Static analysis of parity games: alternating reachability under parity

Michael Huth[1], Jim Huan-Pu Kuo[1], and Nir Piterman[2]

[1] Department of Computing, Imperial College London
London, SW7 2AZ, United Kingdom
{m.huth, jimhkuo}@imperial.ac.uk
[2] Department of Computer Science, University of Leicester
Leicester, LE1 7RH, United Kingdom
nir.piterman@leicester.ac.uk

Abstract. It is well understood that solving parity games is equivalent, up to polynomial time, to model checking of the modal mu-calculus. It is a long-standing open problem whether solving parity games (or model checking modal mu-calculus formulas) can be done in polynomial time. A recent approach to studying this problem has been the design of *partial solvers*, algorithms that run in polynomial time and that may only solve *parts* of a parity game. Although it was shown that such partial solvers can completely solve many practical benchmarks, the design of such partial solvers was somewhat ad hoc, limiting a deeper understanding of the potential of that approach. We here mean to provide such robust foundations for deeper analysis through a new form of game, *alternating reachability under parity*. We prove the determinacy of these games and use this determinacy to define, for each player, a monotone fixed point over an ordered domain of height linear in the size of the parity game such that all nodes in its greatest fixed point are won by said player in the parity game. We show, through theoretical and experimental work, that such greatest fixed points and their computation leads to partial solvers that run in polynomial time. These partial solvers are based on established principles of static analysis and are more effective than partial solvers studied in extant work.

1 Introduction

Model checking [9, 24] is an approach to formal methods in which a system is represented as a model M, system behavior of interest is represented as a formula ϕ of a suitable temporal logic, and the question of whether the model satisfies that property (written $M \models \phi$) is decided using an algorithm parametric in M and ϕ. For infinite models, this question often is undecidable and may therefore require the abstraction of models to finite ones [2].

Program analyses (see e.g. [23]) consider programs P and aim to answer questions such as "Are there portions of code in P that can never be reached during execution?". Since exact answers may be undecidable, abstraction is often used to under-approximate or over-approximate such answers, for example, the

set of program points that can never be reached. Many program analyses can be computed by a static analysis that computes a least fixed point of a monotone function over a complete lattice; see for example Chapter 6 in [23] for more details on this approach based on worklist algorithms.

These two approaches, model checking and static analysis, appear to be quite different even though they share the need for abstraction. For example, it is not immediately clear whether each program analysis might correspond to a property ϕ of some suitable logic. But there is a body of research that points out a close relationship and connections between these approaches. For example, in [26] it is shown how data-flow analyses can be seen as instances of model checking: if programs are represented as models of a modal logic, one can capture a data-flow analysis as a formula in that modal logic, and then partially evaluate the model checker for that logic to thus implement the data-flow analyzer. This insight led to an actual methodology: in [25] one converts a program into a transition system as program model – using its operational semantics, then applies abstraction [3, 4] to eliminate details of that model that are irrelevant to the analysis/formula in question, and finally one can do model checking on the abstract model using formulas that capture the analysis in question.

These contributions furthered the understanding of how program analysis can be seen within the framework of model checking. Conversely, it turns out that the central question of model checking, whether $M \models \phi$ holds, can be computed with techniques from static analysis. In [22], an alternation-free fixed-point logic was defined and it was shown how static analysis over the resulting flow logic can decide model-checking instances for modal logics such as computation tree logic (CTL) [9]. The flow logic in [22] was also demonstrated to have applications in data-flow analysis and constraint solving [11]. In later work [28], this alternation-free least fixed-point logic was extended so that one could capture model checking of the modal mu-calculus [18] (not just of CTL) in this manner, and a Moore family result was proved for this logic; Moore families are the set of closed sets of a closure operator.

The temporal logic CTL and the linear-time temporal logic LTL can be seen as subsets of the temporal logic CTL* (see e.g. [15]). The logic CTL* can in turn be embedded into the modal mu-calculus [5], although at an exponential cost [19]. LTL and CTL capture many practically important property patterns [7] and are therefore very useful. But some have argued that these logics are mathematically somewhat ad hoc. The modal mu-calculus, on the other hand, is more canonical since it does not limit the manner in which fixed-point patterns can be composed (apart from syntactic restrictions that ensure monotonicity of meaning). It is therefore apt to understand the connections between static analysis and model checking over the modal mu-calculus as well, and the work reported in [28] shows how static analysis in the form of flow logics can capture model checking of the modal mu-calculus.

There is another important aspect to the study of such connections though. It is well understood [8, 10, 27] that model checking of the modal mu-calculus is equivalent (within polynomial time) to the solving of parity games. These are

directed graphs whose nodes are owned by one of two players and colored by a natural number. In this chapter, we assume that such graphs are finite. Plays between these players generate infinite paths in these graphs whose winners are decided by minimal colors of cycles generated by these paths. A player wins a node if she can play such that all plays beginning in that node are won by her in this manner. A central result for parity games states that these games are determined [21, 8, 29]: each node is won by exactly one of the two players. Deciding which player wins which nodes, and how they can achieve these wins is what one means by solving parity games.

Using the aforementioned results in [8, 10, 27], we can therefore understand how to use static analysis for model checking by understanding how static analyses may solve parity games. Known approaches of solving parity games in this manner, for example the ones based on *small progress measures* [17], all suffer from the fact that the height of the ordered domain derived from the parity game may be exponentially larger than that game – leading to exponential worst-case running times of least fixed-point computations in the resulting worklist algorithm that implements a static analysis. In fact, the decision problem of whether a given node in a parity game is won by a given player is in $\mathrm{UP} \cap \mathrm{coUP}$ [16], and its exact complexity has been an open problem for over twenty years now.

The work that we report here means to combine static analysis with *abstraction*. The analyses we design below run in polynomial time by construction. But this efficiency is gained by possibly *under-approximating* the solution of a parity game: the used static analysis may not decide the winners of all (or indeed some) nodes although they often solve games completely. Furthermore, in *local* modal checking (see e.g. [27]) it suffices to know whether one or several designated states satisfy a property. In the setting of parity games, this means that it may suffice to statically decide the winner of one or several nodes – which the static analyses we present here may often achieve.

Outline of chapter: In Section 2, we recall background on parity games. Our new type of alternating reachability game is defined and studied in Section 3. In Section 4, we show how this game induces monotone functions for each player of a parity game, and that we can use these functions to build static analyses of parity games that repeatedly compute greatest fixed points of such functions on (residual) games. We discuss, in Section 5, how this approach generalizes our earlier work on fatal attractors in [13]. Our experimental results are reported in Section 6, related work not discussed above already is presented in Section 7, and the chapter concludes in Section 8.

2 Background

In this section, we define key concepts of parity games, and fix technical notation used in this chapter. We write $\mathbb{N}$ for the set $\{0, 1, \dots\}$ of natural numbers. A parity game G is a tuple (V, V_0, V_1, E, c), where V is a set of nodes partitioned into possibly empty node sets V_0 and V_1, with an edge relation $E \subseteq V \times V$ (where for all v in V there is a w in V with (v, w) in E), and a coloring function

$c\colon V \to \mathbb{N}$. In figures, $c(v)$ is written within nodes v, nodes in V_0 are depicted as circles and nodes in V_1 as squares. For v in V, we write $v.E$ for node set $\{w \in V \mid (v,w) \in E\}$ of successors of v. Below we write $\mathsf{C}(G)$ for the set of colors in game G, i.e. $\mathsf{C}(G) = \{c(v) \mid v \in V\}$, and $\mathsf{C}(G)_\perp$ for set $\mathsf{C}(G) \cup \{\perp\}$.

Throughout, we write p (or sometimes p') for one of 0 or 1 and $1-p$ for the other player. In a parity game, player p owns the nodes in V_p. A play from some node v_0 results in an infinite play $\pi = v_0 v_1 \ldots$ in (V,E) where the player who owns v_i chooses the successor v_{i+1} such that (v_i, v_{i+1}) is in E. Let $\mathsf{Inf}(\pi)$ be the set of colors that occur in π infinitely often:

$$\mathsf{Inf}(\pi) = \{k \in \mathbb{N} \mid \forall j \in \mathbb{N}\colon \exists i \in \mathbb{N}\colon i > j \text{ and } k = c(v_i)\}$$

Player 0 wins play π iff $\min \mathsf{Inf}(\pi)$ is even; otherwise player 1 wins play r.

A strategy for player p is a total function $\sigma_p\colon V^* \cdot V_p \to V$ where the pair $(v, \sigma_p(w \cdot v))$ is in E for all v in V_p and w in V^*. A play π conforms with σ_p if for every finite prefix $v_0 \ldots v_i$ of π with v_i in V_p we have $v_{i+1} = \sigma_p(v_0 \ldots v_i)$. A strategy σ_p is memoryless if for all w, w' in V^* and v in V_p we have $\sigma_p(w \cdot v) = \sigma_p(w' \cdot v)$ and such a σ_p can be seen to have type $V_p \to V$.

It is well known that each parity game is determined [21, 8, 29]: (i) node set V is the disjoint union of two, possibly empty, sets W_0 and W_1, the winning regions of players 0 and 1 (respectively); and (ii) there are memoryless strategies σ_0 and σ_1 such that all plays beginning in W_0 and conforming with σ_0 are won by player 0, and all plays beginning in W_1 and conforming with σ_1 are won by player 1. Solving a parity game means computing such data $(W_0, W_1, \sigma_0, \sigma_1)$.

Throughout this chapter, we write G for a parity game (V, V_0, V_1, E, c), denote by p one of its players, and let X be a non-empty set of nodes of G. We write $x\%2$ for x modulo 2 for an integer x, and $\mathsf{Attr}_p[G, X]$ to denote the attractor of node set X for player p, which computes the standard alternating reachability of X for that player in the game graph of G (see e.g. Definition 1 in [13]).

Example 1. In the parity game G depicted in Figure 1, the winning regions are $W_1 = \{\}$ and $W_0 = V$. The memoryless strategy σ_0, defined by $\sigma_0(v_1) = v_2$, is a winning strategy for player 0 on W_0.

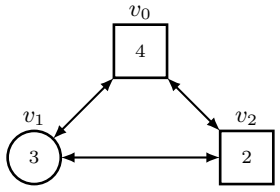

Fig. 1. A parity game: circles denote nodes in V_0, squares denote nodes in V_1.

3 Alternating reachability under parity

In this section, we generalize alternating reachability in parity game graphs, so that this reachability is aware of minimal colors encountered en route:

Definition 1. *Given parity game G, player p, and non-empty node set X, let $\pi = v_0 v_1 \ldots$ be an infinite play in G.*

1. *Player p wins play π in the* reachability game for (X, p) under parity *iff there is some $j > 0$ such that v_j is in X and $\min(\{c(v_i) \mid 0 \leq i \leq j\}\%2 = p$. Dually, player $1 - p$ wins play π in that reachability game iff she detracts from (X, p) under parity, that is to say iff for all $j > 0$ we have that v_j in X implies that $\min(\{c(v_i) \mid 0 \leq i \leq j\}\%2 = 1 - p$.*
2. *A strategy for player p' in this game is defined like a strategy for that player in the parity game G. Also, the definition of when plays conform with strategies in this game is the same as for parity game G.*
3. *Player p' wins a node v for* reachability of (X, p) under parity *iff she has a strategy $\sigma_{p'}$ such that all plays starting from v and conforming to $\sigma_{p'}$ are winning for player p' in the reachability game for (X, p) under parity.*
4. *We write $\mathsf{W}_r^p(G, X)$ for the set of nodes that player p wins in this manner (we won't need notation for the set of nodes won by player $1 - p$).*

This acceptance condition binds p to X: it is player p who wants to reach (X, p) under parity. Also, starting from X in a play does not yet mean that X has been reached. In particular, player $1-p$ wins all plays that don't visit X after the initial node. An immediate question is whether such games are determined and how complex it is to solve them. We answer these questions next.

Lemma 1. *For all parity games G, players p, and non-empty node sets X, the derived game in G of reaching (X, p) under parity is determined.*

Proof. For a color i in $\mathsf{C}(G)$ and node set $S \subseteq V$ let $S_i = \{v \in S \mid c(v) = i\}$ and $S_{\geq i} = \{v \in S \mid c(v) \geq i\}$. Also, let $C = \{c \in \mathsf{C}(G) \mid c\%2 = p\}$. The set of winning plays for player p in the reachability game for (X, p) under parity is the union of $(V_{\geq i}^* \cdot V_i \cdot V_{\geq i}^* \cdot X_{\geq i} \cdot V^\omega) \cup (V_{\geq i}^+ \cdot X_i \cdot V^\omega)$ over all i in C. Note that, for each such i, both expressions in this union capture the non-deterministic choice of reaching X in Definition 1. The difference in these expressions is merely that the minimal color i may be witnessed before that non-deterministic choice of reaching X. The set of winning plays for player p is thus a Borel definable set of paths. From the Borel determinacy of turn-based games [20] it therefore follows that the game is determined. □

Next, we derive from parity game G and node set X a game graph that reduces reachability of (X, p) under parity to (the usual alternating) reachability in the derived game graph. This derived game has nodes of form (v, l) where l records the history of the minimal color encountered so far. In particular, we use $l = \bot$ to model that a play is just beginning.

Definition 2. *For parity game $G = (V, V_0, V_1, E, c)$, player p, and non-empty node set X, game graph $G^p_X = (V \times \mathsf{C}(G)_\perp, E')$ is defined as follows: For c in $\mathsf{C}(G)_\perp$, player 0 owns all nodes (v, c) with $v \in V_0$. Player 1 owns all nodes (v, c) with $v \in V_1$. And the edge relation $E' \subseteq (V \times \mathsf{C}(G)_\perp) \times (V \times \mathsf{C}(G)_\perp)$ is defined as*

$$E' = \{((v, \perp), (v', \min(c(v), c(v')))) \mid (v, v') \in E\} \cup \{((v, c), (v', \min(c, c(v')))) \mid (v, v') \in E, c \in \mathsf{C}(G), (v \notin X \text{ or } c\%2 \neq p)\} \quad (1)$$

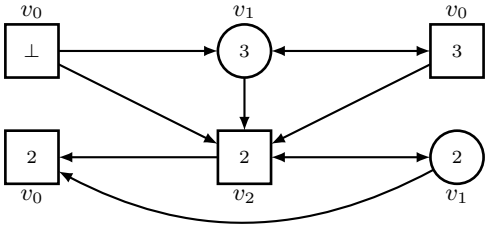

Fig. 2. Game graph G^p_X for G from Figure 1 and X being $\{v_0\}$; only nodes and edges reachable (in non-alternating sense) from $X \times \{\perp\}$ in G^p_X are shown, as this is all needed for deciding which nodes in X are contained in $\mathsf{W}^p_r(G, X)$. The winning strategy for player 0 requires her to make different choices from the same nodes of G when they are combined with different colors: player 0 needs to move from $(v_1, 3)$ to $(v_2, 2)$ and from $(v_1, 2)$ to $(v_0, 2)$ in G^p_X

Note that relation E' is even contained in $(V \times \mathsf{C}(G)_\perp) \times (V \times \mathsf{C}(G))$ and contains dead ends (nodes that don't have outgoing edges in the game graph). The latter is not an issue since all dead ends in G^p_X are target nodes for the alternating reachability in G^p_X. Figures 2 and 3 show examples of this construction.

The intuition of game graph G^p_X is that player p can win node v in G for reaching (X, p) under parity iff player p can win the (alternating) reachability game in G^p_X for target set $X \times \{c \in \mathsf{C}(G) \mid c\%2 = p\}$. We state this formally:

Theorem 1. *For G and G^p_X as above, let Z be $X \times \{c \in \mathsf{C}(G), c\%2 = p\}$ and W be $\{v \in V \mid (v, \perp) \in \mathsf{Attr}_p(G^p_X, Z)\}$. Then W is the winning region of player p in G for reachability of (X, p) under parity.*

Proof. First, let $W_p = \mathsf{W}^p_r(G, X)$ be the winning region of player p in G for reachability of (X, p) under parity. Since this game has a Borel defined winning condition, there exists a strategy $\tau \colon V^* \times V_p \to V$ such that all plays conforming with τ and starting in W_p are won by player p for reachability of (X, p) under parity.

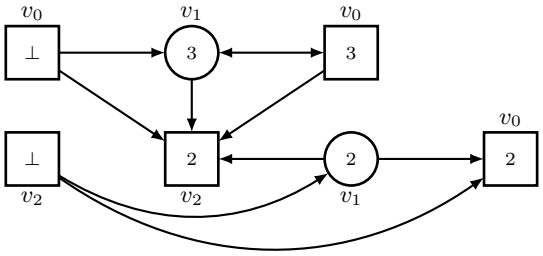

Fig. 3. Game graph G_X^p for G from Figure 1 and X being $\{v_0, v_2\}$. As in Fig. 2, only nodes and edges reachable (in non-alternating sense) from $X \times \{\bot\}$ in G_X^p are shown. The winning strategy for player 0 allows her to make choices that do not depend on the color annotating the states. She can move from (v_1, c) to $(v_2, 2)$ regardless of the value of c.

We write τ' for the same strategy but applied to G_X^p whilst ignoring the second component of nodes in G_X^p. (We note that E' updates the second component of nodes in G_X^p deterministically.) Consider a play π in G_X^p that starts in $W_p \times \{\bot\}$ and conforms with τ'. The projection of π onto the first components of its nodes is a play in G that starts in W_p and conforms with τ. Therefore, that play is won by player p in G, and so it is also won by player p in G_X^p.

Second, it remains to show that $\{v \mid (v, \bot) \in \mathsf{Attr}_p(G_X^p, Z)\}$ is contained in W_p. Let δ' be a winning (attractor) strategy for player p in G_X^p for the attractor $\mathsf{Attr}_p(G_X^p, Z)$. As an attractor strategy, δ' is memoryless. That is, for every node $(v, c) \in V_p \times \mathsf{C}(G)_\bot$ we can write $\delta'(v, c)$ and this is in $V \times \mathsf{C}(G)$. For a sequence of nodes $\pi = v_0, \ldots, v_n$, let $c(\pi)$ denote $\min\{c(v_i) \mid 0 \leq i \leq n\}$. Let $\delta \colon V^* \cdot V_p \to V$ be the strategy obtained from δ' by setting $\delta(\pi \cdot v) = \delta'(v, c(\pi \cdot v))$. Then δ is a strategy in G. Every play that begins in $W = \{v \mid (v, \bot) \in \mathsf{Attr}_p(G_X^p, Z)\}$ and conforms with δ in G can be extended to a play in G_X^p that begins in $\mathsf{Attr}_p(G_X^p, Z)$ and conforms with δ' by adding the deterministic second components. Therefore, this play is winning for player p in G_X^p. It follows from the construction of E' that player p reaches X from $W = \{v \mid (v, \bot) \in \mathsf{Attr}_p(G_X^p, Z)\}$ such that the minimal color encountered on the way in G has parity p. □

This theorem also gives us an upper bound on the complexity of solving games for reachability of (X, p) under parity, noting that alternating reachability is linear in the number of edges of the game graph, and that G_X^p has at most $|E| \cdot |\mathsf{C}(G)|$ many edges.

Corollary 1. *For G, p, and X as above, the reachability game in G for (X, p) under parity can be solved in time $O(|E| \cdot |\mathsf{C}(G)|)$.*

We later consider the issue of whether memoryless strategies suffice for winning in G for reachability of (X, p) under parity (they do not). However, from the proof of Theorem 1 it follows that the size of memory required is bounded by the number of colors in the game (plus 1).

4 Monotone functions for a partial solver

Let player p win node v for reaching (X, p) under parity in G. Then player p can make sure that X is reached from v, and that X can be reached from v such that the minimal color encountered so far has color parity p. If all nodes in X are won by player p, node set X is then won by player p in the parity game G:

Lemma 2. *For all G, X, and p such that X is contained in $\mathsf{W}_r^p(G, X)$, player p wins all nodes from X in parity game G.*

Proof. For each v in X, player p has a strategy σ_v with finite memory such that all plays beginning at node v and conforming with σ_v will reach again some node in X such that the minimal color of that finite play has parity p. Because X is contained in $\mathsf{W}_r^p(G, X)$, player p can compose all these strategies to a strategy σ_p with finite memory as follows:

From v_0 in X, she plays conform with σ_{v_0} until a finite play $v_0 \dots v_k$ is generated such that v_k is in X and $\min\{c(v_j) \mid 0 \leq j \leq k\}$ has color parity p. We know that such a finite subplay will be generated by σ_{v_0} as it is a winning strategy for player p witnessing that v is in $\mathsf{W}_r^p(G, X)$. At node v_k, player p now continues to play conform with strategy σ_{v_k}. She can continue this composition pattern to generate an infinite play $\pi = v_0 \cdots v_k \cdots$ that is partitioned into infinitely many finite sub-plays $(\pi^i)_{i \geq 0}$ that begin and end in X (and may contain other nodes in X) and that each have some minimal color c_i with parity p.

Since G has only finitely many nodes, this means that all colors that occur infinitely often in π are greater than or equal to some color that occurs as minimal color in infinitely many sub-plays π^i (and so has parity p and also occurs infinitely often in π). Therefore, player p wins π in the parity game G and so the described strategy is also winning for player p on node set X in parity game G. □

We now put this lemma to use by characterizing such winning node sets as fixed points of a monotone function. For that, let V^p be the (possibly empty) set of nodes of G that have color parity p, that is V^p equals $\{v \in V \mid c(v)\%2 = p\}$. Let us consider the function F_G^p, defined by

$$F_G^p \colon \mathbb{P}(V^p) \to \mathbb{P}(V^p), \qquad F_G^p(X) = X \cap \mathsf{W}_r^p(G, X) \tag{2}$$

Lemma 2 then says, in particular, that all non-empty fixed points of F_G^p are node sets won by player p in parity game G. That function is monotone:

Lemma 3. *For all G and p, function F_G^p defined in (2) is monotone.*

Proof. Let X and Y be subsets of V^p such that X is contained in Y. We need to show that $F_G^p(X)$ is contained in $F_G^p(Y)$ as well. By definition of F_G^p, monotonicity follows if X or Y is empty. So let X and Y be non-empty. Since $X \subseteq Y$ and since intersection is monotone, it suffices to show that $\mathsf{W}_r^p(G, X)$ is contained in $\mathsf{W}_r^p(G, Y)$. So let v be in $\mathsf{W}_r^p(G, X)$. Then player p has a winning strategy that

ensures that all plays from node v reach X such that the minimal color encountered thus far has parity p. Since X is contained in Y, this means that all such plays will also reach Y with minimal color encountered en route. Therefore, the winning strategy for $v \in \mathsf{W}_r^p(G, X)$ is also a winning strategy for $v \in \mathsf{W}_r^p(G, Y)$, and so v is in $\mathsf{W}_r^p(G, Y)$ as claimed. □

Neither the monotonicity of F_G^p nor the result of Lemma 2 depend on the fact that all nodes in X have color parity p, nor that anything is known about colors in X; for Lemma 2, it only matters that all nodes in X are also in $\mathsf{W}_r^p(G, X)$. It is of interest to note that function F_G^p would not be monotone if we were to change the acceptance condition for reaching (X, p) under parity to mean that player p has to get minimal color parity p at the *first* time she reaches X after the first node in the play. Formally, player p would win a play π iff there were some $j > 0$ with π_j in X such that $\min\{c(\pi_i) \mid 0 \leq i \leq j\}\%2$ equals p *and* there were no k with $0 < k < j$ such that π_k would be in X. The resulting non-monotonicity of this modified acceptance condition is illustrated in Figure 4.

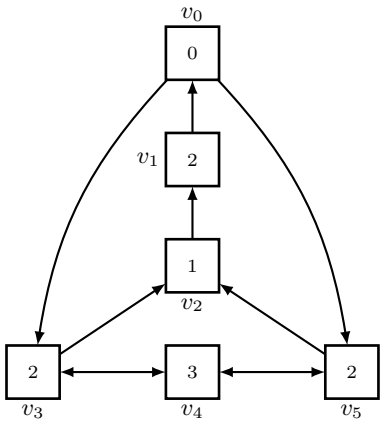

Fig. 4. Function F_G^0 is no longer always monotone when $\mathsf{W}_r^p(G, X)$ has acceptance condition that looks at the minimal color of the prefix for the *first* reached element of X instead of a non-deterministally chosen first or future element of X. For G above and $X = \{v_3, v_5\}$ and $Y = V^0$, we would then have $X \subseteq Y$ but $F_G^0(X) = \{v_3, v_5\}$ would not be contained in $F_G^0(Y) = \{v_0, v_1\}$ under that modified acceptance condition

Monotonicity of F_G^p means that either all its fixed points are empty or its greatest fixed point is non-empty. This suggests an algorithm that recursively computes such greatest fixed points for each player p, and removes non-empty ones as being recognized winning regions for player p from parity game G until either G is solved completely or both F_G^0 and F_G^1 have only empty fixed points. The pseudo-code for this algorithm `psolC` is shown in Figure 5.

```
psolC(G = (V, V_0, V_1, E, c)) {
  W = tryPlayer(G, 0);
  if (W ≠ ∅) {
   return psolC(G \ W);
  } else {
    W = tryPlayer(G, 1);
    if (W ≠ ∅) {
      return psolC(G \ W);
    } else {
       return G;
    }
  }
}

tryPlayer(G, p) {
  X = fixedpoint(G, { v in V | c(v)%2 = p }, p);
  if (X ≠ {}) { return Attr_p[G, X]; }
  else { return ∅; }
}

fixedpoint(G, X, p) {
    W = V;
    repeat { X = X ∩ W; W = X ∩ W_r^p(G, X); } until (X ⊆ W)
    return X;
}
```

Fig. 5. Partial solver `psolC`: in game G^p_X, only $X \cap \mathsf{W}^p_r(G, X)$ needs to be computed. So this is implemented by only constructing nodes and edges in G^p_X that are reachable from $X \times \{\bot\}$ in the non-alternating sense

When a greatest fixed point is discovered for player p, the partial solver removes the p attractor of that fixed point in parity game G from G, not just the fixed point. This is sound since winning node sets for players in parity games are closed under attractors for those players. The pseudo-code does not show the accumulation of the removed node sets into winning regions, as these are routine administrative matters that only detract from the essence of this partial solver.

We show soundness and upper bounds on the complexity of `psolC`:

Theorem 2. *Let G be a parity game as above. Then* $\mathtt{psolC}(G)$ *runs in time* $O(|E| \cdot |\mathsf{C}(G)| \cdot |V|^2)$*, space* $O(|E| \cdot (1 + |\mathsf{C}(G)|))$*, and all node sets* $\mathsf{Attr}_p[G, X]$ *it removes from (residual instances of) G are won by player p in the parity game G.*

Proof. Since $\mathsf{W}^p_r(G, X)$ can be computed in $O(|E| \cdot |\mathsf{C}(G)|)$, each fixed-point computation in $\mathtt{psolC}(G)$ runs in $O(|E| \cdot |\mathsf{C}(G)| \cdot |V|)$ as it can have at most $|V|$ iterations. But there can also be at most $2 \cdot |V|$ many such fixed-point

computations in total as each subsequent such computation requires that at least one node has been removed from G beforehand.

The upper bound on the space complexity follows since the size of G^p_X is the dominating factor for space requirements of `psolC` – larger than the size of G, since there are at most $|E| \cdot (1 + |\mathsf{C}(G)|))$ many edges in G^p_X, and since there is no need to keep copies of G^p_X once $X \cap \mathsf{W}^p_r(G, X)$ has been computed in `psolC`.

The remaining soundness claim for partial solver `psolC` directly follows from Lemma 2 and from the aforementioned fact that winning regions of players in parity games are closed under attractors of those players. The latter also ensures that winning regions of recursive instances of G are winning regions of G. □

It turns out that reachability of (X, p) under parity cannot be solved with memoryless strategies in general, in contrast to the solving of parity games:

Theorem 3. *Solving alternating reachability under parity requires finite memory in general.*

Proof. It suffices to give an example where this is the case. Recall the simple parity game G from Figure 1. Let p be 0 and X be $\{v_0\}$. Then $\mathsf{W}^0_r(G, X)$ equals V and so player 0 wins all nodes for reachability of $(X, 0)$ under parity. But she cannot realize this with a *memoryless* strategy σ_0, for either $\sigma_0(v_1)$ would equal v_2 (and then player 1 can detract from X by moving from v_2 back to v_1) or $\sigma_0(v_1)$ would have to equal v_0 (in which case player 1 can move from v_0 to v_1 to generate an infinite play in which all prefixes that reach X have odd color 3). Let the strategy $\sigma'_0 \colon V^* \cdot \{v_1\} \to V$ be defined, for all w in V^*, by $\sigma'_0(w \cdot v_1) = v_0$ if v_2 is in w; and $\sigma'_0(w \cdot v_1) = v_2$ otherwise. Strategy σ'_0 has finite memory and is winning on all nodes for reachability of $(X, 0)$ under parity: σ'_0 ensures that v_0 is reached, and that v_0 is reached only after v_2 has been reached. This means that the minimal color encountered until X is reached equals 2, a win for player 0. □

The implication of Theorem 3 is that even though `psolC` identifies winning regions in the parity game the strategies that it allows us to construct, in general, require memory. At the same time, we know that there exist memoryless strategies for both players from their respective winning regions in the parity game.

Although finite memory is required in general, we note that $Y = V^0$ is the greatest fixed point of F^G_0 for G from Figure 1, and that the memoryless strategy σ_0 above *is* winning for $\mathsf{W}^0_r(G, Y) = V$. This raises the question of whether non-empty greatest fixed points of F^p_G ever require corresponding winning strategies *with* finite memory or whether they always can be memoryless. This is also apparent in the derived games G^0_X and X^0_Y depicted in Figures 2 and 3, respectively. We formulate this problem as a research question:

Question 1. Is there a parity game G and player p where the greatest fixed point X of F^p_G is non-empty and player p does not have memoryless strategies for witnessing that X is contained in $\mathsf{W}^p_r(G, X)$?

If no finite memory is needed for greatest fixed points of F_G^p, then `psolC` might be able to compute memoryless winning strategies for parity game G. Let us next give an example of how `psolC` may solve games completely:

Example 2. Let us consider the execution of `psolC`(G) for parity game G in Figure 4 (for the acceptance condition as in Definition 1). Initially, $p = 0$ and $X = \{v_0, v_1, v_3, v_5\} = G^0$. Then `psolC` detects in `fixedPoint` that X is the greatest fixed point of F_G^0 and removes its 0 attractor in G (which is all of V) from G. Thus `psolC` completely solves G and recognizes that all nodes are won by player 0. Note that X is a fixed point of F_G^0 since $\mathsf{W}_r^0(G, X)$ equals V: (i) player 0 wins node v_0 as player 1 can only move to v_3 or v_5 from there and so reach X with minimal color 0; (ii) player 0 wins node v_1 since player 1 can only move to v_0 from there and so reach X with minimal color 0; (iii) player 0 wins node v_2 since player 1 can only generate the prefix $v_2v_1v_0$ from there and so get minimal color 0 for this *second* reach of X; (iv) player 0 wins v_3 since player 1 can either move from there to v_2 and so generate a prefix $v_3v_2v_1v_0$ with minimal color 0 for his *second* reach of X or player 1 can move to v_4 from where she can only move to X with minimal color 2 for the first reach of X; (v) player 0 wins v_5 for symmetric reasons; and (vi) player 0 wins v_4 because player 1 can only reach X from here with minimal color 2 on the first reach of X.

Solver `psolC` is partial in that it may not solve even a single node in a parity game. We illustrate this with an example:

Example 3. Figure 6 shows a parity game G for which `psolC` solves no nodes at all. For $p = 0$, set X is initially $V \setminus \{v_1\}$. (i) Node v_0 is lost by player 0 since player 1 can move from there into the cycle $(v_2v_6)^\omega$ with minimal color 1. Player 0 wins all other nodes in X. Therefore, the next value of X equals $\{v_2, v_3, v_4, v_5, v_6\}$. (ii) Now, nodes v_4 and v_5 are lost by player 0, as player 1 can move from them to node v_0 (which is no longer in X) and then play as for the initial X to get minimal color 1. Player 0 wins all other nodes in X. Therefore, the next value of X equals $\{v_2, v_3, v_6\}$. (iii) Next, node v_3 is lost by player 0, as player 1 can move from there directly to node v_4 (which is no longer in X) and then enter the cycle $(v_0v_5)^\omega$ and so avoid X altogether. Player 0 wins nodes v_2 and v_6 though. Therefore, the next value of X equals $\{v_2, v_6\}$. (iv) Now, player 0 loses v_2 as player 1 can avoid reaching that node again from v_2. Player 0 still wins node v_6. Thus, the next value of X equals $\{v_6\}$. (v) Finally, player 1 can avoid reaching X again from node v_6 and so wins v_6, making X empty.

Clearly, F_G^1 computes an empty fixed point as all nodes in parity game G are won by player 0. The inability of `psolC` to solve even a single node in G seems to stem from the fact that the acceptance condition for $\mathsf{W}_r^0(G, X)$ captures a *weak* parity acceptance condition [1] and not a parity acceptance condition.

We could extend the types of F_G^p to be $\mathbb{P}(V) \to \mathbb{P}(V)$. The proofs for monotonicity and for fixed points being won by player p in the parity game G would still carry through then. It may be of interest to compare a variant of `psolC` based on greatest fixed points for this extended type of F_G^p to `psolC`: that variant may run slower in practice but may solve more nodes in G. However, it will still be

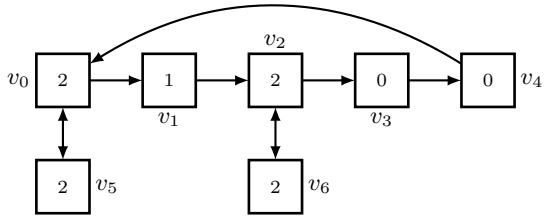

Fig. 6. Parity game G, owned by player 1, won by player 0, and where `psolC` cannot solve even a single node

a partial solver as can be seen from Example 3: for the version of `psolC` based on this extended type, both v_0 and v_1 would be removed from initial $X = V$ in the first iteration and so this still would compute empty fixed points only.

5 Fatal attractors

Our work in [13] defined and studied *monotone* attractors and built partial solvers out of them. Let X be a non-empty node set of G where all nodes in X have color c, and set p to be $c\%2$. Monotone attractors $\mathsf{MA}(X)$ were defined in [13]. For X as above, and subsets A of V this definition is as follows:

$$\begin{aligned} \mathsf{mpre}_p(A, X, c) &= \{v \in V_p \mid c(v) \geq c \wedge v.E \cap (A \cup X) \neq \emptyset\} \cup \\ &\quad \{v \in V_{1-p} \mid c(v) \geq c \wedge v.E \subseteq A \cup X\} \\ \mathsf{MA}(X) &= \mu Z.\mathsf{mpre}_p(Z, X, c) \end{aligned} \tag{3}$$

where $\mu Z.f(Z)$ denotes the least fixed point of a monotone function $f\colon \mathbb{P}(V) \to \mathbb{P}(V)$. It follows that $\mathsf{MA}(X)$ is the set of nodes in G from which player p can attract to X whilst avoiding nodes of color less than c. In [13], we called such an X *fatal* if all of X is in that attractor (i.e. when $X \subseteq \mathsf{MA}(X)$). In Theorem 2 in [13], we showed that all such fatal attractors are won by player p.

To relate this to our work in this chapter, an infinite play π would be won in this monotone attractor game by player p iff there is some $j > 0$ with π_j in X and $c(\pi_i) \geq c$ for all i with $0 \leq i < j$; so X can be reached on π with minimal color c at π_j. This implies that all such fatal attractors X with node color c are fixed points of F_G^p and are therefore contained in the greatest fixed point of F_G^p. We can use this to prove that `psolC` is more effective than the partial solver `psolB` defined in [13]:

Theorem 4. *Let* `psolB` *be the partial solver defined in Figure 7 and let G be a parity game. The call* `psolC`(G) *decides the winner of all nodes for which call* `psolB`(G) *decides a winner.*

```
psolB(G = (V, V_0, V_1, E, c)) {
  for (colors d in descending ordering) {
    X = { v in V | c(v) = d };
    cache = {};
    while (X ≠ {} && X ≠ cache) {
      cache = X;
      if (X ⊆ MA(X)) { return psolB(G \ Attr_{d%2}[G, MA(X)])
      } else { X = X ∩ MA(X); }
    }
  }
  return G
}
```

Fig. 7. Partial solver `psolB` from [13] (figure is a reproduction of Fig. 3 in [13])

Proof. For all players p, the acceptance condition for monotone attractors as discussed above implies that all fatal attractors for that player in G (node sets X of some color c with parity p such that $X \subseteq \mathsf{MA}(X)$) are contained in the greatest fixed point Z of F^p_G. By Theorem 5 in [13], the order of fatal attractor detection does not affect the output of partial solver `psolB`. Therefore, we can assume that all fatal attractors X for player p are contained in the greatest fixed point Z of F^p_G. But by monotonicity, their p-attractors $\mathsf{Attr}_p[G, X]$ are then also contained in the p-attractor $\mathsf{Attr}_p[G, Z]$ of Z. Thus, it follows that all nodes that are decided by `psolB`(G) are also decided by `psolC`(G). □

In [13], we also studied a more precise but more complex partial solver `psolQ`. Although the design of `psolQ` has superficial similarities to that of `psolC`, the latter is more precise: at noted in [13], `psolQ` does not solve even a single node for the parity game in Figure 8. But `psolC` solves this game completely.

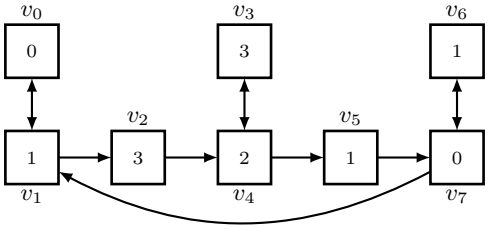

Fig. 8. A 1-player parity game that `psolC` solves completely (as $\{v_0, v_4, v_7\}$ is greatest fixed point of F^0_G) but for which `psolQ` in [13] solves no nodes (figure is Fig. 5 in [13])

6 Experimental results

By Theorem 4, we know that `psolC` will solve completely all games that `psolB` solves completely. From [13], we know that `psolB` completely solves many structured benchmarks. Therefore, there is little value in running `psolC` over these structured benchmarks again. This is why we focused our experimental effort here an random parity games.

We now report our experiments we did on randomly generated games. The aims of these experiments are

1. to experimentally confirm that `psolC` solves all nodes that `psolB` solves, as proved in Theorem 4
2. to compare running times of `psolC` and `psolB` over a large set of random games
3. to determine game configurations for which `psolC` does not really solve more than `psolB` does.

All our experiments were conducted on a test server that has two Intel® E5 CPUs, with 6-core each running at 2.5GHz and 48G of RAM. Experiments were grouped into game configurations, where we generated 100,000 games for each such configuration and ran `psolB` and `psolC` against these games. We also used Zielonka's solver [29] for regression tests to ensure that `psolB` and `psolC` correctly under-approximate winning regions, all of these tests passed.

The game configurations used are shown in the "Game Mode" column of Figure 9. Each such mode is denoted by *xx-yy-aa-bb*. The *xx* is the number of nodes in a game, and the owners (player 0 or 1) of the nodes are chosen independently at random. The color of each node is also uniformly chosen from set $\{0, 1, \ldots, yy\}$, and has between *aa* and *bb* out-going edges to randomly selected successors in the game.

We now summarize key facts that we can observe from the experimental results shown in Figure 9:

- `psolB` has never solved more nodes than `psolC`, experimentally confirming Theorem 4 (column #10).
- For games with low edge density (i.e., when *aa-bb* equals 1-5), `psolC` solves more than `psolB` for around 10% of games (#9).
- For games with higher edge density (i.e., when *aa-bb* is different from 1-5), `psolC` doesn't appear to have an effect over `psolB` (#9).
- `psolC` takes significantly more time to execute than `psolB` for high edge density games (#2).
- Our experimental results suggest that the `psolC` lapse time increases as the color cap increases, whereas we don't observe a similar increase for `psolB` (#2 and #3).

We note that these experiments took quite some time to complete. For example, the total running time of `psolC` for these 800,000 random games was more than 28 days (if converted to calendar time). The experimental data we collected

Game Mode	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
500-5-1-5	100000	384.17	15.64	18.63	18545	19590	18406	80410	1184	0
500-5-1-100	100000	724.14	17.91	51.76	1016	1016	1016	98984	0	0
500-5-5-10	100000	203.67	8.71	14.44	0	0	0	100000	0	0
500-5-50-250	100000	1157.37	31.06	119.10	0	0	0	100000	0	0
500-50-1-5	100000	2522.47	14.30	20.86	18166	19066	17962	80934	1104	0
500-50-1-100	100000	6807.46	18.57	54.58	992	992	992	99008	0	0
500-50-5-10	100000	2155.34	8.35	14.38	0	2	0	99998	2	0
500-50-50-250	100000	10282.66	34.36	135.29	0	0	0	100000	0	0

Fig. 9. Our experimental results for the partial solver `psolC`. The legend for the 10 data columns above is given in Table 1.

#1. Total number of games.
#2. Average `psolC` lapse time (ms).
#3. Average `psolB` lapse time (ms).
#4. Average `zlka` lapse time (ms).
#5. How often G'_C is not 0.
#6. How often G'_B is not 0.
#7. How often $G'_C = G'_B$ and G'_C is not 0.
#8. How often $G'_C = G'_B$ and G'_C is 0.
#9. How often `psolC` solves more than `psolB`.
#10. How often `psolB` solves more than `psolC`.

Table 1. Legend for experimental data shown in Figure 9: G'_B represents the number of games not completely solved by `psolB`. Similarly, G'_C represents the number of games not completely solved by `psolC`.

suggest that the comparison between `psolB` and `psolC` is bimodal on random games: either `psolC` is no more effective than `psolB` on a given game mode, or it appears to be more effective on about 10% of games for a given game mode.

The partial solver `psolC` may therefore have more theoretical than practical value. However, a staging of `psolB` and `psolC` may work reasonably well in practice: on input game G, first run `psolB` to obtain residual game G'; and then run `psolC` only on G' and only when G' is not empty.

7 Other related work

Some easy static analyses for parity games have become part of the folklore of how to preprocess parity games. For example, the tool PGSolver can eliminate self-loops (nodes v with (v, v) in E) and dead ends (nodes v for which there is no w with (v, w) in E) [12]. The latter can be seen as justification for defining parity games not to have dead ends, as we have done in this chapter.

In [17], progress measures are defined and recognized as representations of winning strategies. A monotone function over a complete lattice is then defined such that pre-fixed points of that function capture progress measures. A least fixed-point computation therefore can compute the winning region and a winning strategy for a chosen player. This algorithm has exponential running time, since the complete lattice may be exponentially larger than the parity game. However, the algorithm runs on polynomial space, unlike some other known algorithms for solving parity games.

Our work relates to research on the descriptive complexity of parity games. In [6], it is investigated whether the winning regions of players in parity games can be defined in suitable logics. We mention two results from this paper: it is shown that this is indeed possible for guarded second-order logic (even for infinite game graphs with an unbounded number of colors); and for an arbitrary finite game graph G (the setting of our chapter), it is proved that least fixed-point logic can define the winning regions of G iff these winning regions are computable in polynomial time.

In [14], a transformation is studied that can map a partial solver ρ for parity games to another partial solver $\mathsf{lift}(\rho)$ that first applies ρ until it has no effect on the residual game. Then, $\mathsf{lift}(\rho)$ searches for some node v in V_p with more than one outgoing edge such that the commitment to one such edge (i.e. the removal of all other edges outgoing from v) would make partial solver ρ discover that node v is won by player $1 - p$ in that modified game. If so, it is sound to remove edge (v, w) from G and then try $\mathsf{lift}(\rho)$ again until no such effect can be observed for both p. It was proved in [14] that $\mathsf{lift}(\rho)$ is sound if ρ is sound, idempotent, and satisfies a locality principle; and it was shown that `psolB` satisfies these properties.

8 Conclusions

In this chapter, we studied how one may define static analyses of parity games that run in polynomial time and space and compute parts of the games' winning regions. In particular, the quality of such a static analysis could then be measured by how often it computes winning regions completely, or by what percentage of the winning region it computes across a range of random and structured benchmarks. We developed firm foundations for designing such static analyses, using a novel kind of game derived from parity games: reachability under parity. The intuition of such a game is that player p can reach a node set X whilst ensuring that the minimal color encountered en route has parity p.

We showed that such new reachability games are determined, demonstrated how one can implement their solution efficiently, and used this notion of game to define monotone functions over parity games – one for each player of the parity game. The greatest fixed-points of these functions were proved to be contained in the winning region of the corresponding player in the parity game. This insight led us to design a partial solver `psolC` and its experimental evaluation demonstrated that it is a powerful static analysis of parity games that can solve completely many types of random and structured benchmarks. Theoretical analysis also showed that these monotone functions generalize, in a more canonical and less ad hoc manner, work on fatal attractors that we had conducted previously [13]. In particular, we proved that `psolC` is more effective that the partial solver `psolB` in [13] that performed best in practice.

The decision problem for parity games, whether a given node is won by a given player, is in $\mathrm{UP} \cap \mathrm{coUP}$ [16] and so contained in $\mathrm{NP} \cap \mathrm{coNP}$. It is therefore perhaps no great surprise that all known algorithms that completely compute such winning regions run in worst-case exponential or sub-exponential time in the size of these games. One may therefore think of our chapter as taking a complementary approach to attempting to answer the longstanding open problem of the exact complexity of said decision problem for parity games: how to design static analyses that run in polynomial time (relatively easy to do) and that are provably computing the exact winning regions of all parity games (likely very hard to do under these constraints of efficient static analysis). We hope that the reader may find this approach to be of genuine interest so that he or she may pursue it further.

References

1. Chatterjee, K.: Linear time algorithm for weak parity games. CoRR **abs/0805.1391** (2008)
2. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5), 1512–1542 (1994)
3. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977, pp. 238–252 (1977)

4. Cousot, P., Cousot, R.: Abstract interpretation: past, present and future. In: Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014, p. 2 (2014)
5. Dam, M.: CTL* and ECTL* as fragments of the modal mu-calculus. Theor. Comput. Sci. **126**(1), 77–96 (1994)
6. Dawar, A., Grädel, E.: The descriptive complexity of parity games. In: Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings, pp. 354–368 (2008)
7. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering, ICSE '99, pp. 411–420 (1999)
8. Emerson, E., Jutla, C.: Tree automata, μ-calculus and determinacy. In: Proc. 32nd IEEE Symp. on Foundations of Computer Science, pp. 368–377 (1991)
9. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. **2**(3), 241–266 (1982)
10. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings, pp. 385–396 (1993)
11. Filipiuk, P., Nielson, F., Nielson, H.R.: Layered fixed point logic. CoRR **abs/1204.2768** (2012)
12. Friedmann, O., Lange, M.: Solving parity games in practice. In: Z. Liu, A. Ravn (eds.) Proc. of Automated Technology for Verification and Analysis, *Lecture Notes in Computer Science*, vol. 5799, pp. 182–196. Springer (2009)
13. Huth, M., Kuo, J.H., Piterman, N.: Fatal attractors in parity games. In: Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings, pp. 34–49 (2013)
14. Huth, M., Kuo, J.H., Piterman, N.: Fatal attractors in parity games: Building blocks for partial solvers. CoRR **abs/1405.0386** (2014)
15. Huth, M., Ryan, M.D.: Logic in computer science - modelling and reasoning about systems (2. ed.). Cambridge University Press (2004)
16. Jurdziński, M.: Deciding the winner in parity games is in UP $\cap$ co-UP. Inf. Process. Lett. **68**, 119–124 (1998)
17. Jurdziński, M.: Small progress measures for solving parity games. In: Proc. 17th Symp. on Theoretical Aspects of Computer Science, *Lecture Notes in Computer Science*, vol. 1770, pp. 290–301. Springer-Verlag (2000)
18. Kozen, D.: Results on the propositional mu-calculus. Theor. Comput. Sci. **27**, 333–354 (1983). DOI 10.1016/0304-3975(82)90125-6. URL http://dx.doi.org/10.1016/0304-3975(82)90125-6
19. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47**(2), 312–360 (2000)
20. Martin, D.A.: Borel Determinacy. Annals of Mathematics **102**(2), 363–371 (1975)
21. Mostowski, A.W.: Games with forbidden positions. Tech. Rep. 78, University of Gdańsk (1991)
22. Nielson, F., Nielson, H.R.: Model checking *Is* static analysis of modal logic. In: Foundations of Software Science and Computational Structures, 13th International

Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings, pp. 191–205 (2010)
23. Nielson, F., Nielson, H.R., Hankin, C.: Principles of program analysis (2. corr. print). Springer (2005)
24. Queille, J., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings, pp. 337–351 (1982)
25. Schmidt, D.A., Steffen, B.: Program analysis *as* model checking of abstract interpretations. In: Static Analysis, 5th International Symposium, SAS '98, Pisa, Italy, September 14-16, 1998, Proceedings, pp. 351–380 (1998)
26. Steffen, B.: Data flow analysis as model checking. In: Theoretical Aspects of Computer Software, International Conference TACS '91, Sendai, Japan, September 24-27, 1991, Proceedings, pp. 346–365 (1991)
27. Stirling, C.: Lokal model checking games. In: CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings, pp. 1–11 (1995)
28. Zhang, F., Nielson, F., Nielson, H.R.: Model checking as static analysis: Revisited. In: Integrated Formal Methods - 9th International Conference, IFM 2012, Pisa, Italy, June 18-21, 2012. Proceedings, pp. 99–112 (2012)
29. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science **200**(1–2), 135–183 (1998)