']$ that is not preceded by a color in $\{0, \ldots, d\}$. This configuration is therefore in the stair that simulates the pushing of α onto the stack. Let h be the maximal increase of the stack when α was in. Eve updates Π by pushing θ in Π followed by $[(r, \alpha', \overrightarrow{R'}, \min(\theta', \theta, col(r))), \kappa']$ if $h < \kappa' - 1$, and by $[(r, \alpha', \overrightarrow{R'^{+}}, \min(\theta', \theta, col(r))), \kappa' + 1]$ otherwise.
Assume that the last move, played by Eve or Adam, was to go from $(p, \alpha\sigma)$ to some configuration $(q, \alpha\sigma)$. Then Eve update Π by pushing $[(q, \alpha, \overrightarrow{R}, \min(\theta, col(q))), \kappa]$.
Assume that the last move, played by Eve or Adam, was to go from $(p, \alpha\sigma)$ to some configuration (q, bau), let $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa] = \varphi(StCont(\Pi) \cdot [(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa])$. Intuitively, $\overrightarrow{S}$ describes which states Eve can ensured to reach if β is eventually popped. Eve updates Π by successively pushing $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta), \kappa]$, $[(p, \alpha, \overrightarrow{R}, \theta, q, \beta, \overrightarrow{S}), \kappa]$, and $[(q, \beta, \overrightarrow{S}, col(q)), \kappa - 1]$.

The following lemma gives the meaning of the information stored in Π.

Lemma B6 *Let Λ be a partial play in $\mathbb{G}$, where Eve respects Φ, that starts from $(p_{in}, \bot)$ and that ends in a configuration $(p, \alpha\sigma)$. We have the following facts:*

1. *$top(\Pi) = [(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ with $\overrightarrow{R} \in \mathcal{P}(Q)^{2d+2}$, $0 \leq \theta \leq d$ and $\kappa \geq 0$*
2. *$StCont(\Pi)$ is a partial play in $\widetilde{\mathbb{G}}$ that starts from $[(p_{in}, \bot, (\emptyset, \ldots, \emptyset), col(p_{in})), 0]$, that ends with $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ and where Eve respects φ.*
3. *θ is the smallest color visited since α was pushed.*
4. *Let $\kappa' - 1$ be the last component of $top(\Pi)$ after having pushed α. If Λ is extended by some move that pops α, the configuration (r, σ) that is reached is such that $r \in R^{\star}_{\theta}$, where $\star = +$ if the maximal increase of the stack height was smaller than $\kappa' - 1$ since α was in, and $\star = -$ otherwise.*

Proof. The proof goes by induction on Λ. We first show that the last point is a consequence of the second and third points. For a better readability, one can refer to Figure 2. Assume that the next move after $(p, \alpha\sigma)$ is to apply an action $pop(r) \in \Delta(p, \alpha)$. We start with the case where $\kappa = 0$. The second point implies that $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ in winning for Eve in $\widetilde{\mathbb{G}}$. If $p \in Q_{\mathbf{E}}$, by definition of Φ, there is some edge from that vertex to $[\sharp, \kappa]$, which means that $r \in R^{+}_{\theta}$ and allows us to conclude as we always have $\kappa \geq \kappa'$. If $p \in Q_{\mathbf{A}}$, note that there is no edge from $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ (winning position for Eve) to the loosing vertex $[ff, \kappa]$. Hence we conclude the same way.

Assume now that $\kappa > 0$. Let h be the maximal increase of the stack height since α was in. Let $[(p'', \alpha, \overrightarrow{R''}, col(p'')), \kappa' - 1]$ be equal to $top(\Pi)$ just after α

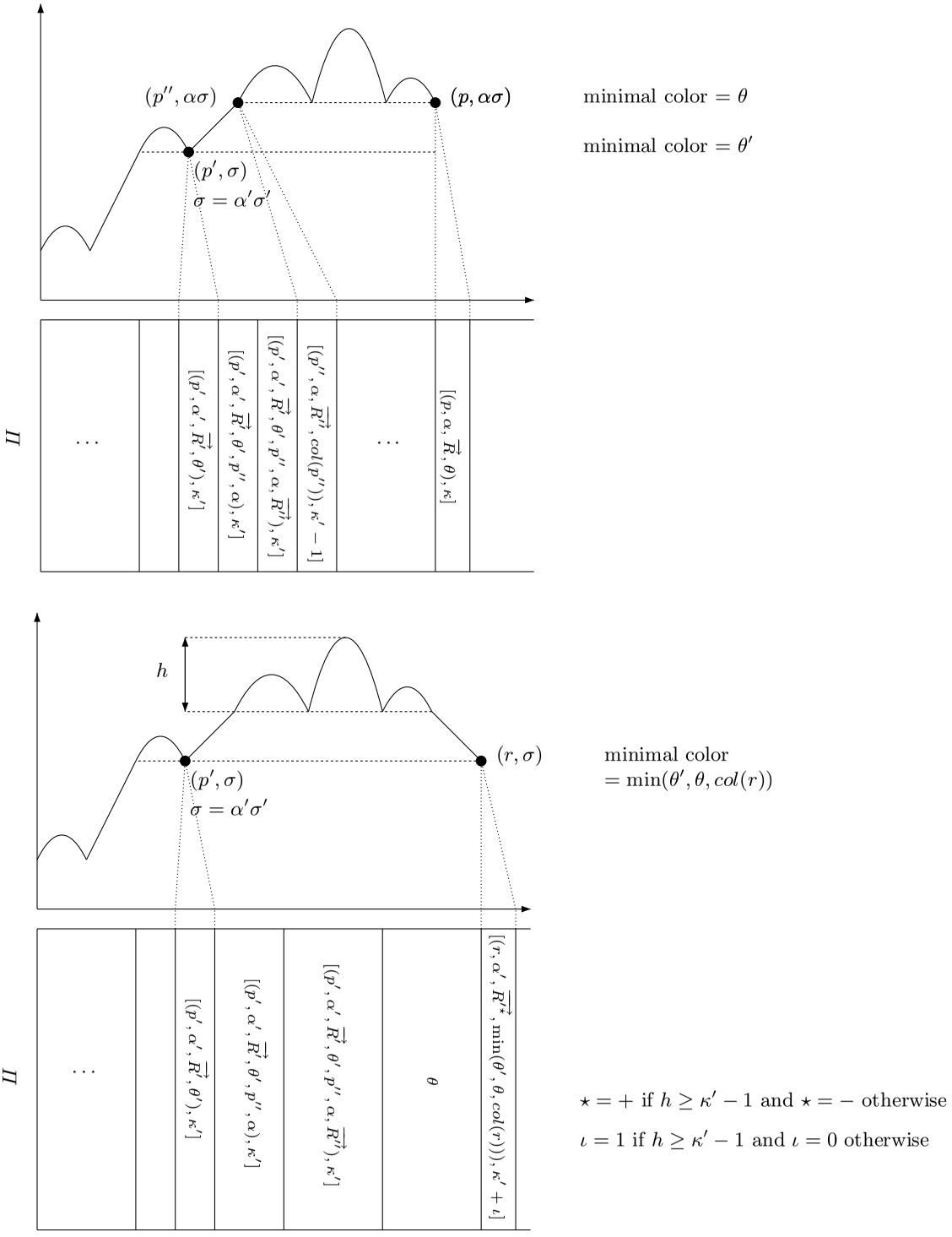

Fig. 2. Updating the strategy's stack Π

was pushed. Considering how Π was updated, one concludes that $\overrightarrow{R} = \overrightarrow{R''}$ if $h < \kappa' - 1$ and that $\overrightarrow{R} = \overrightarrow{R''^{+}}$ otherwise.
Therefore, it is sufficient to show that $r \in R_\theta^-$. The second point implies that $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ is winning for Eve in $\widetilde{\mathbb{G}}$. If $p \in Q_{\mathbf{E}}$, by definition of Φ, there is an edge from that configuration to $[\textit{tt}, \kappa]$, which means that $r \in R_\theta^-$ (as $\kappa > 0$) and allows us to conclude. If $p \in Q_{\mathbf{A}}$, note that there is no edge from $[(p, \alpha, \overrightarrow{R}, \theta), \kappa]$ (winning for Eve) to $[\textit{ff}, \kappa]$. Therefore one concludes the same way.

Let us now prove the other points. For this, assume that the result is proved for some play Λ, and let Λ' be an extension of Λ. We have two cases, depending on how Λ' extends Λ:

- Λ' is obtained by applying a rule of type *skip* or *push*. The result is trivial in that case.
- Λ' is obtained by applying a *pop* rule. Let $(p, \alpha\sigma)$ be the last configuration in Λ, and let $\overrightarrow{R}$ be the last vector component in $top(\Pi)$ when being in configuration $(p, \alpha\sigma)$. By induction hypothesis, it follows that $\Lambda' = \Lambda \cdot (r, \sigma)$ with $r \in R_\theta^\star$. Considering how Π is updated, and using the fourth point, we easily deduce that the new strategy stack Π is as desired (one can have a look at Figure 2 for more intuition).

Actually, we easily deduce a more precise result.

Lemma B7 *Let Λ be a partial play in $\mathbb{G}$ starting from $(p_{in}, \bot)$ and where Eve respects Φ. Let $\lambda = StCont(\Pi)$, where Π denotes the strategy's stack in the last vertex of Λ. Let $(\lambda_i)_{i=0,\dots,k}$ be the round factorization of λ. Then the following holds:*

- *λ_i is a bump if and only if $kind_i^\Lambda = (B, h)$ for some h. Moreover if κ is the counter value in the first vertex of λ_i, $h \geq \kappa$ if and only if λ_i is a big bump.*
- *λ_i has color $mcol_i^\Lambda$.*

Both lemmas B6 and B7 are for partial plays. A version for infinite plays would allow to conclude. Let Λ be an infinite play in $\mathbb{G}$. We define an infinite version of λ by considering the limit of the stack contents $(StCont(\Pi_i))_{i \geq 0}$ where Π_i is the strategy's stack after the i-th first moves in Λ. See [19] for similar constructions. It is easily seen that such a limit exists, is infinite and corresponds to a play won by Eve in $\widetilde{\mathbb{G}}$. Moreover the results of Lemma B7 apply.

Let Λ be a play in $\mathbb{G}$ with initial vertex $(p_{\text{in}}, \bot)$, and where Eve respects Φ, and let λ be the associated infinite play in $\widetilde{\mathbb{G}}$. Therefore λ is won by Eve. Using Lemma B7 and Proposition 3, we conclude, as in the direct implication that Λ is winning.

Solving $\widetilde{\mathbb{G}}$ in ExpSpace.

The one-counter game provided by Theorem 5 is not equipped with a parity condition, so we cannot directly apply Theorem 2. Nevertheless, it is easily seen

that one can always modify $\widetilde{\mathcal{G}}$ to obtain an *equivalent* game $\widetilde{\mathcal{G}}'$ (here equivalent means that one wins in the first game if and only if he wins in the second game). Moreover, constructing $\widetilde{\mathcal{G}}'$ can be done in polynomial time.

We only give the construction for some cases. For the others, it is sufficient to consider the game from Adam's point of view. These constructions are an adaptation of the standard constructions for intersection/union of ω-automata equipped with various acceptance conditions.

- if $\psi(\widetilde{\Omega_1}, \ldots, \widetilde{\Omega_k})$ is the conjunction of a Büchi condition (coming from a stack condition) and of a parity condition, we modify the one-counter process by adding a component that remembers the smallest visited color since the last visit to a final state. Whenever a final state (for the Büchi condition) is visited, a vertex with the stored color is visited. Every other vertex is colored by an odd color larger than all other colors in the game. Hence, Eve wins if and only if infinitely many final states are visited while the parity condition holds.
- if $\psi(\widetilde{\Omega_1}, \ldots, \widetilde{\Omega_k})$ is the disjunction of a Büchi condition (coming from a stack condition) and a parity condition, we introduce a new color which is even and minimal. Any final state for the Büchi condition has this color. Hence, Eve wins in the new parity game if and only if infinitely many final states are visited or if the previous parity condition holds.

Applying Theorem 2 to that new games concludes the proof.