Computing safe winning regions of parity games in polynomial time

(Extended Abstract)

Adam Antonik, Nathaniel Charlton, and Michael Huth
Department of Computing, South Kensington campus
Imperial College London, London, SW7 2AZ, United Kingdom
{aa1001, nac103, M.Huth}@doc.imperial.ac.uk

Abstract

We propose a pattern for designing algorithms that are in P by construction and under-approximate the winning regions of both players in parity games. This approximation is achieved by the interaction of finitely many aspects, each performing an efficient static analysis. We present several such aspects and illustrate their relative precision and interaction.

Keywords: model checking, partial information, abstraction, parity games

1 Introduction. Parity games G (e.g. [1]) have finite or infinite plays on labelled, directed graphs (V, E) and are played between two players (0 and 1) where the winning condition for infinite plays is based on parities derived from a priority function $\chi\colon V \to \{0, 1, \dots, d-1\}$. Game G is then said to have index d. Parity games are determined (e.g. [9, 4]): each player has a winning region of game positions for which she has a memoryless winning strategy, and these two regions form a partition of the set of game positions. For an example, consider the parity game in Fig. 1. A possible infinite play is $v_7 v_0 v_7 v_0 \cdots = (v_7 v_0)^\omega$ which is won by player 1 as the largest priority $\chi(v)$ that occurs infinitely often in that play is 1 and therefore odd (technically, a maximum parity acceptance condition). In this game, the winning region for player 0 is $\{v_4, v_5, v_6\}$, and the winning region for player 1 is $\{v_0, v_1, v_2, v_3, v_7\}$; both sets partition positions.

Designing algorithms for the computation of winning regions in parity games is an important theoretical problem as the corresponding decision problem "is a position won by player 0?" is in UP and coUP [6] but not known to be in P. Designing such algorithms is also important for applications since determining winning regions in parity games is equivalent, in linear time and logarithmic space, to other important problems — we mention model checking formulas of the modal mu-calculus [3, 1].

Traditional approaches design algorithms that compute the exact winning regions of parity games, see e.g. the survey in [2]. These algorithms are then either revealed not to be in P — by constructing defeating input games — or it is presently not known whether they are in P. In this paper we take a complementary approach. Instead of designing an algorithm that computes exact winning regions but has complexity that is unknown or known not to be in P, we present a design pattern for algorithms that are in P by construction but compute only subsets of winning regions by partitioning the set of positions

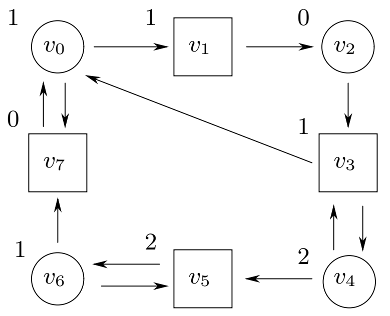


Figure 1: A parity game, taken from [4]. In circled (squared) positions player 0 (resp. 1) may move. Edges indicate possible game moves, and priorities of positions are provided to the left of positions, e.g. $\chi(v_6) = 1$.

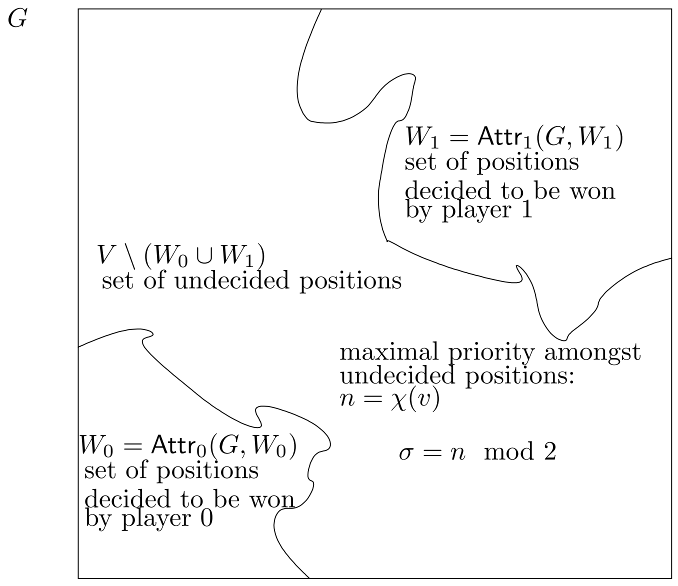
into three regions: wins of player 0 ($W_0 \subseteq V$), wins of player 1 ($W_1 \subseteq V$), and positions whose winning status is left open ($V \setminus (W_0 \cup W_1)$).

Our approach has practical relevance for at least two reasons. Firstly, in local model checking [8] one is only interested in whether a particular state satisfies a formula. This corresponds to determining which player wins a particular position, something our algorithms may well achieve. Secondly, our algorithms can be used as efficient preprocessors: due to the invariants satisfied by both computed regions (detailed in Sec. 2), one can abstract the original game so that each of these two regions collapses to a single winning position. Already existing algorithms, which are either known not to be in P or whose complexity is unknown, may then be used on those abstracted games to determine the winning status of remaining positions. In this extended abstract we refer to [1, 9] for basic concepts of parity games.

2 Design pattern. In its essence, the design pattern we propose is

```
W_0 = Attr_0(G,{});
W_1 = Attr_1(G,{});
cache = 0;
while (rank(G) != cache) {
  cache = rank(G);
  for (i=1; i++; i <= k) { A#i; }
}
return (W0,W1);
```

Throughout σ denotes 0 or 1, $\bar{\sigma}$ means $1-\sigma$, and $\mathsf{Attr}_\sigma(G, X)$ is the σ-attractor of set $X \subseteq V$ is G. The basic invariant for this pattern is, technically speaking, that each W_σ is a σ-paradise (as defined in [4]) and so a $\bar{\sigma}$-trap, and a σ-attractor in parity game G. Also, any increase in the size of W_σ decreases the rank of G. The final value of W_σ represents those positions that are known to be won by player σ, whereas $V \setminus (W_0 \cup W_1)$ contains those positions the algorithm does not classify. See Fig. 2 for illustration. This pattern is inspired by Zielonka's constructive proof of determinacy for parity games [9], except that our pattern is symmetric and non-recursive. Aspects `A#i` perform efficient static analyses that may decrease the rank of G, e.g. by increasing the size of W_σ.



Design pattern invariant: W_σ is σ-attractor, σ-paradise, and $\bar{\sigma}$-trap in G.

Figure 2: Computing safe winning regions of parity games: the set V of game positions is partitioned into regions W_0, W_1, and the complement of their union. Sets W_σ satisfy a basic invariant, ensuring that all positions in W_σ are winning for player σ.

3 Aspects. Aspect A_1 checks, for each position $v \in V \setminus (W_0 \cup W_1)$ with $2 \leq \chi(v)$ whether no cycle in (V, E) through v contains some position w with $\chi(w) = \chi(v) - 1$. If there is indeed no such cycle, one can decrement $\chi(v)$ to $\chi(v) - 2$ without changing the winning regions of G. Repeated application of this aspect gets rid of "gaps" in $\{\chi(v) \mid v \in V \setminus (W_0 \cup W_1)\}$, e.g. the missing 2 in $\{0, 1, 3\}$, whose absence may occur on-the-fly, would change $\{0, 1, 3\}$ to $\{0, 1\}$.

Aspect A_2, a dominance version of A_1, asks whether for any $v \in V \setminus (W_0 \cup W_1)$ all cycles through v have some $w \neq v$ with $\chi(v) \leq \chi(w)$. If so, A_2 re-sets $\chi(v)$ to 0 without changing the winning regions of G. Unlike its strict version, based on $\chi(v) < \chi(w)$, A_2 can resolve non-deterministic choices of such v.

Aspect A_3 abstracts intervals of odd (resp. even) priorities into an odd (resp. even) priority and solves, in P, a game for each such abstraction:

```
n := max { \chi(v) | v in V - (W_0 + W_1) }; // n as is Fig. 2
s := n mod 2;   // the \sigma in Fig. 2
Z_s = {};
Z_{1-s} = {};
for (k = 0; k++; 2*k <= n) {
  Z_s = checkInfFin({n,n-2,...,n-2*k}, {n-1,n-3,...,n-2*k+1}, s);
  if (Z_s != {}) { W_s := Attr_s(G,W_s + Z_s); }
  Z_{1-s} = checkInfFin({n-1,n-3,...,n-2*k-1}, {n,n-2,...,n-2*k}, 1-s);
  if (Z_{1-s} != {}) { W_{1-s} := Attr_{1-s}(G,W_{1-s} + Z_{1-s}); }
}
```

where `+` denotes set union, `{n,..,n-1}` is interpreted as {}, and the method `checkInfFin(I,F,p)` returns those positions in $V \setminus (W_0 \cup W_1)$ for which player $p \in \{0,1\}$ can win all plays in the sub-game $G[V \setminus W_p]$ such that "priorities set I is met infinitely often, and priorities set F is met only finitely often".

Finally aspect A_4 checks whether all cycles C through v have some w_C with $\chi(v) < \chi(w_C)$; if so, A_4 increases $\chi(v)$ to $\min_C \chi(w_C)$.

4 Interaction. By abuse of notation, we write $A_{i_1}A_{i_2}\ldots A_{i_k}$ for the algorithm that instantiates our design pattern with each `A#i` being A_{i_k}. In particular, `A#i` may equal `A#j` if $i \neq j$. We write $\{A_{i_1}A_{i_2}\ldots A_{i_k}\}$ if we mean any of the $k!$ algorithms obtained by permutations of these aspects A_{i_j}.

Fundamental questions are whether the interaction of these aspects commutes, whether swapping two aspects is in some sense confluent, and whether certain aspects can't aid the progress of certain others. These questions are similar to the "phase ordering problem" in the design of optimizing compilers.

For each instantiation, a suitable rank function has to be determined. A rank function for A_3 is $\mathsf{rank}(G) = |V \setminus (W_0 \cup W_1)|$. A rank function for $\{A_1A_3\}$, $\{A_2A_3\}$, and $\{A_1A_2A_3\}$ is $\mathsf{rank}(G) = |V \setminus (W_0 \cup W_1)| + \sum_{v \in V\setminus(W_0 \cup W_1)} \chi(v)$. It is less clear whether a good rank function exists for $\{A_1A_2A_3A_4\}$ as A_4 increases the value of the previously mentioned rank function.

Example 1. Algorithm A_3 completely solves the parity game in Fig. 1 but also $H_{4,3}$ as defined in [7], an exponential worst-case input for the algorithm in [7].

The interaction of aspects can improve the precision of algorithms derived from our design pattern. We illustrate this point by means of a simple example.

Example 2. Let $V = \{v_i \mid 0 \leq i \leq 3\}$ with $\chi(v_i) = i$ for all i, where player 0 (resp. 1) may move at v_0 and v_1 (resp. v_2 and v_3). For $E = \{(v_0,v_1),(v_0,v_3),(v_1,v_0),(v_1,v_3),(v_2,v_0),(v_3,v_2)\}$ player 1 has no choice, A_3 computes empty sets W_σ, whereas $\{A_2A_3\}$ completely solves this parity game: A_2 re-sets $\chi(v_2)$ from 2 to 0 and then `checkInfFin({3,1}, {2}, 1)` determines within A_3 that player 1 wins all positions.

The full paper will feature additional aspects not presented in this extended abstract, give a systematic account of the interaction potential of all of these aspects, and make some recommendations for instantiations of our design pattern. These recommendations will be backed up by experimental results that allow us to compare this approach to that of reducing parity games to SAT [5].

References

[1] E. Grädel, W. Thomas, and T. Wilke (Eds.). Automata, Logics, and Infinite Games — A Guide to Current Research, LNCS 2500, 385 pages. Springer Verlag, October 2002.

[2] H. Klauck. Algorithms for Parity Games. In [1].

[3] D. Kozen. Results on the propositional μ-calculus. *TCS* 27:333–354, 1983.

[4] R. Küsters. Memoryless Determinacy of Parity Games. In [1].

[5] M. Lange. Solving Parity Games by a Reduction to SAT. In *Proc. of the Workshop on Games in Design and Verification*, GDV'05, Edinburgh, Scotland, UK, 2005.

[6] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[7] M. Jurdziński. Small progress measures for solving parity games. In *Proc. of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 1770, pp. 290–301, 2000.

[8] C. Stirling and D. Walker. Local Model Checking in the Modal Mu-Calculus. *TCS* 89(1): 161-177, 1991.

[9] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS* 200:135-183, 1998.