Alpaga: A Tool for Solving Parity Games with Imperfect Information

Dietmar Berwanger[1], Krishnendu Chatterjee[2], Martin De Wulf[3], Laurent Doyen[3,4], and Thomas A. Henzinger[4]

[1] LSV, ENS Cachan and CNRS, France
[2] CE, University of California, Santa Cruz, USA
[3] Université Libre de Bruxelles (ULB), Belgium
[4] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract. Alpaga is a solver for two-player parity games with imperfect information. Given the description of a game, it determines whether the first player can ensure to win and, if so, it constructs a winning strategy. The tool provides a symbolic implementation of a recent algorithm based on antichains.

1 Introduction

Alpaga is a tool for solving parity games with imperfect information. These are turn-based games played on a graph by two players, one of them having imperfect information about the current state of the play. We consider objectives over infinite paths specified by parity conditions that can express safety, reachability, liveness, fairness, and most properties commonly used in verification. Given the description of a game, the tool determines whether the imperfect information player has a winning strategy for the parity objective and, if this is the case, it constructs such a winning strategy.

The Alpaga implementation is based on a recent technique using *antichains* for solving games with imperfect information efficiently [3], and for representing the strategies compactly [2]. To the best of our knowledge, this is the first implementation of a tool for solving parity games with imperfect information.

In this paper, we outline the antichain technique which is based on fixed-point computations using a compact representation of sets. Our algorithm essentially iterates a *controllable predecessor* operator that returns the states from which a player can force the play into a given target set in one round. For computing this operator, no polynomial algorithms is known. We propose a new symbolic implementation based on BDDs to avoid a naive enumerative procedure.

Imperfect-information games arise in several key applications related to verification and synthesis of reactive systems, such as (a) synthesis of controllers for plants with unobservable transitions; (b) distributed synthesis of processes with private variables not visible to other processes; (c) synthesis of robust controllers; (d) synthesis of automata specifications where only observations of automata are visible, and (e) the decision and simulation problem of quantitative specification languages; (f) model-checking secrecy and information flow. We believe that the tool Alpaga will make imperfect information games a useful framework for designers in the above applications.

S. Kowalewski and A. Philippou (Eds.): TACAS 2009, LNCS 5505, pp. 58–61, 2009.

An example of distributed-system synthesis has been solved with the tool. We have considered the design of a mutual-exclusion protocol for two processes. The tool Alpaga was able to synthesize a winning strategy for a requirement of mutual exclusion and starvation freedom which corresponds to Peterson's protocol. Details can be found in an extended version of this paper [1].

2 Games and Algorithms

Let Σ be a finite alphabet of actions and let Γ be a finite alphabet of observations. A *game structure with imperfect information* over Σ and Γ is a tuple $G = (L, l_0, \Delta, \gamma)$, where

- L is a finite set of locations (or states), $l_0 \in L$ is the initial location;
- $\Delta \subseteq L \times \Sigma \times L$ is a set of labelled transitions such that for all $\ell \in L$ and all $a \in \Sigma$, there exists $\ell' \in L$ such that $(\ell, a, \ell') \in \Delta$, i.e., the transition relation is total;
- $\gamma : \Gamma \to 2^L \setminus \emptyset$ is an observability function that maps each observation to a set of locations such that the set $\{\gamma(o) \mid o \in \Gamma\}$ partitions L. For each $\ell \in L$, let $\mathsf{obs}(\ell) = o$ be the unique observation such that $\ell \in \gamma(o)$.

The game on G is played in rounds. Initially, a token is placed in location l_0. In every round, Player 1 first chooses an action $a \in \Sigma$, and then Player 2 moves the token to an a-successor ℓ' of the current location ℓ, i.e., such that $(\ell, a, \ell') \in \Delta$. Player 1 does not see the current location ℓ of the token, but only the observation $\mathsf{obs}(\ell)$ associated to it. A *strategy* for Player 1 in G is a function $\alpha : \Gamma^+ \to \Sigma$. The set of possible *outcomes* of α in G is the set $\mathsf{Outcome}(G, \alpha)$ of sequences $\pi = \ell_1 \ell_2 \dots$ such that $\ell_1 = l_0$ and $(\ell_i, \alpha(\mathsf{obs}(\ell_1 \dots \ell_i)), \ell_{i+1}) \in \Delta$ for all $i \geq 1$. A *visible parity condition* on G is defined by a function $p : \Gamma \to \mathbb{N}$ that maps each observation to a non-negative integer priority. We say that a strategy α for Player 1 is *winning* if for all $\pi \in \mathsf{Outcome}(G, \alpha)$, the least priority that appears infinitely often in π is even.

To decide whether Player 1 is winning in a game G, the basic approach consists in tracing the *knowledge* of Player 1, represented a set of locations called a *cell*. The initial knowledge is the cell $s_0 = \{l_0\}$. After each round, the knowledge s of Player 1 is updated according to the action a she played and the observation o she receives, to $s' = \mathsf{post}_a(s) \cap \gamma(o)$ where $\mathsf{post}_a(s) = \{\ell' \in L \mid \exists \ell \in s : (\ell, a, \ell') \in \Delta\}$.

Antichain algorithm. The antichain algorithm is based on the *controllable predecessor* operator $\mathsf{CPre} : 2^S \to 2^S$ which, given a set of cells q, computes the set of cells q' from which Player 1 can force the game into a cell of q in one round:

$$\mathsf{CPre}(q) = \{s \subseteq L \mid \exists a \in \Sigma \cdot \forall o \in \Gamma : \mathsf{post}_a(s) \cap \gamma(o) \in q\}. \tag{1}$$

The key of the algorithm relies on the fact that $\mathsf{CPre}(\cdot)$ preserves downward-closedness. A set q of cells is *downward-closed* if, for all $s \in q$, every subset $s' \subseteq s$ is also in q. Downward-closed sets q can be represented succinctly by their maximal elements $r = \lceil q \rceil = \{s \in q \mid \forall s' \in q : s \not\subset s'\}$, which form an *antichain*. With this representation, the controllable predecessor operator is defined by

$$\mathsf{CPre}(r) = \left\lceil \{s \subseteq L \mid \exists a \in \Sigma \cdot \forall o \in \Gamma \cdot \exists s' \in r : \mathsf{post}_a(s) \cap \gamma(o) \subseteq s'\} \right\rceil. \tag{2}$$

Strategy construction. The implementation of the strategy construction is based on [2]. The algorithm of [2] employs antichains to compute winning strategies for imperfect-information parity games in an efficient and compact way: the procedure is similar to the classical algorithm of McNaughton [4] and Zielonka [5] for perfect-information parity games, but, to preserve downwards closure, it avoids the complementation operation of the classical algorithms by recurring into subgames with an objective obtained as a boolean combination of reachability, safety, and reduced parity objectives.

Strategy simplification. A strategy in a game with imperfect information can be represented by a set $\Pi = \{(s_1, \mathsf{rank}_1, a_1), \ldots, (s_n, \mathsf{rank}_n, a_n)\}$ of triples $(s_i, \mathsf{rank}_i, a_i) \in 2^L \times \mathbb{N} \times \Sigma$ where s_i is a cell, and a_i is an action. Such a triple assigns action a_i to every cell $s \subseteq s_i$; since a cell s may be contained in many s_i, we take the triple with minimal value of rank_i. Formally, given the current knowledge s of Player 1, let $(s_i, \mathsf{rank}_i, a_i)$ be a triple with minimal rank in Π such that $s \subseteq s_i$ (such a triple exists if s is a winning cell); the strategy represented by Π plays the action a_i in s.

Our implementation applies the following rules to simplify the strategies and obtain a compact representation of winning strategies in parity games with imperfect information.

(*Rule* 1) In a strategy Π, retain only elements that are maximal with respect to the following order: $(s, \mathsf{rank}, a) \succeq (s', \mathsf{rank}', a')$ if $\mathsf{rank} \leq \mathsf{rank}'$ and $s' \subseteq s$. Intuitively, the rule specifies that we can delete (s', rank', a') whenever all cells contained in s' are also contained in s; since $\mathsf{rank} \leq \mathsf{rank}'$, the strategy can always choose (s, rank, a) and play a.

(*Rule* 2) In a strategy Π, delete all triples $(s_i, \mathsf{rank}_i, a_i)$ such that there exists $(s_j, \mathsf{rank}_j, a_j) \in \Pi$ $(i \neq j)$ with $a_i = a_j$, $s_i \subseteq s_j$ (and hence $\mathsf{rank}_i < \mathsf{rank}_j$ by Rule 1), such that for all $(s_k, \mathsf{rank}_k, a_k) \in \Pi$, if $\mathsf{rank}_i \leq \mathsf{rank}_k < \mathsf{rank}_j$ and $s_i \cap s_k \neq \emptyset$, then $a_i = a_k$. Intuitively, the rule specifies that we can delete $(s_i, \mathsf{rank}_i, a_i)$ whenever all cells contained in s_i are also contained in s_j, and the action a_j is the same as the action a_i. Moreover, if a cell $s \subseteq s_i$ is also contained in s_k with $\mathsf{rank}_i \leq \mathsf{rank}_k < \mathsf{rank}_j$, then the action played by the strategy is also $a_k = a_i = a_j$.

3 Implementation

Computing $\mathsf{CPre}(\cdot)$ is likely to require time exponential in the number of observations (a natural decision problem involving $\mathsf{CPre}(\cdot)$ is NP-hard [2]). Therefore, it is natural to let the BDD machinery evaluate the universal quantification over observations in (2). We present a BDD-based algorithm to compute $\mathsf{CPre}(\cdot)$.

Let $L = \{\ell_1, \ldots, \ell_n\}$ be the state space of the game G. A cell $s \subseteq L$ can be represented by a valuation v of the boolean variables $\bar{x} = x_1, \ldots, x_n$ such that, for all $1 \leq i \leq n$, $\ell_i \in s$ iff $v(x_i) = \mathsf{true}$. A BDD over $x_1, \ldots, x_n$ is called a *linear encoding*, it encodes a set of cells. A cell $s \subseteq L$ can also be represented by a BDD over boolean variables $\bar{y} = y_1, \ldots, y_m$ with $m = \lceil \log_2 n \rceil$. This is called a *logarithmic encoding*, it encodes a single cell.

We represent the transition relation of G by the $n \cdot |\Sigma|$ BDDs $T_a(\ell_i)$ $(a \in \Sigma, 1 \leq i \leq n)$ with logarithmic encoding over $\bar{y}$. So, $T_a(\ell_i)$ represents the set $\{\ell_j \mid (\ell_i, a, \ell_j) \in$

$\Delta\}$. The observations $\Gamma = \{o_1, \ldots, o_p\}$ are encoded by $\lceil \log_2 p \rceil$ boolean variables $b_0, b_1, \ldots$ in the BDD B_Γ defined by

$$B_\Gamma \equiv \bigwedge_{0 \leq j \leq p-1} \bar{b} = [j]_2 \rightarrow C_{j+1}(\bar{y}),$$

where $[j]_2$ is the binary encoding of j and $C_1, \ldots, C_p$ are BDDs that represent the sets $\gamma(o_1), \ldots, \gamma(o_p)$ in logarithmic encoding.

Given the antichain $q = \{s_1, \ldots, s_t\}$, let S_k $(1 \leq k \leq t)$ be the BDDs that encode the set s_k in logarithmic encoding over $\bar{y}$. For each $a \in \Sigma$, we compute the BDD CP_a in linear encoding over $\bar{x}$ as follows:

$$\mathsf{CP}_a \equiv \forall \bar{b} \cdot \bigvee_{1 \leq k \leq t} \bigwedge_{1 \leq i \leq n} x_i \rightarrow \big[\forall \bar{y} \cdot \big(T_a(\ell_i) \wedge B_\Gamma\big) \rightarrow S_k\big].$$

Then, we define $\mathsf{CP} \equiv \bigvee_{a \in \Sigma} \mathsf{CP}_a(q)$, and we extract the maximal elements in $\mathsf{CP}(\bar{x})$ as follows, with ω a BDD that encodes the relation of (strict) set inclusion $\subset$:

$$\omega(\bar{x}, \bar{x}') \equiv \Big(\bigwedge_{i=1}^{n} x_i \rightarrow x_i'\Big) \wedge \Big(\bigvee_{i=1}^{n} x_i \neq x_i'\Big),$$

$$\mathsf{CP}^{\min}(\bar{x}) \equiv \mathsf{CP}(\bar{x}) \wedge \neg \exists \bar{x}' \cdot \omega(\bar{x}, \bar{x}') \wedge \mathsf{CP}(\bar{x}').$$

Finally, we construct the antichain $\mathsf{CPre}(q)$ as the following set of BDDs in logarithmic encoding: $\mathsf{CPre}(q) = \{s \mid \exists v \in \mathsf{CP}^{\min} : s = \{\ell_i \mid v(x_i) = \mathsf{true}\}\}$.

Features of the tool. The input of the tool is a file describing the transitions and observations of the game graph. The output is the set of maximal winning cells, and a winning strategy in compact representation. We have also implemented a simulator to let the user play against the strategy computed by the tool. The user has to provide an observation in each round (or may let the tool choose one randomly). The web page of the tool is `http://www.antichains.be/alpaga`. We provide the source code, the executable, an online demo, and several examples.

References

1. Berwanger, D., Chatterjee, K., De Wulf, M., Doyen, L., Henzinger, T.A.: Alpaga: A tool for solving parity games with imperfect information. Technical Report MTC-REPORT-2008-007, EPFL (2008), `http://infoscience.epfl.ch/record/130681`
2. Berwanger, D., Chatterjee, K., Doyen, L., Henzinger, T.A., Raje, S.: Strategy construction for parity games with imperfect information. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 325–339. Springer, Heidelberg (2008)
3. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Algorithms for omega-regular games of incomplete information. Logical Methods in Computer Science 3(3:4) (2007)
4. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65(2), 149–184 (1993)
5. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200, 135–183 (1998)