A Simple Algorithm for Solving Qualitative Probabilistic Parity Games

Ernst Moritz Hahn[1], Sven Schewe[2], Andrea Turrini[1], and Lijun Zhang[1]

[1] State Key Laboratory of Computer Science, Institute of Software, CAS, China
[2] University Of Liverpool, UK

Abstract. In this paper, we develop an approach to find strategies that guarantee a property in systems that contain controllable, uncontrollable, and random vertices, resulting in probabilistic games. Such games are a reasonable abstraction of systems that comprise partial control over the system (reflected by controllable transitions), hostile nondeterminism (abstraction of the unknown, such as the behaviour of an attacker or a potentially hostile environment), and probabilistic transitions for the abstraction of unknown behaviour neutral to our goals. We exploit a simple and only mildly adjusted algorithm from the analysis of non-probabilistic systems, and use it to show that the qualitative analysis of probabilistic games inherits the much celebrated sub-exponential complexity from 2-player games. The simple structure of the exploited algorithm allows us to offer tool support for finding the desired strategy, if it exists, for the given systems and properties. Our experimental evaluation shows that our technique is powerful enough to construct simple strategies that guarantee the specified probabilistic temporal properties.

1 Introduction

The automated synthesis of reactive protocols (strategies, policies) from temporal specifications has recently attracted considerable attention in numerous applications. Such a scenario can present both nondeterministic and probabilistic behaviours, so the resulting model can be seen as a Markov Decision Process (MDP) [31]. MDPs can also be viewed as $1\frac{1}{2}$-*player games*, where the full player decides which action to perform when resolving nondeterministic choices, and the $\frac{1}{2}$-player (or: random player) resolves the probabilistic choices.

This game can be enriched with a second player, e.g. an environment player, who controls some of the nondeterministic choices. Usually, the second player acts in a hostile manner: he tries to prevent the first (full) player from reaching her goal. The resulting game is known as a $2\frac{1}{2}$-*player game*. Examples of $2\frac{1}{2}$-player games are security and communication protocols [24, 28, 29, 34, 35] and robots playing in pursuit-evasion games [20].

A particular application in software engineering is the development of probabilistic reactive protocols and interfaces for probabilistic components. Such an interface would restrict the interactions a component offers to its environment. This technically corresponds to choosing a strategy for the component in a game

with its environment, where the goal of the component is to satisfy its specification, while the goal of the environment is to violate it.

The contribution of this paper is to provide an efficient algorithm to synthesise strategies for the controllable player in $2\frac{1}{2}$-player games that are equipped with a *parity* winning condition. Parity conditions are very general winning conditions that can be used to represent all ω-regular properties. In particular, parity objectives contain temporal logics such as LTL (linear temporal logic [30]), which are useful in specifying system and program properties. Because of this, we can handle LTL with a probabilistic semantics, i.e. the qualitative fragment of PLTL [1], in our synthesis framework.

We focus on computing the regions of the game, in which one of the players is almost sure winning [9]. The algorithm from [9] is based on translating the stochastic parity game into a simple parity game [8] (a parity game without random vertices / a 2-player game) using a so called "gadget" construction, albeit to the cost of a blow-up by a factor linear in the number of priorities. For a small number of priorities, or colours, the complexity of this algorithm is better than the algorithm we suggest here, because solving the blown-up simple parity game with [21] or [32] provides better bounds. This advantage is, however, purely theoretical: even for non-stochastic parity games on which these algorithms can be applied directly without requiring a costly transformation, they do not perform well in practice [14], such that a nested fixed-point algorithm [13, 27] would be the natural choice for analysing the blown-up game.

Our algorithm is an adaptation of the classical nested fixed-point algorithm of McNaughton [13,27] for the analysis of non-probabilistic systems. In particular, it does not involve a translation of the game. Thus, we avoid the practical problems existing algorithms with good complexity have [14]. The simple structure of the exploited algorithm also allows us to offer tool support by implementing a protocol synthesiser.

Present algorithms with the best theoretical complexity bounds for solving 2-player parity games with a low or fixed number of priorities are described in [21, 32]. However, the ranking based approach from [21] does not perform very well in practice [14], and the hybrid approach from [32] will inherit these practical draw-backs.

The direct algorithm we describe has exponential worst-case complexity. In the paper, we exploit the simple structure of our algorithm to lift the subexponential algorithm for 2-players games of Jurdiński, Paterson, and Zwick [22] to the qualitative analysis of $2\frac{1}{2}$-player games.

Related Work. There is a rich body of literature of algorithms for parity games in a two player setting [13, 21, 22, 27, 32, 36], and a few for multi player games with concurrent moves [3–5], in which players make simultaneous choices in every move. All of these algorithms share an $n^{\mathcal{O}(n)}$ running time.

Some experiments have suggested that the algorithm proposed in [13, 27, 36] performs best among them, in particular because it can be implemented symbolically. In this paper, we are considering non-concurrent games. We have adjusted McNaughton's algorithm [13, 27, 36] for solving these $2\frac{1}{2}$-player games.

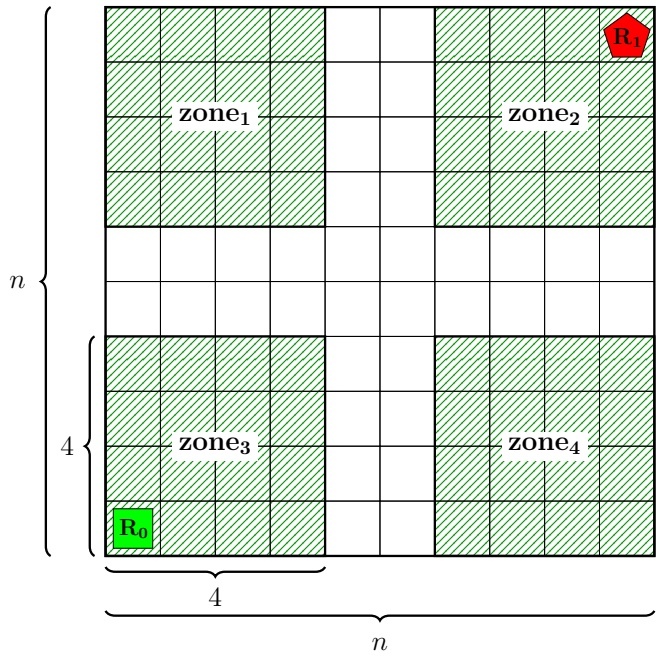

Fig. 1. Robot arena.

Such games have also been considered in the literature, for instance in [2, 6, 9, 15, 16]. Like 2-player games, they have pure memoryless optimal strategies.

Algorithms for solving probabilistic games have been implemented among others in PRISM-games [11]. This tool can, however, only handle a restricted class of properties, namely PCTL [19], which does not support nested temporal operators. As mentioned above, our approach can handle properties specified in the logics PLTL [1], as it translates such properties to parity objectives.

2 A motivating example

In a two-dimensional battlefield (cf. Figure 1), there are two robots, R_0 and R_1, and four marked zones, $\text{zone}_1, \ldots, \text{zone}_4$. The battlefield is surrounded by a solid wall and has a square tiled floor of n fields in breadth and width. Each tile can be occupied by at most one robot. The robots act in strict alternation. When it is the turn of a robot, this robot can move as follows: decide a direction and move one field forward; decide a direction and attempt to move two fields forward. In the latter case, the robot moves two fields forward with a probability of 50%, but only one field forward with a probability of 50%. If the robot would run into a wall or into the other robot, it stops at the field before the obstacle.

We assume that we are in control of R_0 but cannot control the behaviour of R_1. Our goal is to fulfil a certain objective depending on the zones with

probability 1, such as repeatedly visiting all zones infinitely often, visiting the zones in a specific order, performing such visits without entering other zones in the meanwhile, and so on. We have to ensure that these objectives are fulfilled under any possible behaviour of R_1. As an example, we can specify that the robot eventually reaches zone_1 and zone_2, which can be specified using the LTL formula $\mathbf{F}\,\text{zone}_1 \wedge \mathbf{F}\,\text{zone}_2$.

3 Preliminaries

3.1 Markov parity games

We now introduce the formal definition of the Markov parity games that model the $2\frac{1}{2}$-player games together with the other concepts and notations we use in the remainder of the paper.

Definition 1. *A* finite Markov Parity Game*, MPG for short, is a tuple* $\mathcal{P} = (V_0, V_1, V_r, E, \mathsf{pri})$*, where*

- $V = V_0 \cup V_1 \cup V_r$ *is the set of* vertices*, where* V_0*,* V_1*, and* V_r *are three finite disjoint sets of vertices owned by the three players: Player* 0*, Player* 1*, and Player* random*, respectively;*
- $E \subseteq V \times V$ *is a set of* edges *such that* (V, E) *is a* sinkless *directed graph, i.e. for each* $v \in V$ *there exists* $v' \in V$ *such that* $(v, v') \in E$*; and*
- $\mathsf{pri}\colon V \to \mathbb{N}$ *is the priority function mapping each vertex to a natural number. We call the image of* pri *the set of* priorities *(or:* colours*), denoted by* $\mathcal{C}$*.*

Note that, since the set of vertices V is finite, $\mathcal{C}$ is finite as well. For an MPG $\mathcal{P} = (V_0, V_1, V_r, E, \mathsf{pri})$, we call the tuple $\mathfrak{A} = (V_0, V_1, V_r, E)$ the *arena* of $\mathcal{P}$. For ease of notation, we sometimes use games when we refer to their arenas only. We also use the common intersection and subtraction operations on directed graphs for arenas and games: given an MPG $\mathcal{P}$ with arena $\mathfrak{A} = (V_0, V_1, V_r, E)$,

- $\mathcal{P} \cap V'$ denotes the Markov parity game $\mathcal{P}'$ we obtain when we restrict the arena $\mathfrak{A}$ to $\mathfrak{A} \cap V' = (V_0 \cap V', V_1 \cap V', V_r \cap V', E \cap (V' \times V'))$.
- $\mathcal{P} \setminus V'$ denotes the Markov parity game $\mathcal{P}'$ with arena $\mathfrak{A} \setminus V' = \mathfrak{A} \cap (V \setminus V')$, where $V = V_0 \cup V_1 \cup V_r$.

Note that the result of such an intersection may or may not be sinkless. While we use these operations freely in intermediate constructions, we make sure that, wherever they are treated as games, they have no sinks (cf. Lemma 4).

Plays. One can view the dynamics of a parity game as a board game, played by moving a pebble over the game arena. When the pebble is on a vertex v, the next vertex v' is chosen by the player owning the vertex v, that is, by Player 0 if $v \in V_0$, by Player 1 if $v \in V_1$, and by Player random if $v \in V_r$. The respective player chooses a proper successor v' of v, i.e. a vertex v' with $(v, v') \in E$, and pushes the pebble forward to v'. This way, they together construct an infinite

play. If $V_0 = \emptyset$ or $V_1 = \emptyset$, we arrive at the model of *Markov decision processes* (MDPs).

A *play* is an infinite sequence $\pi = v_0 v_1 v_2 v_3 \ldots$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. Each play is evaluated by the lowest priority that occurs infinitely often. Player 0 wins a play $\pi = v_0 v_1 v_2 v_3 \ldots$ if the lowest priority that occurs infinitely often in the sequence $\mathsf{pri}(v_0)\mathsf{pri}(v_1)\mathsf{pri}(v_2)\mathsf{pri}(v_3)\ldots$, $\mathsf{pri}(\pi) = \liminf_{i\to\infty} \mathsf{pri}(v_i)$ is even, while Player 1 wins if $\mathsf{pri}(\pi)$ is odd. Below, we formalise *winning strategies* and *winning regions* for a given MPG:

Definition 2. *For a given MPG $\mathcal{P}$, we say*

- *For $\sigma \in \{0,1\}$, a strategy f_σ of Player σ is a mapping $f_\sigma \colon V_\sigma \to V$ from the vertices of Player σ to their successor states, i.e. for each $v \in V_\sigma$, $(v, f_\sigma(v)) \in E$. A play $\pi = v_0 v_1 v_2 v_3 \ldots$ is called f_σ-conform if, for all $i \in \mathbb{N}$, if $v_i \in V_\sigma$, then $v_{i+1} = f_\sigma(v_i)$. A strategy f_σ for the player σ defines an MDP, namely the MDP where each vertex $v \in V_\sigma$ has exactly one successor, $f_\sigma(v)$.*
- *Given a vertex $v \in V$, a strategy f_0 for Player 0 is called* v-winning *if, starting from v, Player 0 wins almost surely in the MDP defined by f_0; a strategy f_1 for Player 1 is called* v-winning *if, starting from v, Player 1 wins with non-zero probability in the MDP defined by f_1, that is, Player 1 does not lose almost surely in the MDP defined by f_1.*
- *For $\sigma \in \{0,1\}$, a vertex v in V is v-winning for Player σ if Player σ has a v-winning strategy f_σ. We call the set of v-winning vertices for Player σ the* winning region *of Player σ, denoted W_σ.*

As we concentrate on finding almost surely winning/losing regions, it suffices to assume that there is an $\varepsilon > 0$, such that all choices of Player random are made with probability of at least ε. We therefore omit the probabilities in our model.

4 McNaughton's algorithm and memoryless determinacy

In this section, we adjust the classic algorithm for solving 2-player parity games to Markov parity games. The classic algorithm dates back to McNaughton [27] and was first published in this form by Emerson and Lei [13] and Zielonka [36].

The algorithm is the algorithmic version of a simple proof of the memoryless determinacy for parity games. The proof uses an inductive argument over the number of vertices. As an induction basis, games with only one game vertex are clearly memoryless determined: there is only one strategy, and it is memoryless. The game is won by Player 0 if the priority of this vertex is even and by Player 1 if the priority of this vertex is odd. We provide an algorithm for determining where Player 0 wins almost surely, i.e. its winning region W_0. Similarly, we compute the winning region W_1, in which Player 1 wins with non-zero probability. Adjusting it to finding sure winning sets for Player 1 is straight forward.

Lemma 1. *If, for a game $(V_0, V_1, V_r, E, \mathsf{pri})$, the image of* pri *consists only of even priorities, then Player 0 wins (surely) from all vertices, and $W_0 = V$.*

Procedure *prob−McNaughton*($\mathcal{P}$): ($\mathcal{P} = (V_0, V_1, V_r, E, \mathsf{pri})$)

1. if $V = \emptyset$ then return $(\emptyset, \emptyset)$ (note that $V = V_0 \cup V_1 \cup V_r$)
2. set c to the minimal priority occurring in $\mathcal{P}$
3. if the image of pri is even then return $(V, \emptyset)$
4. if the image of pri is odd then return $(\emptyset, V)$
5. if c is even then
   (a) set W_1 to $\emptyset$
   (b) repeat
      i. set $\mathcal{P}'$ to $\mathcal{P} \setminus \mathsf{satr}_0(\mathsf{pri}^{-1}(c), \mathcal{P})$
      ii. set (W_0', W_1') to *prob−McNaughton*($\mathcal{P}'$)
      iii. if $W_1' = \emptyset$ then
         A. set W_0 to $V \setminus W_1$
         B. return (W_0, W_1)
      iv. set W_1 to $W_1 \cup \mathsf{satr}_1(W_1', \mathcal{P})$
      v. set $\mathcal{P}$ to $\mathcal{P} \setminus \mathsf{satr}_1(W_1', \mathcal{P})$
6. set W_0 to $\emptyset$
7. repeat
   (a) set $\mathcal{P}'$ to $\mathcal{P} \setminus \mathsf{satr}_1(\mathsf{pri}^{-1}(c), \mathcal{P})$
   (b) set (W_0', W_1') to *prob−McNaughton*($\mathcal{P}'$)
   (c) if $W_0' = \emptyset$ then
      i. set W_1 to $V \setminus W_0$
      ii. return (W_0, W_1)
   (d) set W_0 to $W_0 \cup \mathsf{watr}_0(W_0', \mathcal{P})$
   (e) set $\mathcal{P}$ to $\mathcal{P} \setminus \mathsf{watr}_0(W_0', \mathcal{P})$
   adding a sink vertex reachable from the random vertices adjacent to W_0 (cf. Lemma 7); this sink is a technicality and does not count when we check $W_0' = \emptyset$ in line 7.c

Fig. 2. The algorithm *prob−McNaughton*($\mathcal{P}$) returns the ordered pair (W_0, W_1) of winning regions of Player 0 and Player 1, respectively. V and pri denote the states and the priority function of the parity game $\mathcal{P}$.

If, for a game $(V_0, V_1, V_r, E, \mathsf{pri})$*, the image of* pri *consists only of odd priorities, then Player* 1 *wins (surely) from all vertices, and* $W_1 = V$*.*

For general parity games $\mathcal{P}$ with lowest priority c, our adjustment of the McNaughton's algorithm, the procedure '*prob−McNaughton*' shown in Figure 2, first determines the set $\mathsf{pri}^{-1}(c)$ of vertices with priority c, i.e. the vertices with minimal priority. If c is even, then Player 0 wins on all plays, where c occurs infinitely often. The algorithm then constructs the region, from which Player 1 cannot almost surely avoid to reach a vertex with priority c. This is obtained using attractors.

4.1 Attractors

Attractors are usually defined for classical 2-player games. For an arena $\mathfrak{A} = (V_0, V_1, E)$, a set $T \subseteq V$ of target vertices, and a player $\sigma \in \{0, 1\}$, the σ-

attractor of T is the set of game vertices from which Player σ can force the pebble into the set T of target vertices. The σ-attractor A of a set T can be defined as the least fixed point of sets that contain T and that contain a vertex v of Player σ if it contains some successor of v and a vertex v of Player $1-\sigma$ if it contains all the successors of v. Equivalently, the σ-attractor $\sigma{-}Attractor(T, \mathfrak{A})$ of T in the arena $\mathfrak{A}$ can be defined as $A = \bigcup_{j\in\mathbb{N}} A_j$ where

$$\begin{aligned} A_0 &= T, \\ A_{j+1} &= A_j \cup \{\, v \in V_\sigma \mid \exists v' \in A_j.\, (v, v') \in E \,\} \\ &\qquad \cup \{\, v \in V_{1-\sigma} \mid \forall (v, v') \in E.\, v' \in A_j \,\}. \end{aligned}$$

This definition also provides a memoryless strategy for Player σ to move the pebble to T from all vertices in A: for a vertex $v \in A$, there is a minimal $i \in \mathbb{N}$ such that $v \in A_i$. For $i > 0$ (i.e. for $v \notin T$) and $v \in V_\sigma$, v has a successor in A_{i-1}, and Player σ can simply choose such a successor. (For $v \in V_{1-\sigma}$, all successors are in A_{i-1}.) Likewise, A itself provides a memoryless strategy to keep the pebble out of A (and hence out of T) for Player $1-\sigma$ when starting from a vertex $v \notin A$, namely to never enter A.

For $2\frac{1}{2}$-player games, we distinguish two different types of attractors: strong attractors, where the random player co-operates with the player σ who wants to push the pebble into the target set (denoted $\mathsf{satr}_\sigma(T, \mathcal{P})$ when Player σ tries to move the pebble to T in the arena of $\mathcal{P}$); and weak attractors, where the target needs to be reached almost surely. (The 'strong' and 'weak' in their names are inspired by $\mathsf{satr}_\sigma(T, \mathcal{P}) \supseteq \mathsf{watr}_\sigma(T, \mathcal{P})$, which makes the first attractor stronger.) Note that the principle of the attractor construction is not affected by the co-operation for the strong attractor: we simply treat the random vertices as vertices of player σ and apply the normal attractor construction from 2-player games.

Lemma 2. *For an arena* $\mathfrak{A}$ *and a set* T *of target states, the strong* σ*-attractor of* T *can be constructed in time linear in the edges of* $\mathfrak{A}$*.*

The construction of weak attractors is more complex since it requires solving a singly nested fixed-point. Consider an MPG $\mathcal{P}$, a set $T \subseteq V$, and the player σ; the weak σ-attractor $\mathsf{watr}_\sigma(T, \mathcal{P})$ of the set T is defined as follows:

$$\begin{aligned} S_0 &= \mathsf{satr}_\sigma(T, \mathcal{P}), \\ C_j &= \mathsf{satr}_{1-\sigma}(V \setminus S_j, \mathcal{P} \setminus T), \\ S_{j+1} &= \mathsf{satr}_\sigma(T, \mathcal{P} \setminus C_j), \\ \mathsf{watr}_\sigma(T, \mathcal{P}) &= \bigcap_{j\in\mathbb{N}} S_j. \end{aligned}$$

As a singly nested fixed point, constructing weak attractors can be reduced to solving Büchi games, which can be done in $O(n^2)$ time [7]. (We have implemented the classic $O(m \cdot n)$ iterated fixed point algorithm.)

Lemma 3. *For an arena* $\mathfrak{A}$ *and a set* T *of target states, the weak* σ*-attractor of* T *can be constructed in time quadratic in the states of* $\mathfrak{A}$*.*

Note that the co-games of attractors are proper games: they have no sinks.

Lemma 4. *For an arena $\mathfrak{A}$ and a set T of target states, $\mathfrak{A}' = \mathfrak{A} \setminus \mathsf{satr}_\sigma(T, \mathfrak{A})$ and $\mathfrak{A}'' = \mathfrak{A} \setminus \mathsf{watr}_\sigma(T, \mathfrak{A})$ are arenas for $\sigma \in \{0, 1\}$.*

For strong attractors, this is because every vertex in $(V_\sigma \cup V_r) \setminus \mathsf{satr}_\sigma(T, \mathfrak{A})$ has the same successors in $\mathfrak{A}'$ and $\mathfrak{A}$, while every vertex in $V_{1-\sigma} \setminus \mathsf{satr}_\sigma(T, \mathfrak{A})$ has some successor in $\mathfrak{A}'$. For weak attractors, every vertex in $V_\sigma \setminus \mathsf{watr}_\sigma(T, \mathfrak{A})$ has the same successors in $\mathfrak{A}''$ and $\mathfrak{A}$, while every vertex in $(V_{1-\sigma} \cup V_r) \setminus \mathsf{watr}_\sigma(T, \mathfrak{A})$ has some successor in $\mathfrak{A}'$.

4.2 Traps and paradises

After constructing $A = \mathsf{satr}_{c_{\min} \mod 2}(\mathsf{pri}^{-1}(c_{\min}), \mathcal{P})$, the co-game $\mathcal{P}' = \mathcal{P} \setminus A$ of $\mathcal{P}$ is solved.

A *σ-trap* $T_\sigma \subseteq V$ is a set of vertices Player σ cannot force to leave, not even with some probability greater than 0 (she is trapped there). A set T_σ is a σ-trap if it is the co-set of a strong σ-attractor, i.e. if it satisfies $V \setminus T_\sigma = \mathsf{satr}_\sigma(V \setminus T_\sigma, \mathcal{P})$.

The co-game $\mathcal{P}'$ is smaller than $\mathcal{P}$: compared to $\mathcal{P}$, it has less vertices. By induction hypothesis, it is therefore memoryless determined. By induction over the size of the game, $\mathcal{P}'$ can therefore be solved by a recursive call of the algorithm.

The following picture shows how the first part of the algorithm works. We first determine the minimal priority in the game (line 2 of *prob−McNaughton*). Let $c := c_{\min}$ denote the minimal priority, and select the target to be $T = \mathsf{pri}^{-1}(c)$, shown as the solid green area in the top right corner. We then construct the respective strong attractor $A = \mathsf{satr}_\sigma(T, \mathcal{P})$ with $\sigma = c \mod 2$, shown as T plus the NE hatched part around T, and consider the co-game $\mathcal{P}'$ (lines 5.b.i and 7.a of *prob−McNaughton*).

We then solve $\mathcal{P}'$, resulting in the winning regions $W'_{1-\sigma}$, shown in solid red, and W'_σ, the green SW hatched part below (lines 5.b.ii and 7.b of the procedure *prob−McNaughton*).

Fig. 3. First part of the algorithm; c is the minimal priority and $\sigma = c \mod 2$

We fix the target set T. We call a subset $P_0 \subseteq W_0$ of the winning region of Player 0 a 0-*paradise* if it is a 1-trap and Player 0 has a memoryless strategy f, which is v-winning for all $v \in P_0$ in the game $\mathcal{P} \cap P_0$, such that P_0 cannot be left in any f-conform play.

We call a subset $P_1 \subseteq W_1$ of the winning region of Player 1 a 1-*paradise* if Player 1 has a memoryless strategy f, which is v-winning for all vertices $v \in P_1$, such that the probability measure of the plays π with odd $\mathsf{pri}(\pi)$ that never leave P_1 is non-zero for all vertices in P_1. (Note that it suffices for this property to define f on P_1.) In particular, all vertices in $v \in P_1$ are v-winning for Player 1.

Returning to the picture, $W'_{1-\sigma}$—the solid red area—is a $(1-\sigma)$-paradise for the game $\mathcal{P}$.

Lemma 5. *For a parity game $\mathcal{P}$ with a σ-trap T_σ and a $(1-\sigma)$-paradise $W_{1-\sigma}$ of $\mathcal{P}' = \mathcal{P} \cap T_\sigma$, $W_{1-\sigma}$ is a $(1-\sigma)$-paradise for $\mathcal{P}$.*

The lemma is obvious: Player $1-\sigma$ can simply use the same winning strategy f for $\mathcal{P}$ as for $\mathcal{P}'$ on T_σ: as T_σ is a σ-trap, neither Player σ nor Player random have additional moves in $\mathcal{P}$, and every f-conform play that starts in $W_{1-\sigma}$ in $\mathcal{P}$ is also an f-conform play in $\mathcal{P}'$. The winning region $W'_{1-\sigma}$ of $\mathcal{P}'$ is therefore a $(1-\sigma)$-paradise in $\mathcal{P}$.

This property can be easily extended to paradises using attractor operations (strong attractor for player odd and weak attractor for player even):

Lemma 6. *The strong 1-attractor $A = \mathsf{satr}_1(P_1, \mathcal{P})$ of a 1-paradise P_1 for a parity game $\mathcal{P}$ is a 1-paradise for $\mathcal{P}$, and the weak 0-attractor $A = \mathsf{watr}_0(P_0, \mathcal{P})$ of a 0-paradise P_0 for a parity game $\mathcal{P}$ is a 0-paradise for $\mathcal{P}$. A winning strategy for the respective player σ on A can be composed of the winning strategy for Player σ on P_σ and an attractor strategy on $A \setminus P_\sigma$.*

As a result, for a given 0-paradise P_0 for Player 0 in a parity game $\mathcal{P}$, we can reduce solving $\mathcal{P}$ to computing the weak 0-attractor $A = \mathsf{watr}_0(P_0, \mathcal{P})$ of P_0, and solving $\mathcal{P} \setminus A$.

Lemma 7. *Let $\mathcal{P}$ be a parity game, P_0 be a 0-paradise with weak 0-attractor $A = \mathsf{watr}_0(P_0, \mathcal{P})$, and W'_0 and W'_1 be the winning regions of Player 0 and Player 1, respectively, on $\mathcal{P}'$, which is obtained from the game $\mathcal{P}'' = \mathcal{P} \setminus A$ by adding a random vertex s, a sink with a self-loop and even priority, which is reachable from the random vertices that have (in the parity game $\mathcal{P}$) a successor in A. Then*

- *$W_1 = W'_1$ is the winning region of Player 1 on $\mathcal{P}$, and he can win by following his winning strategy from $\mathcal{P}'$ on his winning region, and*
- *$W_0 = W'_0 \cup A \setminus \{s\}$ is the winning region of Player 0 and she can win by following her winning strategy for A on A and her winning strategy from $\mathcal{P}'$ on W'_0.*

Proof. Player 0 wins with her strategy from every vertex in A by a composition of the attractor strategy on $A \setminus P_0$ and her winning strategy on P_0 by Lemma 6.

Let g_0 be a winning strategy for Player 0 on W'_0 in $\mathcal{P}'$. Consider a probability distribution on $\mathcal{P}'$ that summarises the probabilities to transfer to A in each situation to the same probability to transfer to s, and a fixed counter strategy f of her opponent. As the likelihood of winning is 1 after transferring to A (with

the strategy from above by Lemma 6) in $\mathcal{P}$ and in $\mathcal{P}'$ (as all plays that reach s are winning), the chance of winning is equal in both cases. Vertices in $W_0' \setminus \{s\}$ are therefore winning for the composed strategy.

Similarly, let g_1 be a winning strategy for Player 1 on W_1' in $\mathcal{P}'$. We consider a probability distribution on $\mathcal{P}'$ that summarises the probabilities to transfer to A in each situation to the same probability to transfer to s, and a fixed counter strategy f of his opponent. Estimating the chance of winning to 0 after reaching A results in the same likelihood of winning in $\mathcal{P}$ and $\mathcal{P}'$. As this is larger than 0, Player 1 wins with an extension of the same strategy on his winning region. □

For our implementation, we have, instead of adding a sink with even priority, reduced the priority of all random vertices having s as successor to 0 (it could be set to any other even priority not greater than any priority of the vertices in $\mathcal{P} \setminus A$). This change is safe to implement: both players have the same memoryless strategies as in the game with the additional edges. Assume the players play according to fixed memoryless strategies f (Player 0) and g (Player 1), such that only stochastic decisions are left open. Player 0 will win almost surely from a vertex v if, and only if, for all leaf SCCs reachable from v the minimal priority among all priorities of the states in this leaf SCC is even. The leaf SCCs reachable from a vertex v other than the singleton leaf SCC $\{s\}$ from Lemma 7 are exactly those leaf SCCs reachable after re-prioritising, that do not contain a vertex whose priority is changed to 0. Thus, the same leaf SCCs with minimal odd priority are reachable. Consequently, the same vertices in A are winning for Player 0 (and for Player 1) in both constructions.

Corollary 1. *Let $\mathcal{P}$ be a parity game, P_0 be a 0-paradise with weak 0-attractor $A = \mathsf{watr}_0(P_0, \mathcal{P})$, and W_0' and W_1' be the winning regions of Player 0 and Player 1, respectively, on $\mathcal{P}'$, which is obtained from the game $\mathcal{P}'' = \mathcal{P} \setminus A$ by changing the priority of the random vertices that have (in the parity game $\mathcal{P}$) a successor in A to 0. Then*

- *$W_1 = W_1'$ is the winning region of Player 1 on $\mathcal{P}$, and he can win by following his winning strategy from $\mathcal{P}'$ on his winning region, and*
- *$W_0 = W_0' \cup A \setminus \{s\}$ is the winning region of Player 0 and she can win by following her winning strategy for A on A and her winning strategy from $\mathcal{P}'$ on W_0'.*

Note that this change of priority is, like adding the sink s, introduced recursively on a level in the call tree. While it is inherited in calls from there, the changes introduced in a level of the call tree (in line 7.e) are revoked when returning the values (in line 7.c.ii).

For a given 1-paradise P_1 for Player 1 in a parity game $\mathcal{P}$, we can reduce the qualitative analysis of a parity game $\mathcal{P}$ to computing the *strong* 1-attractor $A = \mathsf{satr}_1(P_1, \mathcal{P})$ of P_1, and solving $\mathcal{P} \setminus A$.

Lemma 8. *Let $\mathcal{P}$ be a parity game, P_1 be a 1-paradise with winning strategy f_1 and strong 1-attractor $A = \mathsf{satr}_1(P_1, \mathcal{P})$, and W_0' and W_1' be the winning regions of Player 0 and Player 1, respectively, on $\mathcal{P}' = \mathcal{P} \setminus A$. Then*

– $W_1 = W'_1 \cup A$ *is the winning region of Player* 1 *on* $\mathcal{P}$*, and he can win by following his winning strategy for* A *on* A *and his winning strategy* g_1 *from* $\mathcal{P}'$ *on* W'_1*, and*
– $W_0 = W'_0$ *is the winning region of Player* 0*, and she can win by following her winning strategy* g_0 *from* $\mathcal{P}'$ *on* W_0*.*

Proof. Player 0 wins with her winning strategy, g_0, on her complete winning region W_0 of $\mathcal{P} \setminus A$, since Player 1 has no additional choices in W_0 in $\mathcal{P}$. Consequently, the set of g_0-conform plays in $\mathcal{P}$ that start in W_0 coincides with the set of g_0-conform plays (with the same probability distribution on them) in $\mathcal{P} \setminus A$ that start in W_0.

Similarly, Player 1 wins with his strategy from every vertex in A, by a composition of his attractor strategy on $A \setminus P_1$ and his winning strategy f_1 on P_1 by Lemma 6.

Let g_1 be a winning strategy for Player 1 in $\mathcal{P}'$. Then every g_1-conform play in $\mathcal{P}$ that starts in a vertex in W'_1 either eventually reaches A, and is then almost surely followed by a tail (remainder of the play) in $\mathcal{P}$ that starts in A, which is winning for Player 1 with a likelihood strictly greater than 0 by Lemma 6; or it stays for ever in the sub-game $\mathcal{P}'$. But these plays are also won by Player 1 with a non-zero likelihood. □

We now distinguish two cases: firstly, if $W'_{1-\sigma}$ is non-empty, we can reduce solving $\mathcal{P}$ to constructing the weak or strong, respectively, $(1-\sigma)$-attractor $U_{1-\sigma}$ of $W'_{1-\sigma}$, and solving the co-game $\mathcal{P}'' = \mathcal{P} \setminus U_{1-\sigma}$ by Lemma 7 or 8, respectively. The co-game $\mathcal{P}''$ is simpler than $\mathcal{P}$: compared to $\mathcal{P}$, it contains less vertices (though not necessarily less priorities). By induction over the size of the game, $\mathcal{P}''$ can therefore be solved by a recursive call of the algorithm.

The figure below aligns this to our algorithm. We have seen that $W'_{1-\sigma}$ from the previous picture, shown again in solid red, is a $(1-\sigma)$-paradise, and so is its (weak or strong) $(1-\sigma)$-attractor $U_{1-\sigma}$ (the complete red area, full and hatched). It is constructed in lines 5.b.iv and 7.d, respectively, of *prob−McNaughton*.

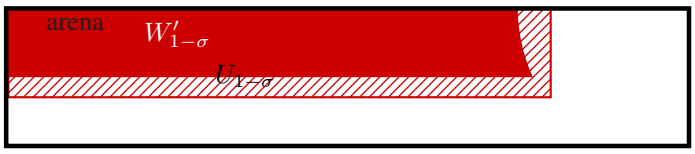

Fig. 4. Attractor of the player $1-\sigma$ where c is the minimal priority and $1-\sigma = c \mod 2$

Secondly, if $W'_{1-\sigma}$ is empty, we can compose the winning strategy for Player σ on $\mathcal{P}'$ with his attractor strategy for $\mathsf{pri}^{-1}(c)$ to a winning strategy on $\mathcal{P}$.

Lemma 9. *Let* $\mathcal{P}$ *be a parity game with minimal priority* c*,* $\sigma = c \mod 2$ *be the player who wins if* c *occurs infinitely often,* A *be the strong* σ*-attractor of* $\mathsf{pri}^{-1}(c)$*,*

and f be an attractor strategy for Player σ on her vertices on $A \setminus \mathsf{pri}^{-1}(c)$. If Player σ has a winning strategy f' for every vertex in $\mathcal{P}' = \mathcal{P} \setminus A$, then f and f' can be composed to a winning strategy for Player σ for every vertex in $\mathcal{P}$.

Proof. Let g be a strategy for Player σ that agrees with f and f' on their respective domain. We distinguish two types of g-conform plays: those that eventually stay in $\mathcal{P}'$, and those that visit A infinitely often. The latter plays almost surely contain infinitely many vertices with priority c and are therefore almost surely winning for Player σ. Games that eventually stay in $\mathcal{P}'$ consist of a finite prefix, followed by an f'-conform play in $\mathcal{P}'$. The lowest priority occurring infinitely often is therefore almost surely even for $\sigma = 0$ and odd with a likelihood strictly greater than 0 for $\sigma = 1$, respectively. □

Theorem 1. *For each parity game $\mathcal{P} = (V_0, V_1, V_r, E, \mathsf{pri})$, the game vertices are partitioned into a winning region W_0 of Player 0 and a winning region W_1 of Player 1. Moreover, Player 0 and Player 1 have memoryless strategies that are v-winning for every vertex v in their respective winning region.*

In the following proof, we do not count the sink that is added by Lemma 7.

Proof. Games with a single vertex are trivially won by the player winning on the priority of this vertex (induction basis).

For the induction step, assume that the memoryless determinacy holds for games with up to n vertices. For a parity game with $n+1$ vertices, we can then select the lowest priority $c_{\min}$, set σ to $c_{\min} \bmod 2$ to identify the Player σ who wins if $c_{\min}$ occurs infinitely often (note that $c_{\min}$ is the dominating priority in this case), and set $A = \mathsf{satr}_\sigma(\mathsf{pri}^{-1}(c_{\min}), \mathcal{P})$.

Then $\mathcal{P}' = \mathcal{P} \setminus A$ is a—possibly empty—parity game with strictly less states and priorities. (Note that, by the attractor construction, every vertex in $\mathcal{P}'$ has a successor, and the co-set of A is a σ-trap.)

By induction hypothesis, $\mathcal{P}'$ vertices are partitioned into winning regions of the two players, and both players have memoryless winning strategies on their winning regions.

We can now distinguish three cases:

1. The winning region of Player $1 - \sigma$ on $\mathcal{P}'$ is empty. In this case, Player σ wins memoryless by Lemma 9.
2. $\sigma = 0$ and the winning region of Player 1 is non-empty.
   Then $W_1'' = \mathsf{satr}_1(W_1', \mathcal{P})$ is a 1-paradise for $\mathcal{P}$ by Lemmas 5 and 6. We can therefore solve the remainder of the game, $\mathcal{P} \setminus W_1''$, individually and use the respective winning regions and (by induction) memoryless winning strategies of the players by Lemma 8.
3. $\sigma = 1$ and the winning region of Player 0 is non-empty.
   Then $W_0'' = \mathsf{watr}_0(W_1', \mathcal{P})$ is a 0-paradise for $\mathcal{P}$ by Lemmas 5 and 6. We can therefore solve the remainder of the game, $\mathcal{P} \setminus W_0''$, individually and use the respective winning regions and (by induction) memoryless winning strategies of the players by Lemma 7.

In case (1) we are done, in (2), (3) we reduced the problem to solving games with less states. By induction, memoryless determinacy extends to the complete game. □

The worst case running time of this extension to McNaughton's algorithm [13,27,36] (cf. Procedure *prob−McNaughton* of Figure 2) occurs if $U_{1-\sigma}$ is always small and exactly one vertex with minimal priority c belongs to $U_{1-\sigma}$. For parity games with c priorities, n vertices, and m edges, the adjusted algorithm based on McNaughton's requires $\mathcal{O}\big(m \cdot \big(\frac{n}{\mathsf{c}} + 1\big)^{\mathsf{c}-1}\big)$ steps when the highest priority of the game is even, like McNaughton's algorithm itself [13, 36]: the cost of the attractor constructions is always dominated by the cost of the recursive calls, and the complexity analysis is the same.

When the highest priority is odd, the higher complexity for the weak attractor construction leads to a $\mathcal{O}\big(\big(\frac{n}{\mathsf{c}}+1\big)^{\mathsf{c}+1}\big)$ complexity, and to an $\mathcal{O}\big(m \cdot \big(\frac{n}{\mathsf{c}}+1\big)^{\mathsf{c}}\big)$ complexity of our implementation, due to the cost of constructing weak attractors of $\mathcal{O}(n^2)$ and $\mathcal{O}(m \cdot n)$ in our implementation, respectively. (The only point where this complexity is not dominated by other steps is when only the highest and the second highest priority are left. When the second highest priority is even, only strong attractors are used.)

The presented algorithm has a theoretical drawback: its running time is exponential in the number of priorities, which in turn can be as high as the number of states. For parity games without random vertices, the development of a deterministic subexponential algorithm [22] was considered a breakthrough, which also led to better bounds for games with a small number of priorities [32].

Theorem 2. *Parity games with n vertices can be solved in time $n^{\mathcal{O}(\sqrt{n})}$.*

The proof is completely analogously to the one from [22]. It uses the fact that the proofs do not rely on the way intermediate paradises are constructed. It is therefore possible to try out all small sets of vertices (i.e. all sets of vertices up to size $\sqrt{n}$) and check if they are paradises for the player who loses on the minimal priority. This can be done using brute force and takes $n^{\mathcal{O}(\sqrt{n})}$ time. Afterwards, one builds their union, computes the attractor for the set obtained this way, and performs a recursive call to decide the remaining game.

The progress obtained in one step, consisting of trying out all small sets of vertices, followed by a recursive call, is therefore at least $\sqrt{n}$ – unless the call returns an empty set, in which case we are finished. This provides a call tree of size $n^{\mathcal{O}(\sqrt{n})}$, with cost $n^{\mathcal{O}(\sqrt{n})}$ on each node of the call tree, such that the total cost is $n^{\mathcal{O}(\sqrt{n})} \cdot n^{\mathcal{O}(\sqrt{n})} = n^{\mathcal{O}(\sqrt{n})}$.

5 Implementation and experimental results

5.1 Overview of the algorithm

We have written a prototypical implementation for the approach of this paper. In Figure 5, we sketch how we analyse properties of formal models. Our tool

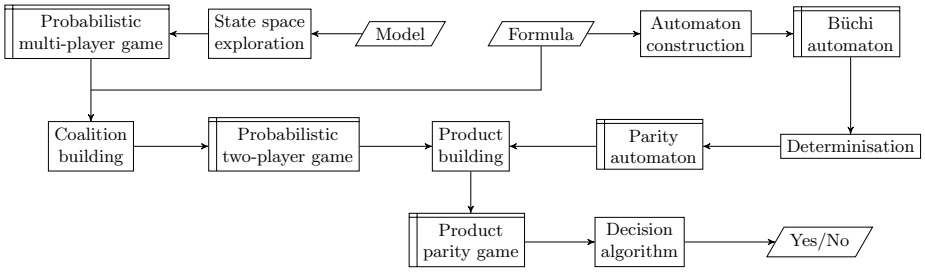

Fig. 5. Overview of the algorithm

reads a model specification in the input language of the probabilistic model checker PRISM-games [11]. This input language is an extension of probabilistic guarded-commands language of the probabilistic model checker PRISM [25]. We extend the recent probabilistic model checker IscasMC [17,18] that also supports this language. Each state of the model semantics is assigned to a specific player. When being in a state, a player can choose from several guarded commands. In each state, the player controlling this state can choose to execute one of the commands the guard of which is valid in the state. Afterwards, one of the possible effects of the command is chosen according to the probability of this effect. The next state is decided according to this effect. The models specified in this way may contain more than two players, and they do not contain any winning condition.

The property specification we use in our tool is also inspired by PRISM-games [11]. Its general form is $\langle\langle \text{coalition} \rangle\rangle \mathcal{P}_{\geq 1}[\phi]$. Here, coalition is a subset of the players from the model specification, and $\mathcal{P}_{\geq 1}[\phi]$ is a qualitative probabilistic LTL formula. This formula requires that the coalition of the players of coalition (by subsuming all of them into a single player) can enforce that ϕ holds with probability 1 under all possible behaviours of the players not in coalition. Variants like $\langle\langle \text{coalition} \rangle\rangle \mathcal{P}_{>0}[\phi]$ requiring only positive probability can be computed by minor adaptation of our algorithm.

To obtain the actual parity game to apply our algorithm on we proceed as follows. We first construct the explicit-state semantics of the model originally given in the high-level input language of PRISM-games. This semantics is thus a probabilistic multi-player game without a winning condition. Then, in this semantics we subsume the players of coalition to the single even player and the remaining players to the odd player, so as to obtain a probabilistic two-player game, still without a winning condition. We then transform ϕ into a generalised Büchi automaton using the tool SPOT [12]. Afterwards, we transform the Büchi into a deterministic parity automaton, using the algorithm from [33]. Simultaneously, we build a product of the model with this parity automaton, so as to fix the winning condition according to the property we want to decide. This on-the-fly approach allows us to only construct the states of the parity automaton which will actually be needed to construct the product. A state of

this product is thus a tuple of a state from the two-player model and a state of the parity automaton.

The algorithm from [33] produces automata whose priorities are on the transitions (rather than on states). The product we produce is however a state-labelled parity game as in Definition 1: when building the product of the model with the automaton, the current state fixes the successor transition of the deterministic parity automaton. Therefore, we can label the product state with the priority of this successor transition. The first component of the successor states consists of a successor state of the model state. The second one consists of the single successor state of the parity automaton.

Finally, we can apply the algorithm discussed on the resulting MPG and decide whether all initial states of the product are winning, which means deciding whether the specification holds for all initial states of the model. If it does, we also obtain a winning strategy.

As stated before, the mutually optimal strategies are memoryless. However, this only holds for the game we are computing strategies for. If we map back these strategies to the original PRISM-games model, indeed memory might be required. This is because we have to remember the current state of the parity automaton with which the model had been composed to obtain the product on which we finally apply our algorithm.

5.2 Robots

Reconsider our robot example introduced in Section 2. We have applied our tool to construct the state-space of several instantiations of this model which we have modelled in the input language of PRISM-games. The machine we used is a 3.6 GHz Intel Core i7-4790 with 16 GB 1600 MHz DDR3 RAM of which 12 GB assigned to the tool; the timeout has been set to 30 minutes. In Table 1 we provide for each size "n" of the battlefield we consider, the number of vertices of the corresponding multiplayer game ("vertices") and the time in seconds required to construct it ("t_{constr}").

Table 1. Arena construction

n	vertices	t_{constr}
12	370 656	1
16	1 175 040	3
20	2 872 800	6
24	5 961 600	13
28	11 049 696	25
32	18 855 936	47
36	30 209 760	69
40	46 051 200	112

We remark that the constructed game encodes the behaviour of the robots on the battlefield without winning conditions, i.e. without considering the formula to be checked; as seen, the state-space contains millions of vertices for larger battlefield sizes n (it grows with $\mathcal{O}(n^4)$). Note that we cannot compare to PRISM-games because it does not support general PLTL formulas, and we are not aware of other tools to compare with.

We have applied our tool on a number of properties that require the robot R_0 to visit the different zones in a certain order. In Table 2 we report the performance measurements for these properties. In the column "property" we state the PLTL formula we consider, column "n" states the width of the battlefield instance,

Table 2. Robots analysis: different reachability properties

property	n	sat	game construction			gadget construction		solving time		
			vertices	colours	t_{prod}	vertices	t_{gadg}	t_{gMcN}	t_{gJur}	t_{pMcN}
Reachability	12	true	1 324 704	2	2	5 351 584	3	0	4	0
	16	true	4 418 592	2	9	17 996 832	13	3	24	1
$\langle\langle R_0 \rangle\rangle \mathcal{P}_{\geq 1}$	20	true	11 050 656	2	22	45 166 752	28	8	84	3
[$\mathbf{F}$ zone$_1$	24	true	23 211 552	2	51	95 045 152	58	18	219	7
$\wedge \mathbf{F}$ zone$_2$]	28	true	43 334 304	2	102	177 634 464	115	35	–MO–	14
	32	true	74 294 304	2	197	-	–MO–	-	-	24
	36	-	-	-	–MO–	-	-	-	-	-
	40	-	-	-	–MO–	-	-	-	-	-
Repeated	12	true	1 324 704	2	2	6 010 528	3	1	5	0
Reachability	16	true	4 418 592	2	9	20 085 792	15	3	25	1
	20	true	11 050 656	2	20	50 273 952	29	9	85	3
$\langle\langle R_0 \rangle\rangle \mathcal{P}_{\geq 1}$	24	true	23 211 552	2	44	105 643 552	59	21	227	8
[$\mathbf{GF}$ zone$_1$	28	true	43 334 304	2	88	197 278 368	121	38	–MO–	15
$\wedge \mathbf{GF}$ zone$_2$	32	true	74 294 304	2	180	-	–MO–	-	-	27
$\wedge \mathbf{GF}$ zone$_3$	36	-	-	-	–MO–	-	-	-	-	-
$\wedge \mathbf{GF}$ zone$_4$]	40	-	-	-	–MO–	-	-	-	-	-
Repeated	12	true	693 048	5	0	4 009 976	2	0	3	0
Ordered	16	true	2 264 184	5	3	13 206 072	9	2	16	0
Reachability	20	true	5 611 320	5	6	32 844 792	17	6	50	1
	24	true	11 729 784	5	14	68 787 512	62	13	129	4
$\langle\langle R_0 \rangle\rangle \mathcal{P}_{\geq 1}$	28	true	21 836 088	5	27	128 198 136	69	24	282	7
[$\mathbf{GF}$(zone$_1 \wedge$	32	true	37 367 928	5	54	219 543 096	135	–MO–	–MO–	13
$\mathbf{F}$ zone$_2$)]	36	true	59 984 184	5	65	-	–MO–	-	-	21
	40	true	91 564 920	5	121	-	–MO–	-	-	36
Reach-avoid	12	true	1 616 400	4	2	7 656 720	4	1	–TO–	0
	16	true	5 452 848	4	14	25 820 208	15	6	–TO–	1
$\langle\langle R_0 \rangle\rangle \mathcal{P}_{\geq 1}$	20	true	13 703 184	4	35	64 877 328	33	19	–TO–	5
[$\neg$zone$_1$ $\mathbf{U}$ zone$_2$	24	true	28 855 728	4	79	136 606 128	116	40	–TO–	11
$\wedge \neg$zone$_4$ $\mathbf{U}$ zone$_2$	28	true	53 951 760	4	171	255 402 000	135	–MO–	–MO–	21
$\wedge \neg$zone$_4$ $\mathbf{U}$ zone$_1$	32	true	92 585 520	4	323	-	–MO–	-	-	38
$\wedge \mathbf{F}$zone$_4$]	36	-	-	-	–MO–	-	-	-	-	-
	40	-	-	-	–MO–	-	-	-	-	-

and "sat" shows whether the formula is satisfied. For the "game construction" part, we present the number of "vertices" of the resulting MPG, the number of "colours", and the time "t_{prod}" required to generate the MPG. The construction of [8] to turn a stochastic parity game into a non-stochastic parity game replaces each stochastic node by a so-called "gadget", which consists of a combination of player odd and even nodes. Applying this construction thus leads to a game which can be solved using existing methods, although at the cost of increasing the number of vertices and the time to perform the transformation. The "gadget construction" part of the table shows the total number of "vertices" after applying this construction and the time "t_{gadg}" spent to do so.

Finally, the "solving time" part shows the time spent by the specific solvers: t_{gMcN} and t_{gJur} correspond to the gadget construction solved by using the classical non-stochastic McNaughton and Jurdziński approach, respectively, while t_{pMcN} refers to our *prob−McNaughton* algorithm proposed in Figure 2. Note that these times represent only the actual time spent by the solver on the final MPG. Entries marked by "–TO–" and "–MO–" mean that the corresponding

phase has failed by time-out or has gone out of memory; entries marked by "-" mean that the phase has not been executed due to a failure in a previous phase.

The results show that our approach can be used to solve games with several million vertices, even though it is currently only implemented as an explicit-state solver. In particular, the part of the process that consumes the largest share of time is not the solution algorithm itself, but the preprocessing steps: most of the time was spent on constructing the product of the battlefield and the parity automaton. The largest amount of time spent in the solver was 38 seconds for a parity game with more than 90 million vertices. Despite the exponential complexity of the algorithm, our prototype performs quite well on the large state-spaces of this model. One reason is that the maximal number of different priorities seen was just 5, and the implementation was often able to use the lines 3 and 4 of the algorithm *prob−McNaughton* in Figure 2 to terminate the construction quickly. Indeed, we did not see more than 5 recursive calls of the algorithm. It is worthwhile to remark that, even if all properties are satisfied, not all vertices are winning for the robot R_0: for instance, for the reach-avoid property, around 1/3 of the vertices are winning for the robot R_1; R_0 is anyway able to avoid such vertices and win with probability 1.

In comparison, the solution methods based on the gadget construction have an additional phase that takes quite some time and memory to be completed, so this affects their performances as they have to work on much larger games. While the classical non-stochastic McNaughton algorithm has a reasonable performance on these games, it still consumes much more resources than our approach *prob−McNaughton*. Jurdziński's approach turns out to be really slow in practice, confirming the previous results of [14]. As the results in the table show, our approach really outperforms the methods based on gadget construction: for instance for the reach-avoid property for $n = 24$, our approach takes only 11 seconds instead of 40 (plus 79 for the gadget construction) taken by the classical McNaughton and the time-out of 30 minutes by the Jurdziński algorithm.

5.3 Two investors

As a further small case study, which provides similar results regarding the performance of the Jurdziński theoretical better algorithm, we consider an example originally from [26] in the version of [10]. In this version of the model, there are two investors, who are able to make investments in *futures*. Buying a future means to reserve an amount of company shares at a certain point of time and will be delivered at a later point of time to the market price the share has then. In this version of the model, there are three players: investor 1, investor 2, and the market. We considered a number of properties for which we provide results in Table 3; the meaning of the formulas is as follows:

1. Investor 1 is able to ensure (against the market and the other investor) that the share has eventually a value of at least 5 without ever stopping to invest.
2. She can ensure that the share repeatedly has a value of 5.
3. She can guarantee a permanent value of at least 5.

4. She can ensure that the probability of this event is non-zero.
5. If all players collaborate, this event is certain.

The parity game constructed to decide these properties contains less than 1.5 million vertices. Therefore, the time to decide the properties is also almost negligible: all experiments have taken a total of one or two seconds to complete, except for the Jurdziński approach that went time-out on all except one property.

Table 3. Two investors analysis.

	property	sat	game construction vertices	colours	t_{prod}	gadget construction vertices	t_{gadg}	solving time t_{gMcN}	t_{gJur}	t_{pMcN}
1.	$\langle\langle \text{investor}_1 \rangle\rangle \mathcal{P}_{\geq 1}[$ $\mathbf{G}\neg\text{done}_1 \wedge \mathbf{F} v \geq 5]$	true	410 531	4	0	1 001 645	1	0	–TO–	0
2.	$\langle\langle \text{investor}_1 \rangle\rangle \mathcal{P}_{\geq 1}[$ $\mathbf{G}\neg\text{done}_1 \wedge \mathbf{GF} v \geq 5]$	false	410 531	4	0	1 265 755	1	0	–TO–	0
3.	$\langle\langle \text{investor}_1 \rangle\rangle \mathcal{P}_{\geq 1}[$ $\mathbf{G}\neg\text{done}_1 \wedge \mathbf{FG} v \geq 5]$	false	413 171	5	0	1 486 783	1	0	–TO–	0
4.	$\langle\langle \text{investor}_1 \rangle\rangle \mathcal{P}_{>0}[$ $\mathbf{G}\neg\text{done}_1 \wedge \mathbf{FG} v \geq 5]$	true	413 171	5	0	1 486 783	1	0	0	0
5.	$\langle\langle \text{investor}_1, \text{investor}_2,$ $\text{market} \rangle\rangle \mathcal{P}_{\geq 1}[$ $\mathbf{G}\neg\text{done}_1 \wedge \mathbf{FG} v \geq 5]$	false	413 171	5	0	1 486 783	1	0	–TO–	0

6 Conclusions and Future Work

We have introduced a simple and effective algorithm for solving Markov games with a parity winning condition and implemented the algorithm as an explicit-state prototype as an extension of the model checker IscasMC [17, 18]. The algorithm has proven to be capable of handling rather large examples, obtaining strategies that almost surely obtain their goal or demonstrating that such strategies do not exist. This is a very encouraging result for the automated construction of simple probabilistic reactive protocols, as they are already used in leader election problems.

The construction of such protocols is already difficult for deterministic systems, and new protocols (as well as the old ones) had been discovered when they had been recently synthesised [23]. Some complicated programming problems like mutual exclusion, leader election, and variations thereof, have complete specifications. Yet, they are very difficult to implement, e.g. due to problems arising through context switches. While such problems have proven to be difficult to implement for human developers due to parallelism and nondeterminism in traditional systems, allowing for randomness does—while potentially simplifying the algorithm—add another layer of difficulty for the human developer. We believe that this establishes the need for synthesis techniques, and in this light it is a very good news that the solution we have developed in this paper shows po-

tential. In future work, we will extend this efficient technique to the quantitative analysis of systems.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grants No. 61472473, 61532019, 61550110249, 61550110506), by the National 973 Program (No. 2014CB340701), by the CDZ project CAP (GZ 1023), by the Chinese Academy of Sciences Fellowship for International Young Scientists, by by the CAS/SAFEA International Partnership Program for Creative Research Teams, and by the Engineering and Physical Sciences Research Council (EPSRC) through grant EP/M027287/1 (Energy Efficient Control).

References

1. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*, pages 499–513, 1995.
2. K. Chatterjee. *Stochastic ω-Regular Games*. PhD thesis, University of California at Berkeley, 2007.
3. K. Chatterjee. Qualitative concurrent parity games: Bounded rationality. In *CONCUR*, volume 8704 of *LNCS*, pages 544–559, 2014.
4. K. Chatterjee, L. de Alfaro, and T. A. Henzinger. Strategy improvement for concurrent reachability games. In *QEST*, pages 291–300. IEEE Computer Society, 2006.
5. K. Chatterjee, L. de Alfaro, and T. A. Henzinger. Qualitative concurrent parity games. *ACM Trans. Comput. Log.*, 12(4):28, 2011.
6. K. Chatterjee, L. Doyen, H. Gimbert, and Y. Oualhadj. Perfect-information stochastic mean-payoff parity games. In *FOSSACS*, volume 8412 of *LNCS*, pages 210–225, 2014.
7. K. Chatterjee and M. Henzinger. An $O(n^2)$ time algorithm for alternating Büchi games. In *SODA*, pages 1386–1399. SIAM, 2012.
8. K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Simple stochastic parity games. In *CSL*, volume 2803 of *LNCS*, pages 100–113, 2003.
9. K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Quantitative stochastic parity games. In *SODA '04*, pages 121–130, 2004.
10. T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.
11. T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *TACAS*, volume 7795 of *LNCS*, pages 185–191, 2013.
12. A. Duret-Lutz. LTL translation improvements in Spot. In *VECoS*, pages 72–83, 2011.
13. E. A. Emerson and C. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. LICS*, pages 267–278. IEEE Computer Society Press, 1986.
14. O. Friedmann and M. Lange. Solving parity games in practice. In *ATVA*, volume 5799 of *LNCS*, pages 182–196, 2009.

15. T. Gawlitza and H. Seidl. Games through nested fixpoints. In *CAV*, volume 5643 of *LNCS*, pages 291–305, 2009.
16. H. Gimbert and W. Zielonka. Perfect information stochastic priority games. In *ICALP*, volume 4596 of *LNCS*, pages 850–861, 2007.
17. E. M. Hahn, G. Li, S. Schewe, A. Turrini, and L. Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR*, volume 42 of *LIPIcs*, pages 354–367, 2015.
18. E. M. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang. IscasMC: A web-based probabilistic model checker. In *FM*, volume 8442 of *LNCS*, pages 312–317, 2014.
19. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *FAC*, 6(5):512–535, 1994.
20. J. P. Hespanha, H. J. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. In *CDC*, pages 2432–2437, 1999.
21. M. Jurdziński. Small progress measures for solving parity games. In *STACS*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.
22. M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.
23. G. Katz and D. Peled. Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. In *ATVA*, volume 5311 of *LNCS*, pages 33–47. Springer, 2008.
24. S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 11(3):399–430, 2003.
25. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
26. A. McIver and C. Morgan. Results on the quantitative mu-calculus qMu. *ACM Transactions on Computational Logic*, 8(1), 2007.
27. R. McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.
28. T. Park and K. G. Shin. Lisp: a lightweight security protocol for wireless sensor networks. *ACM Transactions on Embedded Computing Systems*, 3:634–660, 2004.
29. A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. Spins: Security protocols for sensor networks. In *Wireless Networks*, pages 189–199, 2001.
30. A. Pnueli. The temporal logic of programs. In *Proc. FOCS*, pages 46–57. IEEE Computer Society Press, 1977.
31. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.
32. S. Schewe. Solving parity games in big steps. In *FSTTCS*, volume 4805 of *LNCS*, pages 449–460. Springer, 2007.
33. S. Schewe and T. Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *ATVA*, volume 7561 of *LNCS*, pages 42–56. Springer, 2012.
34. W. van der Hoek and M. Wooldridge. Model checking cooperation, knowledge, and time - a case study. *Research in Economics*, 57(3):235–265, 2003.
35. L. Wu, K. Su, and Q. Chen. Model checking temporal logics of knowledge and its application in security verification. In *CIS*, volume 3801 of *LNAI*, pages 349–354, 2005.
36. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *LNCS*, 200(1-2):135–183, 1998.