Imperfect Recall and Counter Games

Dietmar Berwanger, Łukasz Kaiser,
Simon Leßenich

September 2011

Research report LSV-11-20

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Imperfect Recall and Counter Games*

Dietmar Berwanger[1], Łukasz Kaiser[2], and Simon Leßenich[3]

[1] LSV, CNRS & ENS Cachan, France
[2] LIAFA, CNRS & Université Paris Diderot – Paris 7, France
[3] Mathematische Grundlagen der Informatik, RWTH Aachen, Germany

Abstract. We study a class of ω-regular games with imperfect information and imperfect recall, for which we present an automata-based solution method. Furthermore, we show a reduction from a class of counter parity games to games with this kind of imperfect recall. By combining the two results, we obtain an elementary algorithm for solving counter parity games, which leads to improved complexity bounds for model checking quantitative variants of the μ-calculus.

1 Introduction

Games with ω-regular winning conditions are a fundamental model for program verification and synthesis [13]. In this context, games with imperfect information have been studied extensively [12,9,1,2]. These studies commonly assume *perfect recall*, i.e., that players never forget their past observations and actions. We study ω-regular games with a kind of imperfect information that violates perfect recall. Our basic model, we call it the *second-life* game, is intended as a proof of concept for addressing the issue of imperfect recall in infinite games.

At the outset, there is a game graph $\mathcal{G}$ with perfect information for two players, Player 0 and Player 1. The second-life game arena consists of several copies of this graph. Plays begin in the main instance of $\mathcal{G}$, which we call *first life*, and proceed as usual by moving a token along the edges of the graph. However, when Player 1 is in turn to move, he may switch to a copy of $\mathcal{G}$, a *second life*, without informing Player 0. If a terminal position is reached in the first life, the play simply ends. In contrast, if this happens in a second life, the play returns to the first life, and Player 0 forgets the part of the history spent in the second life. This part of the history is nevertheless relevant for the winning condition.

For a game graph $\mathcal{G}$ with n positions, the structure of the second-life game consists of $n+1$ disjoint copies of $\mathcal{G}$: one second-life copy $(\mathcal{G}, v)$ for any node v, and the first life copy $\mathcal{G}$. Imperfect information is induced by composing moves of Player 1 in $\mathcal{G}$ with unobservable calls to second-life copies as follows: for a move (u, v) of Player 1, we add a Call move $(u, (v, v))$ and Return moves $((w, v), v)$ for each terminal node w of $\mathcal{G}$. We postulate that Player 0 cannot distinguish between the move (u, v) in the main component and the move $(u, (v, v))$ switching to $(\mathcal{G}, v)$. The resulting uncertainty about whether the play continues in the first

* Work partially supported by the ESF Research Networking Programme GAMES.

or second life, after reaching u, is still subsumed under the usual notion of imperfect information with perfect recall. However, in the event that a return move occurs, our model requires Player 0 to forget the segment of history elapsed since the last call occurred. Technically, the play history from u through the second life $(\mathcal{G}, v)$ that returns via a move $((w, v), v)$ leads to the same information set as the play moving from u directly to v in the first life.

We show that this class of games with imperfect recall admits finite-memory strategies, and propose an effective, automata-based procedure for constructing winning strategies in such games. Moreover, we provide an explicit and elementary bound on the size of memory required in second-life games.

In the second part of the paper, we present an algorithmic application of second-life games to model checking. Specifically, we develop a new algorithm for solving counter parity games. These are quantitative parity games with a set of counters that are updated by affine transformation along the moves of a play. When a play ends, these counters are used to determine the payoff, whereas on infinite plays, a parity condition is applied. Counter parity games are a strict generalization of counter-reset games, which were used in [6] to approximate a quantitative logic over a class of hybrid systems. It was shown there that counter-reset games are algorithmically solvable, but the presented procedure was of non-elementary complexity. Our new algorithm for counter parity games has elementary complexity and works for a strictly larger class than counter-resets games.

Beyond generalizing counter-reset games, counter parity games offer a promising framework for model-checking quantitative logics. Counter reachability games play an important role in model-checking weak cost-MSO [3] and weak MSO with the unbounding quantifier [7]. Recently, the model-checking problem of a quantitative μ-calculus for structure rewriting systems has been reduced to solving counter parity games [8].

2 Preliminaries

When referring to a sequence s, we write $s[i]$ for the i-th element of s. We always count from 0, i.e., $s[0]$ is the first element of s. For a set $I \subseteq \mathbb{N}$ of indices, we write $s|^I$ to denote the sub-sequence of s consisting only of the elements with indices in I. We refer to a sequence of fixed finite length as a vector. For a vector s, we write $s^{>0}$ to denote the vector t with $t[i] := 1$ if $s[i] > 0$ and $t[i] := 0$ otherwise. Finally, we write $[n]$ to denote the set $\{0, \ldots, n-1\}$.

2.1 Games and strategies

For a given set A of *actions*, an *arena* is a labeled graph $\mathcal{G} = (V, V_0, V_1, E)$ with a set V of vertices partitioned into subsets V_0 and V_1 belonging to Player 0 and Player 1, respectively, and a set $E \subseteq V \times A \times V$ of moves corresponding to edges labeled with actions from A.

For a finite arena $\mathcal{G} = (V, V_0, V_1, E)$, we will often assume implicitly that the moves in E are linearly ordered. This allows us to order the moves going out from each vertex v as $\{v\} \times A \times V \cap E = \{(v, a_0, w_0), \ldots, (v, a_{k-1}, w_{k-1})\}$. Then, we say that a_i is the i-th action from v and write $a_i = \mathrm{act}_i(v)$; likewise, we say that w_i is the i-th successor from v and write $w_i = \mathrm{succ}_i(v)$.

A sequence $\pi \in (VA)^*V$ is a *path* through $\mathcal{G}$, if $\pi = v_0a_0v_1 \cdots a_nv_n$ such that $(v_i, a_i, v_{i+1}) \in E$ for each $i \in [n]$.

An (unconstrained) *strategy* of Player i, starting from v_0, is a function f that assigns to each path π through $\mathcal{G}$ starting in v_0 and ending at $v \in V_i$, an action a and a successor w such that $(v, a, w) \in E$. We say that a path $v_0a_0v_1 \cdots a_nv_n$ is *consistent* with a strategy f, if it assign to every prefix path $v_0a_0v_1 \cdots a_{l-1}v_l$ with $l < n$ the action a_l and the vertex v_{l+1}. An *winning condition* is a set $W \in A^\omega$ of infinite sequences of actions. A strategy f is *winning* from an initial position $v_0 \in V$, if each play from v_0 that is consistent with f belongs to W.

Given a vertex $v_0 \in V$, we define the *unfolding* of $\mathcal{G}$ from v_0 as the ranked tree $\mathcal{T}(\mathcal{G}, v_0)$ of all finite paths $\pi \in (VA)^*V$ rooted in v_0 with the i-th successor given by

$$S_i = \{(\pi,\ \pi\,\mathrm{act}_i(v_n)\,\mathrm{succ}_i(v_n)) \mid \pi = v_0a_0v_1a_2 \cdots a_nv_n\}.$$

In addition, every vertex $\pi = v_0a_0v_1 \cdots v_n$ in $\mathcal{T}(\mathcal{G}, v_0)$ is labeled by a special predicate v_n.

We represent a strategy f of Player i by a labeling of the unfolding $\mathcal{T}(\mathcal{G}, v_0)$ with an additional predicate S as follows.

(1) For every $\pi = v_0a_0 \cdots v_n$ with $v_n \in V_i$, the vertex $\pi f(\pi)$ is labeled with S and all other successors of π are not labeled with S, and
(2) for every $\pi = v_0 \cdots v_n$ with $v_n \in V_{1-i}$, *all* successors are labeled with S.

As there is a one-to-one correspondence between strategies of Player i and labelings S that satisfy the above two properties, we use both representations interchangeably.

We say that a strategy f of Player 0 uses *memory* M if there exists a $m_0 \in M$, a function update : $M \times V \to M$ and a function $f_M : M \times V \to V$ such that $f(v_0v_1 \ldots v_n) = f_M(\mathrm{update}^*(v_0v_1 \ldots v_n, m_0), v_n)$, where update^* is defined inductively by $\mathrm{update}^*(\varepsilon, m) = m$ and $\mathrm{update}^*(v_0v_1 \ldots v_{k+1}, m) = \mathrm{update}(\mathrm{update}^*(v_0v_1 \ldots v_k, m), v_{k+1})$. The *size* of the memory is $|M|$ and a *finite-memory strategy* is one that uses a finite memory M.

2.2 Alternating tree automata

We assume that the reader is familiar with automata on ω-words and on infinite trees. Here, we only recall the notion of alternating tree automata [11].

To define the transitions of alternating automata over trees with arbitrary branching, we consider, for a given set of states Q and a set of directions D, the set $\mathcal{B}^+(D \times Q)$ of all *positive Boolean formulas* over $D \times Q$. That is, $\mathcal{B}^+(D \times Q)$ is the set of formulas built using propositions from $D \times Q$, the connectives $\wedge$ and

$\vee$, and the constants true and false; note that negation is not allowed. A subset $X \subseteq D \times Q$ *satisfies* a formula $\varphi \in \mathcal{B}^+(D \times Q)$ if φ is satisfied by the assignment that sets all elements of X to true and all elements of $(D \times Q) \setminus X$ to false.

Fix a finite ranked alphabet Σ and denote the arity of a symbol $f \in \Sigma$ by $\mathrm{ar}(f)$. An *alternating automaton* $\mathcal{A}$ over Σ is a tuple $(Q, \delta, q_0, \mathcal{F})$, where Q is the set of states, q_0 is the initial state, $\mathcal{F} \subseteq Q^\omega$ is the acceptance condition, and δ assigns a positive Boolean formula over possible successors and states to each state and symbol:

$$\delta : \; q, f \to \varphi \in \mathcal{B}^+ \left([\mathrm{ar}(f)] \times Q \right).$$

Note that δ respects the arities, i.e., for a symbol f the only possible directions in the formula are $0, \ldots, \mathrm{ar}(f) - 1$.

Intuitively, a *correct run* of $\mathcal{A}$ on a tree $\mathcal{T}$ is a tree labeled with $\mathcal{T} \times Q$ where the successors of each node form a satisfying set for the Boolean condition related to the state in this node and to the corresponding letter in $\mathcal{T}$. More precisely, the root of a run is labeled with (r, q_0), where r is the root of $\mathcal{T}$. The first component π' of the label (π', q') of a successor of a vertex with label (π, q) must correspond to a child of π in the sense that $(\pi, \pi') \in S_i$ for some i, and the second component q' must be such that $(i, q') \in \delta(q, a)$ for the label a of π in $\mathcal{T}$. A run is *correct*, if for every vertex (π, q) the successors induce a set $E = \{(i_0, q_{i_0}), \cdots, (i_{l-1}, q_{i_{l-1}})\}$ that satisfies $\delta(q, a)$. A run is *accepting*, if all infinite paths in the run are accepting and all finite paths leading to a terminal vertex end with the label true.

For two alternating tree automata $\mathcal{A}_0$ and $\mathcal{A}_1$, we write $\mathcal{A}_0 \cap \mathcal{A}_1$ for the *intersection automaton*, which accepts all trees $\mathcal{T}$ that are accepted by both $\mathcal{A}_0$ and $\mathcal{A}_1$. The intersection automaton is constructed as follows: as a state space $Q^\cap := \{q_0\} \uplus Q_0 \uplus Q_1$ we use disjoint copies of the state spaces of $\mathcal{A}_0$ and $\mathcal{A}_1$ and a new initial state q_0. On Q_0 and Q_1 the transitions $\delta^\cap$ are defined as in the original automata. For every symbol a, we set the initial transition to $\delta^\cap(q_0, a) := \delta_0(q_0^0, a) \wedge \delta_1(q_0^1, a)$, where q_0^i is the initial state of $\mathcal{A}_i$. Clearly, a tree $\mathcal{T}$ is accepted by $\mathcal{A}_0 \cap \mathcal{A}_1$ if it is accepted by both $\mathcal{A}_0$ and $\mathcal{A}_1$. Notice that the intersection automaton is of size $|Q_0| + |Q_1| + 1$.

The *complement automaton* $\overline{\mathcal{A}}$ of an alternating tree automaton $\mathcal{A}$ is the automaton obtained by exchanging $\wedge$ and $\vee$, true and false in the transition relation, and by negating the acceptance condition. Thus, $\overline{\mathcal{A}}$ is of the same size as $\mathcal{A}$.

We will often use parity acceptance conditions. A parity condition is given by a priority coloring $\Omega : \; Q \to [d]$ for some $d \in \mathbb{N}$. It induces an acceptance set $\mathcal{F} \subseteq Q^\omega$ in the following way: a word $\alpha \in Q^\omega$ belongs to $\mathcal{F}$, if the minimal color seen infinitely often in $\Omega(\alpha)$ is even.

2.3 Tree automata for unfoldings and strategies

Lemma 1. *Given an arena $\mathcal{G}$ and an initial vertex v_0, there exists an alternating tree automaton $\mathcal{A}_{\mathcal{G}, v_0}$ of size $|V|$ which accepts a tree $\mathcal{T}(\mathcal{G}, v_0)$ if, and only if, the vertex labeling is correct, in the sense that $\pi = v_0 a_1 \cdots v_n$ is labeled with v_n.*

Proof. Let $\mathcal{A}_{\mathcal{G},v_0} = (Q, \delta, q^{v_0\epsilon}, \mathcal{F} = Q^\omega)$ be the automaton with $Q = \{q^v \mid v \in V\}$ and

$$\delta(q^v, v') = \begin{cases} \bigwedge_i (i, q^{\mathrm{succ}_i(v)}) & \text{if } v = v', \\ \mathsf{false} & \text{otherwise,} \end{cases}$$

where $(v, \mathrm{act}_i(v), \mathrm{succ}_i(v)) \in E$ is the i-th edge from v. Then a tree is accepted if false never occurs, i.e., if a run exists. By construction, this ensures that the labeling is correct. □

Lemma 2. *Given an arena $\mathcal{G}$ and an initial vertex v_0, there exists an alternating tree automaton $\mathcal{A}_S$ of size $|V|+3$ that accepts a tree $\mathcal{T}(\mathcal{G}, v_0)$ iff the vertex labeling is correct and it encodes a strategy of Player 0.*

Proof. We construct an automaton $\mathcal{A}' = (Q', \delta', q'_0, \mathcal{F} = Q^\omega)$ that checks the constraint on S, i.e., that every $v \in V_0$ has only one S-labeled successor, and every $v \in V_1$ has only S-labeled ones, and then take the intersection automaton with $\mathcal{A}_{\mathcal{G},v_0}$. Towards this, let $Q' = \{q_S, q_{\neg S}\}$, and define δ' as follows:

$$\begin{aligned} \delta'(q_S, vS) &= \begin{cases} \bigvee_i \left((i, q_S) \wedge \bigwedge_{j \neq i} (j, q_{\neg S}) \right) & \text{if } v \in V_0 \\ \bigwedge_i (i, q_S) & \text{if } v \in V_1 \end{cases} \\ \delta'(q_S, v) &= \delta'(q_{\neg S}, vS) = \mathsf{false} \\ \delta'(q_{\neg S,v}) &= \delta(q_S, vS) \end{aligned}$$

By construction, $\mathcal{A}'$ accepts if, and only if, the S-labeling encodes a strategy of Player 0. Thus, $\mathcal{A}_S := \mathcal{A}' \cap \mathcal{A}_{\mathcal{G},v_0}$. □

Lemma 3. *For an arena $\mathcal{G}$ with an initial vertex v_0, let W be a regular winning condition, and let $\mathcal{A} = (Q, \delta, q_0, \Omega)$ be a deterministic parity automaton that recognizes the complement $\overline{W}$. Then, there exists an alternating tree automaton $\mathcal{A}_1$ of size $|Q| + |V| + 4$ that recognizes all trees $\mathcal{T}(\mathcal{G}, v_0)$ labeled with a strategy of Player 0 against which Player 1 can win on $\mathcal{G}$.*

Proof. Notice that $\mathcal{A}$ accepts exactly the sequences of actions that are winning for Player 1. We construct an alternating tree automaton $\mathcal{A}'$, which updates and maintains the states of the word automaton along paths in the tree and immediately rejects when reaching a state that is not labeled with S. This automaton will then be intersected with those for the strategy and the vertex labeling. We define $\mathcal{A}' = (Q, \delta', q_0, \Omega)$ with the same state space and priority coloring as $\mathcal{A}$ and with transitions:

$$\begin{aligned} \delta'(q, vS) &= \bigwedge_i (i, \delta(q, \mathrm{act}_i(v))), \\ \delta'(q, v) &= \mathsf{false}. \end{aligned}$$

Now, set $\mathcal{A}_1 = \mathcal{A}' \cap \mathcal{A}_S$. A tree is accepted if, and only if, there exists a play consistent with S that is won by Player 1. □

3 Second-life games

In this section, we define the class of second-life games described in the introduction. Let $\mathcal{G} = (V, V_0, V_1, E)$ be an arena with actions over A, and let $T = \{t \in V : tE = \emptyset\}$ denote the set of terminal vertices in $\mathcal{G}$. The *second-life game* $\mathcal{S}(\mathcal{G}, W)$ is a game with the set of actions

$$A_{\|} := A \cup \{\mathsf{Return}\} \cup \{\mathsf{Call}(a) \mid a \in A\}.$$

and over the arena $(V \cup (V \times V), V_0', V_1', E')$, with

$$\begin{aligned}
&V_0' := V_0 \cup \{(u,v) \mid u \in V_0\} \cup \{(t,v) \mid t \in T, v \in V\}, \quad \text{and} \quad V_1' := V' \setminus V_0';\\
&E' := E \cup \{((u,v), a, (u',v)) \mid (u,a,v) \in E\}\\
&\qquad \cup \{(u, \mathsf{Call}(a), (v,v)) \mid u \in V_1, (u,a,v) \in E\} \qquad \text{(CALL)}\\
&\qquad \cup \{((t,v), \mathsf{Return}, v) \mid t \in T\} \qquad \text{(RETURN).}
\end{aligned}$$

Winning conditions for second-life games have the form $W \subseteq A_{\|}^{\omega}$.

Before explaining how second-life games are played, let us introduce the notion of a *a* Call-Return *sequence*, that is, a sequence of the form

$$u \cdot \mathsf{Call}(a) \cdot (v,v) \cdot a_1 \cdot (v_1, v) \cdots (t, v) \cdot \mathsf{Return} \cdot v.$$

Moves. Player 0 has the same moves as in the arena $\mathcal{G}$ from vertices in V_0, regardless of which component $\mathcal{G}$ or $(\mathcal{G}, v)$ the game is in. In contrast, at vertices in the main copy $\mathcal{G}$, Player 1 can decide to either take a move within the copy, or choose a Call move to a copy, for any successor.

Plays. A play α is a, possibly infinite, alternating sequence of vertices and edges, $\alpha = v_0 a_0 v_1 a_1 v_2 \cdots$, such that always $(v_i, a_i, v_{i+1}) \in E'$. A finite play always ends in some vertex v, and is won by Player 0. An infinite play is won by Player 0 if, and only if, its action sequence belongs to W.

Information. Player 0 is not informed about whether the current vertex is in the main copy or in some other component. Furthermore, after any Call-Return sequence, Player 0 will forget everything in between. Thus, for any finite path π starting at a vertex v in the main copy, we define the path $\hat{\pi}$ obtained by replacing every Call-Return sequence $u \cdot \mathsf{Call}(a) \cdots \mathsf{Return} \cdot v$ by $u \cdot a \cdot v$, then replacing the remaining last $\mathsf{Call}(a)$ by a (if such last call exists), and finally projecting every occurring (u, v) to u.

Strategies. Strategies of Player 1 are not restricted in any way. Strategies of Player 0 are functions $f : (V' A_{\|})^* V_0' \to A \times V$, such that, for any π ending in a vertex of Player 0, $f(\pi) = f(\hat{\pi})$, i.e., they respect the imperfect information constraint described above. A play $\alpha = v_0 a_0 v_1 a_1 v_2 \cdots$ is consistent with a strategy f of Player 0 if, for each $v_i \in V_0$, the next $(a_i, v_{i+1}) = f(v_0 a_0 \cdots v_i)$, and f is winning for Player 0 if all plays consistent with f are won by Player 0.

For every play α, note that $\hat{\alpha}$ is a play solely in the main copy. As W is given over $A_{\|}$, the desired behavior of Player 0 in some component $(\mathcal{G}, v)$ might

be different from that in the first life. Nonetheless, Player 0 has no information about whether he is moving in one of the components or in the main copy, and immediately after noticing that the play continues after a terminal (which means it must have been in a second-life component), forgets this, and everything that happened in the component.

Accordingly, any strategy of Player 0 can be viewed as a strategy over the vertex set V with actions A, i.e., as a strategy of Player 0 for the arena $\mathcal{G}$. Our main result regarding second-life games is that finite-memory strategies for the arena $\mathcal{G}$ suffice for Player 0 in the second-life game over $\mathcal{G}$.

Theorem 4 (Finite-memory winning strategies in second-life games). *Let $\mathcal{G}$ be an arena and W a regular winning condition. If Player 0 has a winning strategy in the second-life game $\mathcal{S}(\mathcal{G}, W)$, then he also has a finite-memory one.*

The proof of this theorem consists of several steps. As we already mentioned, strategies of Player 0 for $\mathcal{S}(\mathcal{G}, W)$ can be viewed as strategies for the arena $\mathcal{G}$, and vice-versa. We represent such strategies by labelings of trees.

For a deterministic parity automaton $\mathcal{A}$ over $A_{\|}$, a (q_1, c, q_2)-Return is a run of $\mathcal{A}$, starting in state q_1, ending in a state q such that $\delta(q, \mathsf{Return}) = q_2$, on which the minimal occurring priority (in the run and of q_2) is c.

Lemma 5. *Let $\mathcal{A}$ be a deterministic parity automaton over $A_{\|}$, with two designated states q_1, q_2, and let c be a priority. Then, there exists an alternating tree automaton $\mathcal{A}^C(q_1, q_2, c)$ of size $2 \cdot |Q^{\mathcal{A}}|$ which accepts a tree $\mathcal{T}(\mathcal{G}, v)$ labeled with a strategy S of Player 0 if there exists a path consistent with S from the root to a terminal that corresponds to a (q_1, c, q_2)-Return of $\mathcal{A}$.*

Proof. Notice that we do not require $\mathcal{A}^C(q_1, c, q_2)$ to check that S is indeed a strategy or that the V-labeling is correct (still, the automaton rejects if a non-S-vertex is seen). Thus, we only have to check that a run of $\mathcal{A}$ exists on which c is seen and no smaller priority is, and which starts in q_1 and ends so that a Return-action would result in q_2.

As a state space we use two disjoint copies of Q, and we denote elements from one by q, and the ones of the other by q^c. The idea is to store in the states whether c has already been seen (q^c) or not (q). By definition, if $\Omega(q_2) < c$, the automaton always rejects. If $\Omega(q_2) = c$, the automaton directly goes to the q^c-states in the first transition, otherwise the transitions are defined as in Figure 1.

As the automaton should only accept if a terminal is seen, we assign the priority 1 to all states. The initial state of the automaton is q_1.

By construction, an accepting run describes a path in the tree on which state labels correspond to the induced run of $\mathcal{A}$. A run can only be accepting if a transition to true occurs, thus only if a terminal is reached from where a Return leads to q_2, and the lowest priority of the (q_1, c, q_2)-Return is c (or $\Omega(q_2) = c$). □

Lemma 6. *For an arena $\mathcal{G}$ with an initial vertex v_0, let W be a regular winning condition and let $\mathcal{A}$ be a deterministic parity automaton recognizing $\overline{W}$. Then,*

$$\delta^C(q, v) = \mathsf{false}$$

$$\delta^C(q, vS) = \begin{cases} \mathsf{false} & \text{if } \Omega(q) < c \\ \mathsf{true} & \text{if } \Omega(q) = c,\ \delta(q, \mathsf{Return}) = q_2,\ v \text{ terminal} \\ \bigvee_i (i, \delta(q, \mathrm{act}_i(v))^c) & \text{if } \Omega(q) = c \text{ and not the above} \\ \bigvee_i (i, \delta(q, \mathrm{act}_i(v))) & \text{if } \Omega(q) > c \end{cases}$$

$$\delta^C(q^c, v) = \mathsf{false}$$

$$\delta^C(q^c, vS) = \begin{cases} \mathsf{false} & \text{if } \Omega(q) < c \\ \mathsf{true} & \text{if } \delta(q, \mathsf{Return}) = q_2 \text{ and } v \text{ is a terminal} \\ \bigvee_i (i, \delta(q, \mathrm{act}_i(v))^c) & \text{if } \Omega(q) \geq c \end{cases}$$

Fig. 1. Transitions for $\mathcal{A}^C$.

there exists an alternating tree automaton $\mathcal{B}$ of size $\mathcal{O}(|\mathcal{A}|^4)$ that accepts a tree $\mathcal{T}(\mathcal{G}, v_0)$ labeled with a strategy of Player 0 if, and only if, Player 1 can win in $\mathcal{S}(\mathcal{G}, W)$ against the strategy.

Proof. For every $q_1, q_2 \in Q^{\mathcal{A}}$ and every $c \in \Omega^{\mathcal{A}}(Q^{\mathcal{A}})$, let $\mathcal{A}^C(q_1, c, q_2)$ be the automaton from Lemma 5. The automaton $\mathcal{B}$ will only call the automata $\mathcal{A}^C(q_1, c, q_2)$, i.e., use them as black-boxes, so we do not describe them below.

The state space of $\mathcal{B}$ contains disjoint copies of the state spaces of the $\mathcal{A}^C(q_1, c, q_2)$, and, furthermore, the set of main states, which consists of triples

$$(q \in Q^{\mathcal{A}} \quad , \quad q \in Q^{\mathcal{A}} \cup \{\bot\} \quad , \quad c \in \Omega^{\mathcal{A}}(Q^{\mathcal{A}})).$$

The size of the state space is thus

$$|Q^{\mathcal{B}}| = \big((|Q^{\mathcal{A}}|^2 + |Q^{\mathcal{A}}|) \cdot |\Omega^{\mathcal{A}}(Q^{\mathcal{A}})|\big) + \big(2 \cdot |Q^{\mathcal{A}}|^3 \cdot |\Omega^{\mathcal{A}}(Q^{\mathcal{A}})|\big) = \mathcal{O}(|\mathcal{A}|^4). \quad (1)$$

Intuitively, $\mathcal{B}$ inspects all plays consistent with S in $\mathcal{S}(\mathcal{G}, W)$ and models Call-Return sequences in such a way that, at V_1-vertices, a play can either continue without a Call (i.e., as in $\mathcal{G}$), or a Call occurs. In the latter case, the state q_2 reached after the Return is guessed, and so is the minimal priority on the Call-Return sequence. Such a guess has to be provable – it is checked via $\mathcal{A}^C$, and the game continues from q_2. An accepting run of $\mathcal{B}$ thus describes a play in $\mathcal{S}(\mathcal{G}, W)$, where the Call-Return sequences are given by the sub-runs of $\mathcal{A}^C$-automata, and the main states describe the main part of the play.

More formally, let us first say that, as before, the automaton $\mathcal{B}$ will reject as soon as a non-S-labeled vertex is reached: $\delta^{\mathcal{B}}(p, v) = \mathsf{false}$ for all $p \in Q^{\mathcal{B}}$. As we copy the automata $\mathcal{A}^C$, the transition function on the respective states are as in the original automata.

For vertices $v \in V_0$, the automaton moves to the S-labeled successor and updates the first entry of the triple according to $\mathcal{A}$:

$$\delta^{\mathcal{B}}((q, \bot, c), vS) = \bigvee_i (i, (\delta^{\mathcal{A}}(q, \mathrm{act}_i(v)), \bot, c)).$$

For vertices $v \in V_1$, the automaton can choose any successor and proceed as for V_0, but, to model Call-Return sequences, it can also be claimed that such a sequence with minimal priority c ending in state q_2 after the return and consistent with the strategy exists:

$$\delta^{\mathcal{B}}((q,\bot,c),vS) = \bigvee_i (i,(\delta^{\mathcal{A}}(q,\mathrm{act}_i(v)),\bot,c)) \\ \vee \bigvee_i \bigvee_{q_2,c'} (i,(\delta^{\mathcal{A}}(q,\mathsf{Call}(\mathrm{act}_i(v))),q_2,c')).$$

If the second entry of the state triple is not $\bot$, the transition function is the same for all $v \in V$ (as we define it using the two different transitions above):

$$\delta((q,q_2,c),vS) = \delta^{\mathcal{B}}((q_2,\bot,c),vS) \wedge \delta^{\mathcal{A}^C(q,c,q_2)}(q,vS).$$

For the priority coloring, we set $\Omega^{\mathcal{B}}((q,q_2,c)) = c$, $\Omega^{\mathcal{B}}(q,\bot,c)) = \Omega^{\mathcal{A}}(c)$ and otherwise copy the coloring of the respective $\mathcal{A}^C(q_1,c,q_2)$.

Claim. $\mathcal{B}$ accepts a tree $\mathcal{T}(\mathcal{G},v_0)$ labeled with a strategy S_0 of Player 0 if, and only if, Player 1 wins the game $\mathcal{S}(\mathcal{G},W)$ from v_0 against the strategy S_0.

(⇒) Let ρ be an accepting run of $\mathcal{B}$. Player 1 follows, in the play in $\mathcal{S}(\mathcal{G},W)$, the run of the automaton and, whenever a transition with a state q_2 as second entry occurs, plays the corresponding Call-Return sequence (given by the subrun of $\mathcal{A}^C$). Afterwards, he continues as in the main part of the run. For the resulting play, it follows, as q_2 is colored with the smallest priority appearing in the Call-Return sequence, that the minimal priority seen infinitely often corresponds to the minimal priority seen infinitely often in the accepting run (and thus in $\mathcal{A}$, as there are only finitely many priorities), and therefore the play is won by Player 1.

(⇐) Let S_1 be a strategy of Player 1 with which he wins against S_0 in $\mathcal{S}(\mathcal{G},W)$, and let $\alpha(S_0,S_1)$ be the corresponding unique consistent play from v_0. Then $\alpha(S_0,S_1)$ induces an accepting run of $\mathcal{B}$ on $\mathcal{T}(\mathcal{G},v_0)$ labeled with S_0 by the following construction. At positions in V_0, choose the transition that follows the strategy S_0. At positions in V_1, if the play continues, by S_1, with an action $a \in A$, take the appropriate transition in the automaton. If a Call occurs, take the Call action in which the state after the next return is correctly guessed, and so is the minimal priority of the part in between. Then verify this claim via $\mathcal{A}^C$ and continue the run as described above from after the next Return in $\alpha(S_0,S_1)$. By definition of $\mathcal{B}$, it follows that all paths in the run are accepting, thus $\mathcal{B}$ accepts the S_0-labeled tree $\mathcal{T}(\mathcal{G},v_0)$. □

3.1 Proof of Theorem 4

We use the above lemmas to show that Player 0 has a finite-memory winning strategy (if he has one at all) by building a nondeterministic parity tree automaton accepting the winning strategies of Player 0. We then use properties of such

automata to conclude that finite-memory winning strategies exist if the language of the automaton is non-empty. One can also prove this in an alternative way, using MSO compatibility of graph unfoldings, see Appendix A.

Proof (Theorem 4). Let $\mathcal{B}^* = \overline{\mathcal{B}} \cap \mathcal{A}_S$ be an alternating tree automaton. By definition of $\mathcal{B}$ and $\mathcal{A}_S$, $\mathcal{B}^*$ accepts exactly those strategy-labeled trees $\mathcal{T}(\mathcal{G}, v_0)$ that describe winning strategies of Player 0 in $\mathcal{S}(\mathcal{G}, W)$. Recall that $|\mathcal{B}| = \mathcal{O}(|\mathcal{A}|^4)$ by (1), where $\mathcal{A}$ is the deterministic parity automaton recognizing $\overline{W}$, and thus $\mathcal{B}^*$ is of size $\mathcal{O}(|\mathcal{A}|^4 + |V| + 3)$. By [11], since the Rabin pairs of a parity condition form a chain, $\mathcal{B}^*$ can be turned into a nondeterministic parity tree automaton $\mathcal{A}^*$ of size $2^{\mathcal{O}(|\mathcal{B}^*|^2 \log |\mathcal{B}^*|)}$.

The automaton $\mathcal{A}^*$ checks that a given strategy labeling corresponds to a winning strategy of Player 0. However, as the automaton is nondeterministic, one can also construct a nondeterministic automaton $\mathcal{A}^\dagger$, of size $\mathcal{O}(|\mathcal{A}^*|)$ that guesses the labeling S and simulates $\mathcal{A}^*$ on the guess, i.e., verifies that the guess corresponds to a winning strategy.

Consider the product of $\mathcal{A}^\dagger$ with the arena $\mathcal{G}$. This results in a parity game in which the existential player has a winning strategy if he can guess a labeling S on the unraveling of $\mathcal{G}$, i.e., a labeling S of $\mathcal{T}(\mathcal{G}, v_0)$ that corresponds to a winning strategy. Since parity games are positionally determined [4,10], there is a positional winning strategy in this parity game if there is one at all. This positional strategy of Player 0 on the product of $\mathcal{A}^\dagger$ and $\mathcal{G}$ describes a strategy on $\mathcal{G}$ that uses memory $\mathcal{A}^\dagger$ of size

$$|\mathcal{A}^\dagger| = 2^{\mathcal{O}(|\mathcal{B}^*|^2 \log |\mathcal{B}^*|)} = 2^{\mathcal{O}(|\mathcal{A}|^8 \log |\mathcal{A}| + |V|^2 \log |V|)} = 2^{\mathcal{O}(|\mathcal{A}|^9 + |V|^3)}. \quad (2)$$

□

4 Counter parity games

In this section, we use second-life games and the existence of finite-memory winning strategies to solve counter parity games. These are quantitative parity games [5] with a finite set of counters that are updated along edges by affine transformations, and which are used to determine the payoff of finite plays. Counter parity games are a strict generalization of the counter-reset games used in [6] to approximate a quantitative logic over a class of hybrid systems. Furthermore, model-checking games of other quantitative logics for structure rewriting systems can be reduced to counter parity games [8].

To define counter parity games, let us fix a natural number k, and let $\mathcal{F}_k$ be the set of k-dimensional affine functions $f : \mathbb{N}^k \to \mathbb{N}^k$ with $f(c) = A \cdot c + B$, for natural matrices A, B. A *counter parity game* $\mathcal{G} = (V, V_{\max}, V_{\min}, E, \Omega, \lambda)$ with k counters is played by two players, Maximizer and Minimizer, on a directed graph (V, E). The vertex set is partitioned into vertices $V_{\max}$ of Maximizer and vertices $V_{\min}$ of Minimizer. Vertices are colored by the priority function $\Omega : V \to \{0, \cdots, d-1\}$, edges are labeled with affine functions, i.e., $E \subseteq V \times \mathcal{F}_k \times V$, and terminal vertices are labeled by $\lambda : V \to \{+, -\} \times \{0, \cdots, k-1\}$.

The k counters are represented by a vector $c \in \mathbb{N}^k$ of k natural numbers. We write c_i for the i-th component of c, i.e., the i-th counter. At the beginning of a play, all counters are 0, thus $c = 0^k$. Throughout a play, counters are updated according to the edge labels, i.e., if the current value of the counter vector is c and an edge (u, f, v) is taken, then the new value is $f(c)$. As usual, Maximizer moves at positions $V_{\max}$, while Minimizer moves at $V_{\min}$.

Counter parity games are games of perfect information, thus, in contrast to second-life games, there is no constraint on the strategies. The objective of Maximizer is to maximize the payoff, while Minimizer's goal is to minimize it. For finite plays π, the payoff $p(\pi)$ is determined by λ in the terminal vertex t: it is $s\, c_i$ if $\lambda(t) = (s, i)$ and the current vector is c. For infinite plays, the payoff is $-\infty$ if the minimal priority that has been seen infinitely often is odd, and ∞ otherwise.

A counter parity game $\mathcal{G}$ is *determined*, if the supremum of the payoffs Maximizer can achieve and the infimum of the payoffs that the Minimizer cannot avoid coincide, that is,

$$\sup_{f \in \Sigma_{\max}} \inf_{g \in \Sigma_{\min}} p(\alpha_{f,g}(v)) = \inf_{g \in \Sigma_{\min}} \sup_{f \in \Sigma_{\max}} p(\alpha_{f,g}(v)) =: \operatorname{val} \mathcal{G}(v),$$

where $\Sigma_{\max}$ ($\Sigma_{\min}$) is the set of strategies of Maximizer (Minimizer), while $\alpha_{f,g}(v)$ is the unique play consistent with f and g.

As counter parity games are a special case of quantitative parity games on infinite arenas (we can encode counter values in the vertices and adjust the edges accordingly), and it was shown in [5] that quantitative parity games are determined on arenas of arbitrary size, we obtain the following corollary.

Corollary 7. *Every counter parity game $\mathcal{G}$ is determined: the value* $\operatorname{val} \mathcal{G}(v)$ *exists, for each vertex v.*

However, from this results it does not follow that the value of counter parity game can actually be computed. This is what we will prove in the next section.

5 Solving counter parity games

In this section, we presents an algorithm for solving counter parity games.

Theorem 8. *For any finite counter parity game $\mathcal{G}$ with initial vertex v, the value* $\operatorname{val} \mathcal{G}(v)$ *can be computed in* 6EXPTIME. *When the number of counters is fixed, the value can be computed in* 4EXPTIME.

We present the proof in three steps. In the first step, we describe an abstraction, where we introduce marks for update functions. This later allow us to gather important information about how counters change after applying a sequence of updates. In fact, we solve counter parity games not only for linear counter update functions, but for arbitrary ones which allow to be marked.

In the second step, we construct, with the help of the introduced marks, a second-life game with a regular winning condition which allows us to check

whether the value of the game is ∞ or bounded by a computable constant. This will be done in both the positive and negative direction, i.e., for ∞ and $-\infty$, and it will provide us either with the precise value of the game, or with a lower and an upper bound.

In case the second step only provides bounds for the value, we reduce the problem of finding the precise value to the solution of a finite game without counters. This is done in the third step.

5.1 Counter updates and their marks

Let us fix a dimension k of the counter vector $c = \langle c_0, \cdots, c_{k-1} \rangle \in \mathbb{N}^k$. We consider *counter update functions* $f : \mathbb{N}^k \to \mathbb{N}^k$ which allow to be *marked* in the following way.

A *mark* is a function $m : \{0,1\}^k \times [k] \to \{\bot\} \cup [k] \cup \mathcal{P}([k])$. A function $f : \mathbb{N}^k \to \mathbb{N}^k$ has mark m if the following holds for all $c \in \mathbb{N}^k, i \in [k]$.

(i) If $m(c^{>0}, i) = \bot$ then $f(c)[i] = 0$.
(ii) If $m(c^{>0}, i) = j \in [k]$ then $f(c)[i] = c_j$.
(iii) If $m(c^{>0}, i) = D \in \mathcal{P}([k])$ and $D \neq \emptyset$ then $f(c)[i] > \max_{j \in D} c_j$.
(iv) If $m(c^{>0}, i) = \emptyset$ then $f(d)[i] = C > 0$ is constant for all d with $d^{>0} = c^{>0}$.
(v) $f(c)[i]$ depends only on the counters from $m(c^{>0}, i) = D$,
i.e., there exists a function f'_i such that $f(c)[i] = f'_i(c|^D)$.

Note that (iv) could be seen as special case of (v), but we distinguish whether the constant is 0, as in (i), or not. Intuitively, a mark determines, depending on which counters are 0 and which are not, whether the result will be 0, always stay equal to another counter, or increase over some other. In particular, if $m(d, i) = D$ then, after applying the counter update function, the counter i will be strictly bigger than each of the counters from D. To capture the set of counters which c_i will be greater or equal to after update, we write

$$m^{\geq}(d,i) = \begin{cases} \emptyset & \text{if } m(d,i) = \bot, \\ \{l\} & \text{if } m(d,i) = l \in [k], \\ D & \text{if } m(d,i) = D \in \mathcal{P}([k]). \end{cases}$$

Additionally, we write $m^{>0}(c^{>0})$ for the vector $d^{>0}$ if d results from the application of a function f with mark m to the vector c. Observe that $f(c)[i] = 0$ if, and only if, $m(c^{>0}, i) = \bot$ or $m(c^{>0}, i) = l$ and $c_l = 0$, and thus $m^{>0}(c^{>0}) = f(c)^{>0}$ is computable from $c^{>0}$ and m.

Example 9. Consider two counters c_0, c_1 and the update function f assigning $c_0 + c_1$ to c_0 and $2 \cdot c_0$ to c_1. This function has the following mark m: $m(0,0,i) = \bot$, $m(0,1,0) = 1$ as $c_0 + c_1 = c_1$ if $c_0 = 0$, and $m(1,0,0) = 0$ analogously; $m(0,1,1) = \bot$ as $2 \cdot 0 = 0$, but $m(1,0,1) = m(1,1,1) = \{0\}$ as $2 \cdot c_0 > c_0$ for $c_0 > 0$. Finally, $m(1,1,0) = \{0,1\}$ as c_0 exceeds both counters in this case.

Note that not all functions allow to be marked. For example, if we updated c_1 to $c_0 \cdot c_1$ above, we would not be able to assign a mark. In particular, $m(1,1,i)$ is not definable, because, whether the counter increases or stays unchanged depends on whether $c_i > 1$ and not just on whether $c_i > 0$. The methods we present generalize to more involved markings, but we do not introduce them here as we are interested in one particular class of functions, for which the above marks suffice.

Lemma 10. *Let $f : \mathbb{N}^k \to \mathbb{N}^k$ be an affine function, i.e., there exist $a^i_j, b^i \in \mathbb{N}$ such that $f(c)[i] = \sum_j a^i_j \cdot c_j + b^i$ for all $c \in \mathbb{N}^k$. Then there exists a mark m_f for the function f.*

Proof. To compute $m_f(d,i)$, let $D = \{n \mid d_n = 1 \text{ and } a^i_n > 0\}$. Observe that, for all c with $c^{>0} = d$, it holds $f(c)[i] = \sum_{j \in D} a^i_j c_j + b^i$. Thus, when we set

$$m_f(d,i) = \begin{cases} \bot & \text{if } D = \emptyset \text{ and } b^i = 0, \\ l & \text{if } D = \{l\} \text{ and } b^i = 0 \text{ and } a^i_l = 1, \\ D & \text{in all other cases,} \end{cases}$$

then conditions (i)-(v) follow. □

One important property of marks is that, when functions are composed, their marks can be composed as well.

Lemma 11. *Let f_1 and f_2 be counter update functions with marks m_1 and m_2. Then, a mark $m = m_1 \circ m_2$ for $f(c) = f_2(f_1(c))$ can be computed.*

Proof. Recall that $f_1(c)^{>0} = m_1^{>0}(c^{>0})$ is computable from $c^{>0}$ and m_1. Consider the following cases.

(1) $m_2(f_1(c)^{>0}, i) = \bot$. In this case $m(c^{>0}, i) = \bot$.
(2) $m_2(f_1(c)^{>0}, i) = l$, i.e., $f_2(f_1(c))[i] = f_1(c)[l]$. Then $m(c^{>0}, i) = m_1(c^{>0}, l)$.
(3) $m_2(f_1(c)^{>0}, i) = D$. In this case $m(c^{>0}, i) = \bigcup_{j \in D} m_1^{\geq}(c^{>0}, j)$. □

Let us denote by $\mathcal{M}$ the set of all marks, which is finite by definition. For a fixed number k, by definition, $|\mathcal{M}| \leq \left(2^k + k + 1\right)^{2^{k+\log k}} = 2^{2^{\mathcal{O}(k)}}$. Moreover, by the above lemma, $\circ$ induces a computable finite semigroup structure on $\mathcal{M}$. It follows that languages of sequences of marks with definable properties are regular. For example, the language of all sequences $m_0 m_1 \dots m_n \in \mathcal{M}^*$ such that $m = m_1 \circ \dots \circ m_n$ satisfies, for a fixed C, i and d, that $C \subseteq m^{\geq}(d,i)$ or $m(d,i) \cap C \neq \emptyset$, is regular. This means that, for a fixed set of counters C and starting information about which counter is 0, we can determine in a regular fashion whether c_i at the end will be at least as big as some counter from C.

As mentioned above, we use these marks as a level of abstraction. Accordingly, we extend counter parity games by marks in the following way.

Let $\mathcal{G}$ be a counter parity game with k counters. The *marked counter parity game* $\mathcal{G}_m = (V_m, V'_{\max}, V'_{\min}, E_m, \Omega_m, \lambda_m)$ is a game with k counters defined as:

– $V_m := V \times \{0,1\}^k$ (storing which counters are greater than 0).
– $V'_{\max} = \{(v, c^{>0}) \in V_m \mid v \in V_{\max}\}$, $V'_{\min} = V_m \setminus V'_{\max}$.
– $E_m \subseteq V_m \times (\mathcal{F} \times \mathcal{M}) \times V_m$ stores the marks and updates the $c^{>0}$-vectors:

$$E_m := \{((u, c^{>0}), (f, m_f), (v, m_f^{>0}(c^{>0}))) \mid (u, f, v) \in E,\ c^{>0} \in \{0,1\}^k\}.$$

– $\Omega_m(v, c^{>0}) = \Omega(v)$ and $\lambda_m(v, c^{>0}) = \lambda(v)$.

5.2 The unboundedness game

In the next step, we take a marked counter parity game and check whether its value is unbounded, i.e., ∞, or not. To do this, we transform the marked game into a second-life game, where Minimizer takes the role of Player 0.

From the definition of the value of a counter parity game, there are two ways for the value to be ∞: Maximizer could have a winning strategy with respect to the parity condition, or he could have a sequence $f_0, f_1, \cdots$ of strategies which ensure arbitrarily high payoffs. Via the reduction to second-life games, we combine this sequence of strategies for the latter case into a single strategy. Intuitively, Maximizer will get the option to decide to try to reach a terminal position to "save" a payoff, and then continue increasing the counters. If, in such a game, Maximizer has a strategy to save higher and higher payoffs, or to win via the parity condition, this corresponds to a value of ∞. We exploit that marks form a finite semigroup to show that this can be formulated as a regular objective. Intuitively, the reason why we consider second-life games with imperfect information and recall for Minimizer is that we need to avoid that Minimizer learns about whether Maximizer attempts to win by parity or by reaching arbitrarily high payoffs. If Minimizer had this information, he could adapt his strategy and neglect the other way of ensuring payoff ∞.

Let $\mathcal{G}_m$ be a marked counter parity game with arena $\mathcal{G}$ and terminal vertices T. The unboundedness game $\mathcal{G}^u$ is the second-life game $\mathcal{S}(\mathcal{G}_m, W)$ using $V_0 := V_{\min}$ and $V_1 := V_{\max}$ and with the winning condition W described below.

Recall that, if we remove all Call-Return sequences from a path in $\mathcal{G}^u$, we obtain a path in $\mathcal{G}_m$ that we call the *main part*. Due to better readability, we describe the winning condition in terms of both edge- and vertex-labels (functions/marks and priorities, respectively). Technically, this can be avoided by adding an appropriately colored vertex to each edge.

We describe the winning condition for Maximizer, i.e., Player 1, which is sufficient since regular languages are closed under complementation. Maximizer wins a play α if, and only if, the main copy is visited infinitely often, no terminal vertex inside the main copy is seen, and

– the main part satisfies the parity condition of $\mathcal{G}_m$, or
– there exists a counter d such that, from some point onwards, counter d is increased in the main part, then a Call is taken and a Return from a terminal where a payoff greater than d would be obtained in the original counter game, and after the Return this is repeated, ad infinitum.

By properties of marks, finite sequences of marks after which a counter d has been increased form a regular language $d\nearrow$. Also, finite sequences starting with a Call and ending with a Return from a vertex with $\lambda = c$ such that, for the sequence of marks in between, counter c is, at the end, greater than d at the beginning, form a regular language $c^{>d}$. Thus, the later part of W is the union of the main part satisfying the parity condition and the play being of the form $(V'A_{\|})^* \cdot (d\nearrow \cdot c^{>d})^\omega$. This is ω-regular.

Let us calculate the size of the automaton for the above condition. To check the language $d\nearrow$, $|\mathcal{M}|$ states suffice for a deterministic finite word automaton (using composition on the marks), and the same holds for $c^{>d}$. Checking $d\nearrow \cdot c^{>d}$ can thus be done with $\mathcal{O}(|\mathcal{M}|)$ many states by a non-deterministic automaton. By considering Büchi acceptance with the same accepting states, we get a Büchi automaton for the language $(d\nearrow \cdot c^{>d})^\omega$ with $\mathcal{O}(|\mathcal{M}|)$ states. If we add a new initial state and take a copy of the automaton for $(d\nearrow \cdot c^{>d})^\omega$ for every $d < k$, we build a nondeterministic Büchi automaton of size $\mathcal{O}(k \cdot |\mathcal{M}|)$ which accepts a play if it is won via some counter. (It waits in the initial state until the actual d is correctly guessed and then moves to the respective copy.) For the parity part, we need an automaton of size $|\Omega(V)| < |V|$. Taking the union of the two, we get a non-deterministic parity automaton of size $\mathcal{O}(|V| + k|M|)$ and index $|V|$. After determinization, the deterministic parity automaton for W has size $2^{\mathcal{O}(|V|(|V|+k|\mathcal{M}|)\log(|V|+k|M|))} = 2^{\mathcal{O}((|V|+k|\mathcal{M}|)^3)}$.

Combining this with Equation 2 from the proof of Theorem 4, we get that if Minimizer has a winning strategy for the unboundedness game, then also one with memory of size

$$K_0 := 2^{\mathcal{O}\left(\left(2^{\mathcal{O}((|V|+k|\mathcal{M}|)^3)}\right)^9 + |V|^3\right)} = 2^{2^{\mathcal{O}\left((|V|+k|\mathcal{M}|)^3\right)}}. \tag{3}$$

What remains to be shown is the connection between the value of $\mathcal{G}$ and the existence of a winning strategy of Minimizer in $\mathcal{G}^u$. We will use Ramsey's theorem for a finite path π in $\mathcal{G}^u$ or $\mathcal{G}$ played consistently with a strategy using memory of size $\leq K_0$. We write π as a sequence of vertices and memory states with edges labeled by the corresponding marks:

$$\pi = (v_0, q_0) \xrightarrow{m_0} (v_1, q_1) \xrightarrow{m_1} (v_2, q_2) \cdots \xrightarrow{m_{n-2}} (v_{n-1}, q_{n-1}),$$

where each v_i is a vertex and each q_i is a memory state (and $q_{i+1} = \text{update}(q_i, v_i)$ according to the strategy). The path π induces a complete edge-colored undirected graph over $[n]$, where an edge i, j is colored by (m, v_i, q_i, v_j, q_j), where m is the composition of the marks $m_i \circ m_{i+1} \circ \cdots \circ m_{j-1}$. Let l be the number of such colors for $\mathcal{G}^u$ and memory size K_0:

$$l = |\mathcal{M}| \cdot |V_u|^2 \cdot K_0^2 = |\mathcal{M}| \cdot K_0^2 \cdot (|V_m| \cdot |V_m|)^2 = |\mathcal{M}| \cdot K_0^2 \cdot (|V| \cdot 2^k)^4.$$

As K_0 is already doubly exponential, it dominates this product and thus l is also doubly exponential. We write $\mathcal{R} = R(\underbrace{3, 3, \cdots, 3}_{l \text{ times}})$ for the Ramsey-number for 3-cliques with l colors. As $\mathcal{R} \leq 3l!$ [14], we get that $\mathcal{R} = 2^{2^{2^{\mathcal{O}\left((|V|+k|\mathcal{M}|)^3\right)}}}$.

Before we state the proposition about the connection between winning strategies and values, we list some properties of idempotent marks, i.e., of marks m such that $m = m \circ m$. We write $i \leqq m(c^{>0}, j)$ if $i \in m(c^{>0}, j)$ or $i = m(c^{>0}, j)$.

Lemma 12. *Let m be an idempotent mark. Then, for all initial values c, and all $i < k$: if $i \not\leqq m(c^{>0}, i)$, then i will not appear in any $m(c^{>0}, j)$.*

Proof. Assume that i does appear in some $m(c^{>0}, j)$. As c_i does not depend on c_i in one application of m, after a second one, i will not appear in $m(c^{>0}, j)$, which contradicts m being idempotent. □

In the following, we show that if Minimizer has a finite-memory winning strategy for $\mathcal{G}^u$, then the value of $\mathcal{G}$ is bounded from above. Otherwise, it is ∞.

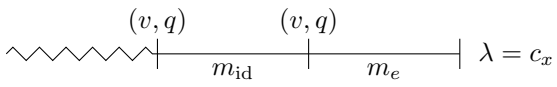
Proposition 13. *There exists a constant K, computable from $\mathcal{G}$, such that if Minimizer has a strategy ensuring a win from v in the main copy of $\mathcal{G}$ in $\mathcal{G}^u$ then* $\operatorname{val}\mathcal{G}(v) < K$. *In the other case,* $\operatorname{val}\mathcal{G}(v) = \infty$.

Proof. Consider the set of paths of length greater than $\mathcal{R}$ for a memory of size K_0. Set K to the maximal counter value plus 1 occurring anywhere on any of these paths when starting with initial counter values $c = (a, a, \cdots, a)$, where a is the maximal number occurring in any update function's matrices A or B. A rough upper bound can be computed as follows: after one application of any of the update functions, the maximal value is at most $a \cdot a \cdot k$ (the sum of all counters initialized with a, each weighted with a). After two steps, we get at most $k \cdot a \cdot k \cdot a \cdot a = k^2 a^{2+1}$. After $\mathcal{R}$ steps, we thus get $K \leq k^{\mathcal{R}} a^{\mathcal{R}+1} + 1$.

Assume first that Minimizer has a winning strategy ρ in $\mathcal{G}^u$. Note that, because of imperfect information and imperfect recall, ρ can also be viewed as a strategy for $\mathcal{G}$. Consider thus, towards a contradiction, a play α in $\mathcal{G}$ that is consistent with ρ and that has a payoff $\geq K$. Let further β be the corresponding play in $\mathcal{G}^u$ in which Maximizer never takes a Call, i.e., which consists only of a main part. We distinguish two cases: if α is infinite, then so is β. Because ρ is a winning strategy for Minimizer in $\mathcal{G}^u$, β – and thus α – violates the parity condition. But then the payoff for α is $-\infty$, a contradiction.

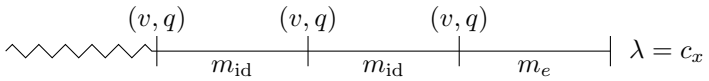
If α is finite, but has a payoff $\geq K$, we first prove the following claim.

Claim. Every play consistent with ρ and with payoff $\geq K$ has a suffix $(*)$:



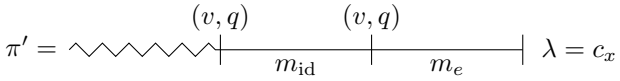
with m_{id} idempotent, such that for some j with $j \leqq m_{\mathrm{id}} \circ m_e(c_x)$: $j \in m_{\mathrm{id}}(j)$.

Proof. Let π be the shortest path with payoff $\geq K$ such that no suffix $(*)$ exists. By choice of K, there exists at least one suffix



as, by Ramsey's theorem, there exist indices $i_0 < i_1 < i_2$ such that the marks of $\pi[i_0 \cdots i_1], \pi[i_1 \cdots i_2]$ and $\pi[i_0 \cdots i_2]$ coincide. Fix the last such suffix in π. As $(*)$ is not realized, for all $j \leqq m_{\mathrm{id}} \circ m_e(c_x)$ it holds that $j \notin m_{\mathrm{id}}(j)$.

Let π' be the path obtained from π by removing the second $(v,q)-m_{\mathrm{id}}-(v,q)$-part, i.e.,



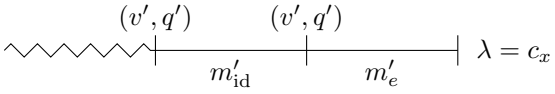
We show that

(a) the payoff of π' equals the payoff of π, i.e., is $\geq K$, and
(b) no suffix $(*)$ exists in π'.

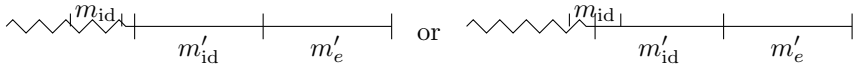
Together, (a) and (b) provide a contradiction to π being the shortest such path, which proves the claim.

Proof of (a). Let $j \leqq m_{\mathrm{id}} \circ m_e(c_x)$. By choice of π, $j \notin m_{\mathrm{id}}(j)$. Thus, $j = m_{\mathrm{id}}(j)$ or $j \notin m_{\mathrm{id}}(i)$ for all i (by Lemma 12). If $j \notin m_{\mathrm{id}}(i)$ for all i, then the value of j is lost during m_{id}. But then $j \notin m_{\mathrm{id}} \circ m_e(c_x)$, a contradiction. It follows that $j = m_{\mathrm{id}}(j)$ for all $j \leqq m_{\mathrm{id}} \circ m_e(c_x)$. But, as every counter c_x depends on is left untouched during m_{id}, the value of π' equals that of π.

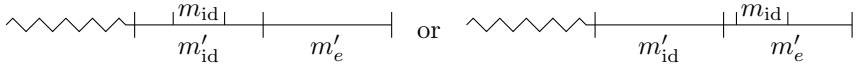
Proof of (b). Assume that there exists a suffix in π' for which $(*)$ holds, i.e.,



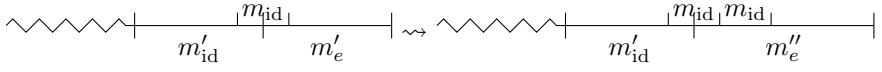
such that there exists some $j \leqq m'_{\mathrm{id}} \circ m'_e(c_x)$ with $j \in m'_{\mathrm{id}}(j)$. We show that this leads to a contradiction. There are several different cases.

(1) 

When reinserting the second $(v,q) - m_{\mathrm{id}} - (v,q)$-part, we can simply shift the $m'_{\mathrm{id}}m'_e$-part to the right, thus $(*)$ would hold in π.

(2) 

As m_{id} is idempotent, repeating it does not change m'_{id}, or respectively m'_e, thus $(*)$ also holds in π.

(3) 

We have to show that $m'_{\mathrm{id}} \circ m'_e(c_x) = m'_{\mathrm{id}} \circ m''_e(c_x)$. If this holds, then $(*)$ holds in π. But because $m_{\mathrm{id}} \circ m_{\mathrm{id}} = m_{\mathrm{id}}$, also $m'_{\mathrm{id}} \circ m'_e = m'_{\mathrm{id}} \circ m''_e$.

By the above, it follows that π' does not have a suffix for which $(*)$ holds. $\square$

By the above claim, it follows that α contains a cycle which can be repeated arbitrarily many times by Maximizer. As repeating the cycle increases the payoff, repeating the cycle, taking a Call towards the Return, then repeating the cycle and taking the Call again, and so on, is a witness for a win of Maximizer in $\mathcal{G}^u$. Because of imperfect information and imperfect recall, this witness is consistent with ρ, contradicting the assumption that ρ is a winning strategy of Minimizer.

Assume now that Minimizer does not have a winning strategy. This means that, for any strategy ρ of Minimizer, there exists a consistent play $\alpha(\rho)$ won by Maximizer. Note that $\mathcal{G}^u$ is not necessarily determined, but $\mathcal{G}$ is determined (cf. Corollary 7). Thus, it suffices to show that, for any strategy ρ of Minimizer and any natural number $N \in \mathbb{N}$, Maximizer has a strategy to ensure a payoff $> N$ against ρ in $\mathcal{G}$. Recall that strategies of Minimizer in $\mathcal{G}$ correspond to strategies in $\mathcal{G}^u$. Let thus ρ and N be given. Maximizer can play as follows: play as in $\alpha(\rho)$ until the first Call occurs. Skip the Call-Return sequence. If $\alpha(\rho)$ is won via the parity condition, do this infinitely often. Otherwise, wait until the winning counter d has reached a value $> N$ and a Call occurs. Take the Call and realize the payoff as required. □

5.3 Proof of Theorem 8

Notice that the construction of the unboundedness game also works in the dual case, i.e., for Maximizer, in order to determine a lower bound on the value, or a value of $-\infty$, respectively.

Proof (Theorem 8). Given a game $\mathcal{G}$, compute the marked game, and the corresponding unboundedness game for Minimizer. Determine whether the value is ∞, and compute an upper bound K^+ otherwise. Dually, construct the unboundedness game for Maximizer and check for a value of $-\infty$ or compute a lower bound K^-. Set $K := \max(|K^+|, |K^-|)$. Note that it follows from the above proof that $K \leq k^{\mathcal{R}} a^{\mathcal{R}+1} + 1$. Then, the value of $\mathcal{G}$ satisfies $-K < \operatorname{val} \mathcal{G}(v) < K$.

As the only way for counters to decrease is to be overwritten by a constant or a smaller counter, we can limit counter values to numbers $\leq K$ and store $\top$ otherwise. This results in a quantitative parity game of size $K \cdot |V|$ and with $\mathcal{O}(|V|)$ priorities. Such games can be solved in time $\mathcal{O}((K \cdot |V|)^{|V|})$ [5]. Since $\mathcal{R}$ is triply exponential in $|V| + k|\mathcal{M}|$ and $|\mathcal{M}|$ is doubly exponential in k, this gives a 4EXPTIME solution for a fixed k and a general solution in 6EXPTIME. □

6 Conclusion

Games with imperfect recall have been studied in mathematical game theory, but raised little interest in computer science so far. We show that already the class of second-life games, which exhibit a basic form of imperfect recall, can have interesting algorithmic applications. The result that finite-memory strategies are sufficient for winning second-life games allows to derive a bound for counter parity games, and gives the first elementary algorithm for the model-checking

problem studied in [6]. This application opens up many new questions. First of all, can other classes of ω-regular games with imperfect recall be solved algorithmically? And can these be applied in other already studied problems? The results above motivate further study of ω-regular games with imperfect recall.

References

1. A. Arnold and I. Walukiewicz. Nondeterministic controllers of nondeterministic processes. In *Logic and Automata*, volume 2. Amsterdam University Press, 2007.
2. K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games of incomplete information. *LMCS*, 3(3:4), 2007.
3. T. Colcombet and C. Löding. Regular cost functions over finite trees. In *Proc. of LICS '10*, pages 70–79, 2010.
4. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proc. of FOCS '11*, pages 368–377, 1991.
5. D. Fischer, E. Grädel, and Ł. Kaiser. Model checking games for the quantitative μ-calculus. *Theory Comput. Syst.*, 47(3):696–719, 2010.
6. D. Fischer and Ł. Kaiser. Model checking the quantitative μ-calculus on linear hybrid systems. In *Proc. of ICALP '11 (2)*, volume 6756 of *LNCS*, pages 404–415. Springer, 2011.
7. T. Ganzow and Ł. Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In *Proc. of CSL '10*, volume 6247 of *LNCS*, pages 366–380. Springer, 2010.
8. Ł. Kaiser and S. Leßenich. Counting μ-calculus on structured transition systems. 2012. Submitted.
9. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. of LICS '01*, pages 389–398. IEEE Computer Society Press, June 2001.
10. A. W. Mostowski. Games with forbidden positions. Technical Report 78, Instytut Matematyki, Uniwersytet Gdański, Poland, 1991.
11. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1–2):69–107, 1995.
12. J. H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Sciences*, 29(2):274–301, 1984.
13. W. Thomas. Infinite games and verification. In *Proc. of CAV '02,*, volume 2404 of *LNCS*, pages 58–64. Springer, 2002.
14. H. Wan. Upper bounds for Ramsey numbers R(3, 3,..., 3) and Schur numbers. *Journal of Graph Theory*, 26(3):119–122, 1997.

A Alternative proof of Theorem 4

Theorem 4 (Finite-memory winning strategies in second-life games). *Let $\mathcal{G}$ be an arena and W a regular winning condition. If Player 0 has a winning strategy in the second-life game $\mathcal{S}(\mathcal{G}, W)$, then he also has a finite-memory one.*

Proof. Notice that when considering the set of all possible plays in a game with perfect information, strategies of one player correspond to labelings in the tree, where every position of this player is labeled, and has a single labeled child (unless it is a terminal). For games with imperfect information, this is more involved, as not the tree of all plays, but the tree of all observations of plays needs to be considered. For imperfect recall, we also have to add that subpaths might be forgotten.

To obtain the right tree structure for strategies of Player 0 in $\mathcal{S}(\mathcal{G}, W)$, consider the tree t of all plays in $\mathcal{G}$, that is, the unfolding of $\mathcal{G}$ from a given initial vertex. To simplify arguments, we assume that there is a monadic relation P_v for every vertex, with $P_v^{\mathcal{G}} = \{v\}$. We extend t to a graph t^u by adding edges:

(i) for every position $\pi = v_0 \cdots v_n$ with $v_n \in V_1$ in the tree, add an $\mathsf{Call}(a)$-labeled edge from π to every a-successor.
(ii) from every $\pi = v_0 \cdots v_n$ such that v_n is a terminal vertex, add a Return-labeled edge to every $\pi' < \pi$ such that $\pi' = \cdots v_{k-1} a_k v_k$ with $v_{k-1} \in V_1$.

Then, every strategy of Player 0 in $\mathcal{S}(\mathcal{G}, W)$ that respects the imperfect information and recall-constraint corresponds to a labeling representing a strategy in this tree, and vice-versa (where strategies respect certain consistency constraints, see below). Notice that, in t^u, there are multiple edges connecting some vertices (imperfect information), and furthermore, the Return-edges are backward ones (imperfect recall), thus t^u is not a tree anymore. However, t^u is MSO-interpretable in t, as (i) and (ii) above are expressed in MSO-definable terms.

Consider now the unfolding $U(t^u)$ of t^u. In $U(t^u)$, both the multiple edge- and the backward edge-problems are resolved, and $U(t^u)$ contains the tree of all strategies in the perfect information variant of $\mathcal{S}(\mathcal{G}, W)$. Again, strategies are labelings of the trees. Because of the definition of the arena, strategies of Player 1 are not arbitrary labelings of his vertices and successors, but have to respect the Call and Return definitions: whenever a Call is marked, then the next Returns (which will occur because all terminals belong to Player 1) have to be ones to the target vertex of the Call. Furthermore, no Return can occur without a previous Call. This is a regular constraint, as it is expressed in MSO-definable terms. The set of labeled $U(t^u)$-trees corresponding to unconstrained strategies of Player 0 in the perfect information variant is regular, as Player 0 can play neither Call nor Return actions. Thus, there exist MSO sentences φ_0 and φ_1, such that a labeled $U(t^u)$-tree satisfies φ_i iff the labeling corresponds to a strategy of Player i (respecting the above constraints).

As the winning condition W is regular, there exists an MSO sentence ψ satisfied by exactly those labeled $U(t^u)$-trees where the labeling encodes strategies of both Player 0 and Player 1, i.e., the tree satisfies both φ_0 and φ_1, and the unique

play defined by the two strategies, i.e., the path following the labeled nodes, is in the set W. Similarly, there exists a formula ψ_0 accepting a tree iff the labeling corresponds to an unconstrained winning strategy of Player 0 (using universal quantification over φ_0 in ψ). Thus, in the perfect information and recall variant, the unconstrained winning strategies of Player 0 are regular.

By [15], every regular property over the unfolding has a regular pre-image. Thus, there exists an MSO-formula ψ_0', which holds in a labeled graph t^u, if, and only if, the labeling in the unfolding encodes a winning strategy of Player 0. By definition of t^u, these labelings (and also all labelings which satisfy the pre-image formula φ_0) comply with the imperfect information and recall constraint, and thus are valid strategies of Player 0 for $\mathcal{S}(\mathcal{G}, W)$. From this follows that the set of winning strategies of Player 0 is regular, and since t^u is MSO interpretable in t, there exists a regular labeled t-tree encoding a winning strategy of Player 0 for $\mathcal{S}(\mathcal{G}, W)$. But then, Player 0 has a finite-memory winning strategy. □

Note that, in the above proof, if Player 0 has a winning strategy in the perfect information variant, the set of $U(t^u)$-models of ψ_0 is non-empty. However, this does not mean that Player 0 also has a winning strategy in $\mathcal{S}(\mathcal{G}, W)$, as the pre-image of these models can still be empty.

Appendix References

15. B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph converings and unfoldings of transition systems. *Ann. of Pure and Applied Logic*, 92(1):35–62, 1998.