Strategy Construction for Parity Games with Imperfect Information[☆,☆☆]

Dietmar Berwanger[a], Krishnendu Chatterjee[b], Martin De Wulf[c], Laurent Doyen[*,a], Thomas A. Henzinger[b,d]

[a]*LSV, CNRS and ENS Cachan, France*
[b]*IST Austria (Institute of Science and Technology Austria)*
[c]*Université Libre de Bruxelles (ULB), Belgium*
[d]*École Polytechnique Fédérale de Lausanne (EPFL), Switzerland*

Abstract

We consider two-player parity games with imperfect information in which strategies rely on observations that provide imperfect information about the history of a play. To solve such games, i.e., to determine the winning regions of players and corresponding winning strategies, one can use the subset construction to build an equivalent perfect-information game. Recently, an algorithm that avoids the inefficient subset construction has been proposed. The algorithm performs a fixed-point computation in a lattice of antichains, thus maintaining a succinct representation of state sets. However, this representation does not allow to recover winning strategies.

In this paper, we build on the antichain approach to develop an algorithm for constructing the winning strategies in parity games of imperfect information. One major obstacle in adapting the classical procedure is that the complementation of attractor sets would break the invariant of downward-closedness on which the antichain representation relies. We overcome this difficulty by decomposing problem instances recursively into games with a combination of reachability, safety, and simpler parity conditions. We also report on an experimental implementation of our algorithm; to our knowledge, this is the first implementation of a procedure for solving imperfect-information parity games on graphs.

[☆] A preliminary version of this paper appeared in the *Proceedings of the 19th International Conference on Concurrency Theory* (CONCUR), Lecture Notes in Computer Science 5201, Springer-Verlag, 2008, pages 325-339.

[☆☆] This research was supported in part by the NSF grants CCR-0132780, CNS-0720884, and CCR-0225610, by the Swiss National Science Foundation, by the Deutsche Forschungsgemeinschaft (DFG), by the European projects Combest, Gasics, Lint, and Quasimodo, by the Belgian PAI program Moves and by the Belgian Federated Center in Verification (CFV) funded by the F.R.S.-FNRS. We thank three anonymous referees for their useful comments.

[*] Corresponding author: Laurent Doyen, LSV, CNRS UMR 8643 & ENS Cachan, 61 avenue du Président Wilson, 94235 Cachan Cedex, France.

Email addresses: dwb@lsv.ens-cachan.fr (Dietmar Berwanger), Krishnendu.Chatterjee@ist.ac.at (Krishnendu Chatterjee), madewulf@gmail.com (Martin De Wulf), doyen@lsv.ens-cachan.fr (Laurent Doyen), tah@epfl.ch (Thomas A. Henzinger)

1. Introduction

Parity games capture the algorithmic essence of fundamental problems in state-based system analysis [1]. They arise as natural evaluation games for the μ-calculus, an expressive logic that subsumes most specification formalisms for reactive systems, and they are closely related to alternating ω-automata [2].

In the basic variant, a parity game is played on a finite graph with nodes labelled by natural numbers denoting *priorities*. There are two players, Player 1 and Player 2, who take turns in moving a token along the edges of the graph starting from a designated initial node. In a play, the players thus form an infinite path, and Player 1 wins if the least priority that is visited infinitely often is even; otherwise Player 2 wins. These are games of *perfect information*: during the play each of the players is informed about the current position of the token. One key property of parity games is memoryless determinacy: from every initial node, either Player 1 or Player 2 has a winning strategy that does not depend on the history of the play [3]. As a consequence, a winning strategy can be represented as a subset of the edges of the graph, and the problem of constructing a winning strategy is in NP $\cap$ coNP.

The perfect-information setting is often not sufficient in practice. The need to model uncertainty about the current state of a system arises in many situations. For instance in controller-synthesis applications, certain parameters of the plant under control may not be observable by the controller. Likewise in multi-component design, individual components of a complex system may have private variables invisible to other components. As a way to handle state-explosion problems, one may accept a loss of information in a concrete model in order to obtain a manageable abstract model with imperfect information.

One fundamental question is how to model *imperfect information*. In the classical theory of extensive games, this is done by partitioning the game tree into information sets signifying that a player cannot distinguish between different decision nodes of the same information set [4]. Technically, this corresponds to restricting the set of strategies available to a player by requiring a uniform choice across all nodes of an information set. However, for the algorithmic analysis of games of infinite duration on graphs, the information sets need to be finitely represented. Such a model is obtained by restricting to strategies that rely on observations corresponding to a partitioning of the game graph.

The model of imperfect information games that we consider here was originally introduced in [5]. Like in the perfect-information case, the game is played by two opposing players on a finite graph. The nodes of the graph, called *locations*, are partitioned into information sets indexed by *observations*. Intuitively, the only visible information available to Player 1 during a play is the observation corresponding to the current location, whereas Player 2 has perfect information about the current location of the game. This is a natural model for controller synthesis, where the controller does not have access to the private variables of the plant, and the control strategy needs to be winning against arbitrary behavior of the plant, which corresponds to give the full-power of perfect-information to Player 2. It can be formally shown that the existence of a winning strategy for Player 1 does not depend on the ability or not for Player 2 to see the exact position of the game [6]. The structure of the graph (including the starting location) is known to both players, and the parity winning condition is defined in terms of priorities assigned to observations. Therefore, the winning condition is itself visible, which is natural in the context of controller synthesis; it also enables simpler algorithmic solutions. Games with non-observable winning conditions require more involved techniques, as witnessed by the fact that the universality problem for nondeterministic Büchi automata can be reduced to them.

The basic algorithmic problems about parity games are (1) to determine the *winning region* of a player, that is, the set of initial locations from which he has a winning strategy, and (2) to construct such a *winning strategy*. One straightforward way to solve parity games of imperfect information is based on the following idea [5, 6]: after an initial prefix of a play, Player 1 may not know in which precise location the play currently is but, by keeping track of the history, he can identify a minimal set of locations that is guaranteed to contain the current location. Such a set, to which we refer as a *cell*, reflects the knowledge derived by a player from past play. Via a subset construction that associates moves in the game to transitions between cells, the original imperfect-information game over locations is transformed into an equivalent game with perfect information over cells. This approach, however, incurs an exponential increase in the number of states and is therefore inefficient.

For computing the winning region of a game, an algorithm that avoids the explicit subset construction has been proposed recently in [6]. The algorithm exploits a monotonicity property of imperfect-information games: if a cell is winning for Player 1, that is, if he wins from every location of the cell, then he also wins from every subset of the cell. Intuitively, the subcell represents more precise knowledge than the entire cell. It is therefore sufficient to manipulate sets of cells that are *downward-closed* in the sense that, if a cell belongs to the set, then all its subcells also belong to it. As a succinct representation for downward-closed sets of cells, the algorithm maintains antichains that consist of maximal elements in the powerset lattice of cells. The winning region can now be computed symbolically by evaluating its characterization as a μ-calculus formula over the lattice. One particular effect of this procedure is that the discovery of winning cells propagates backwards, rather than forwards from the initial location, and thus avoids the construction and exploration of cells that are not relevant for solving the game.

On many instances, the antichain algorithm performs significantly better than the subset construction for computing winning regions. However, in contrast to the latter, the antichain algorithm does not construct winning strategies. Indeed, we argue that there is no direct way to extract a winning strategy from the symbolic fixed-point computation. In terms of logic, the algorithm evaluates a μ-calculus formula describing the winning region, which corresponds to evaluating a monadic expression with second-order quantifiers that range over (sets of) nodes in the game graph. On the other hand, strategies are not monadic objects; already memoryless location- or observation-based strategies are composed of binary objects, namely, edges of the graph or pairs of cells. In particular, we show that already in parity games with perfect information knowing the winning region of a game does not make the problem of constructing a winning strategy easier. In imperfect-information games there are additional sources of complexity: the size of a winning strategy may be exponentially larger than the winning region, already for reachability objectives. Nevertheless, the construction of winning strategies is crucial for many applications such as controller synthesis or counterexample-guided abstraction-refinement [7].

In this paper, we present an algorithm for constructing winning strategies in parity games with imperfect information. One main concern is to avoid the subset construction. To accomplish this, our algorithm builds on the antichain technique and works with symbolic representations of sets of cells. It generalizes a fundamental procedure for solving parity games proposed by McNaughton [8] and presented in detail by Zielonka [9]. The procedure works recursively, taking the viewpoint of Player 1: in each stage a smaller game is obtained by removing the attractor region from which Player 2 can ensure to reach the minimal odd priority. This operation of removal marks the main difficulty in adapting the algorithm to antichains, as the residual subgame is in general not downward closed. Intuitively, switching between the sides of the two players breaks the succinct representation. We overcome this difficulty by letting Player 1 simulate Player 2, in a certain sense. Technically, this amounts to replacing two alternating reachability computations by the computation of a strategy that simultaneously satisfies a reachability and a safety objective.

We have implemented the algorithm in a prototype called Alpaga[1], based on a fixed-point computation that essentially iterates a *controllable predecessor* operator returning the states from which a player can force the play into a given target set in one round. No polynomial algorithm for computing this operator is known. In order to avoid the naive enumerative procedure, we propose a new symbolic implementation based on BDDs. To our knowledge, this is the first automatic tool for solving imperfect-information parity games on graphs.

We give two examples of distributed-system synthesis solved with Alpaga. First, we illustrate the need for imperfect information in the games that arise in the synthesis problem by considering a simple lock-based program. The specification requires that the lock is never acquired or released twice in a row, even if the status of the lock is not visible to the program. We also consider the design of a mutual-exclusion protocol for two processes. Using Alpaga, we have synthesized a winning strategy for a requirement of mutual exclusion and starvation freedom which corresponds to Peterson's protocol.

[1] The tool Alpaga is available at http://www.antichains.be/alpaga/.

2. Definitions

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, and $\gamma : \Gamma \to 2^L \setminus \emptyset$ is an observability function that maps each observation to a set of locations. Abusing notation, we usually identify the set $\gamma(o)$ with the observation symbol o. We require the following two conditions on G: (i) for all $\ell \in L$ and all $\sigma \in \Sigma$, there exists $\ell' \in L$ such that $(\ell, \sigma, \ell') \in \Delta$, i.e., the transition relation is total, and (ii) the set $\{\gamma(o) \mid o \in \Gamma\}$ partitions L. For each $\ell \in L$, let $\mathrm{obs}(\ell) = o$ be the unique observation such that $\ell \in \gamma(o)$. In the special case where $\Gamma = L$ and $\mathrm{obs}(\ell) = \ell$, for all $\ell \in L$, we say that G is a game structure of *perfect information* over Σ. For infinite sequences of locations $\pi = \ell_1 \ell_2 \dots$, we define $\mathrm{obs}(\pi) = o_1 o_2 \dots$ where $\mathrm{obs}(\ell_i) = o_i$ for all $i \geq 1$, and similarly for finite sequences of locations. For $\sigma \in \Sigma$ and $s \subseteq L$, we define $\mathsf{post}_\sigma(s) = \{\ell' \in L \mid \exists \ell \in s : (\ell, \sigma, \ell') \in \Delta\}$ as the set of σ-successors of locations in s.

The game on G is played in rounds. In each round, Player 1 chooses an action $\sigma \in \Sigma$, and Player 2 chooses a successor ℓ' of the current location ℓ such that $(\ell, \sigma, \ell') \in \Delta$. A *play* in G is an infinite sequence $\pi = \ell_1 \ell_2 \dots$ of locations such that (i) $\ell_1 = l_0$, and (ii) for all $i \geq 0$, there exists $\sigma_i \in \Sigma$ such that $(\ell_i, \sigma_i, \ell_{i+1}) \in \Delta$.

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 $\mathrm{Outcome}(G, \alpha)$ of plays $\pi = \ell_1 \ell_2 \dots$ such that $(\ell_i, \alpha(\mathrm{obs}(\ell_1 \dots \ell_i)), \ell_{i+1}) \in \Delta$ for all $i \geq 1$. We say that a strategy α is *memoryless* if $\alpha(\rho \cdot o) = \alpha(\rho' \cdot o)$ for all $\rho, \rho' \in \Gamma^*$. We say that a strategy uses *finite memory* if it can be represented by a finite-state deterministic *transducer* $(M, m_0, \lambda, \delta)$ over a finite set of states M (the memory of the strategy) with an initial state $m_0 \in M$, where $\lambda : M \to \Sigma$ is a labelling of states with actions, and $\delta : M \times \Gamma \to M$ is a transition function. In state m, the strategy recommends the action $\lambda(m)$, and when Player 2 chooses a location with observation o, it updates the internal state to $\delta(m, o)$. Formally, $(M, m_0, \lambda, \delta)$ defines the strategy α such that $\alpha(\rho) = \lambda(\hat{\delta}(m_0, \rho))$, for all $\rho \in \Gamma^+$, where $\hat{\delta}$ extends δ to sequences of observations in the usual way. The *size* of a finite-state strategy represented by such a transducer is the number $|M|$ of its states. Note that we do not need to consider randomized (or mixed) stratregies because pure strategies are sufficient to win a game [6].

An *objective* for a game structure $G = (L, l_0, \Delta, \gamma)$ is a set $\phi \subseteq \Gamma^\omega$ of infinite sequences of observations. A strategy α for Player 1 is *winning* for an objective ϕ if $\mathrm{obs}(\pi) \in \phi$ for all $\pi \in \mathrm{Outcome}(G, \alpha)$. We say that set of locations $s \subseteq L$ is *winning* for ϕ if there exists a strategy α for Player 1 such that α is winning for ϕ in $G_\ell := (L, \ell, \Delta, \gamma)$ for all $\ell \in s$. A *game* is a pair (G, ϕ) consisting of a game structure and a matching objective. We say that Player 1 wins the game, if he has a winning strategy for the objective ϕ.

We consider the following classical objectives. Given a target set $\mathcal{T} \subseteq \Gamma$ of observations, the *safety* objective $\mathsf{Safe}(\mathcal{T}) = \{o_1 o_2 \dots \mid \forall i \geq 1 : o_i \in \mathcal{T}\}$ requires that the play remain within the set $\mathcal{T}$. Dually, the *reachability* objective $\mathsf{Reach}(\mathcal{T}) = \{o_1 o_2 \ldots \mid \exists i \geq 1 : o_i \in \mathcal{T}\}$ requires that the play visit the set $\mathcal{T}$ at least once. The *Büchi* objective $\mathsf{Buchi}(\mathcal{T}) = \{o_1 o_2 \ldots \mid \forall i \cdot \exists j \geq i : o_j \in \mathcal{T}\}$ requires that an observation in $\mathcal{T}$ occur infinitely often. Dually, the *coBüchi* objective $\mathsf{coBuchi}(\mathcal{T}) = \{o_1 o_2 \ldots \mid \exists i \cdot \forall j \geq i : o_j \in \mathcal{T}\}$ requires that only observations in $\mathcal{T}$ occur infinitely often. Finally, given a *priority function* $p : \Gamma \to \mathbb{N}$ that maps each observation to a non-negative integer priority, the *parity* objective $\mathsf{Parity}(p)$ requires that the minimum priority that appears infinitely often be even. Formally, $\mathsf{Parity}(p) = \{o_1 o_2 \ldots \mid \min\{p(o) \mid \forall i \cdot \exists j \geq i : o = o_j\}$ is even$\}$. We denote by $\mathsf{coParity}(p)$ the objective complementary to $\mathsf{Parity}(p)$, that is, $\mathsf{coParity}(p) = \{o_1 o_2 \ldots \mid \min\{p(o) \mid \forall i \cdot \exists j \geq i : o = o_i\}$ is odd$\}$. Parity objectives are a canonical form to express all ω-regular objectives [10]. In particular, they subsume safety, reachability, Büchi, and coBüchi objectives.

Notice that objectives are defined as sets of sequences of observations, they are thus visible to Player 1. A game with a safety (or reachability) objective defined via a set of target states rather than observations can be transformed into an equivalent game with a visible safety (or reachability) objective in polynomial time, by simply making the target states observable.

3. Antichain Algorithm

Let Σ be an alphabet of actions and let Γ be an alphabet of observations. We consider the problem of deciding, given a game structure $G = (L, l_0, \Delta, \gamma)$ and a parity objective ϕ, whether Player 1 has a winning strategy for ϕ in G. If the answer is YES, we ask to construct such a winning strategy. This problem is known to be EXPTIME-complete already for reachability objectives [5, 6]. The basic algorithm proposed in [5] constructs a game (G^{K}, ϕ') such that

(i) $G^{\mathsf{K}} = (S, s_0, \Delta', \gamma')$ is a game structure of *perfect information* over the action alphabet Σ, and
(ii) Player 1 has a winning strategy for ϕ in G if and only if Player 1 has a winning strategy for ϕ' in G^{K}.

The game structure G^{K} is obtained via a subset construction where $S = 2^L \setminus \{\emptyset\}$ and $(s_1, \sigma, s_2) \in \Delta'$ if and only if there exists an observation $o \in \Gamma$ such that $s_2 = \mathsf{post}_\sigma(s_1) \cap \gamma(o)$ and $s_2 \neq \emptyset$. In the sequel, we call a set $s \subseteq L$ a *cell*. A cell summarizes the current *knowledge* of Player 1, i.e., the set of possible locations in which the game G can be after the sequence of observations seen by Player 1. Every cell reachable in G^{K} is a subset of some observation and, accordingly, the parity objective ϕ' is defined by extending the priority function that defines ϕ in a natural way to cells. Notice that an objective for G^{K} is a set of infinite sequences of cells, as locations and observations coincide in games with perfect information. In (G^{K}, ϕ'), memoryless winning strategies always exist and they can be converted into winning strategies in (G, ϕ) which depend only on the current cell in G^{K}. Intuitively, there is a one-to-one correspondence between plays π^{K} in G^{K} and sequences of observations $\mathrm{obs}(\pi)$ of plays in G. Since the strategies in G are functions of the observations only, this correspondence can be extended to strategies in G^{K} and in G [6, 11].

Due to the explicit construction of G^{K}, this approach involves an exponential blow-up of the original game structure.

In [6], an alternative algorithm is proposed for solving games with imperfect information. Winning cells are computed symbolically, avoiding the exponential subset construction. The 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. Formally,

$$\mathsf{CPre}(q) = \{s \in S \mid \exists \sigma \in \Sigma \cdot \forall s' : \text{ if } (s, \sigma, s') \in \Delta' \text{ then } s' \in q\}. \tag{1}$$

The key of the algorithm is that $\mathsf{CPre}(\cdot)$ preserves downward-closedness, which intuitively means that, if Player 1 has a strategy from s to force the game to be in q in the next round, then he also has such a strategy from all $s' \subseteq s$ because then Player 1 has a more precise knowledge in s' than in s. Formally, a set q of cells is *downward-closed* if $s \in q$ implies $s' \in q$, for all $s' \subseteq s$. If q is downward-closed, then so is $\mathsf{CPre}(q)$. Since parity games can be solved by evaluating a μ-calculus formula over the powerset lattice $(S, \subseteq, \cup, \cap)$, and because $\mathsf{CPre}(\cdot)$, $\cap$, and $\cup$ preserve downward-closedness, it follows that a symbolic algorithm maintains only downward-closed sets q of cells and can therefore use a compact representation, namely their maximal elements $\lceil q \rceil = \{as \in q \mid s \neq \emptyset \text{ and } \forall s' \in q : s \not\subset s'\}$, forming *antichains* of cells, i.e., sets of $\subseteq$-incomparable cells. The set $\mathcal{A}$ of antichains is partially ordered by setting $q \sqsubseteq q'$ for $q, q' \in \mathcal{A}$ if and only if for all $s \in q$ there exists $s' \in q'$ such that $s \subseteq s'$. The least upper bound of two antichains $q, q' \in \mathcal{A}$ is $q \sqcup q' = \lceil \{s \mid s \in q \text{ or } s \in q'\} \rceil$, and their greatest lower bound is $q \sqcap q' = \lceil \{s \cap s' \mid s \in q \text{ and } s' \in q'\} \rceil$. The partially ordered set $(\mathcal{A}, \sqsubseteq, \sqcup, \sqcap)$ forms a complete lattice. We view antichains of location sets as a symbolic representation of $\subseteq$-downward-closed sets of cells.

The advantage of the symbolic antichain approach over the explicit subset construction has been demonstrated in practice for different applications in model-checking (*e.g.* [12, 13]). The following proposition shows that the antichain algorithm may be exponentially faster than the subset construction.

Proposition 1 (see also [12]). *There exists a family $(G_k)_{k \geq 2}$ of reachability games with imperfect information over k locations such that, on input G_k the subset-construction algorithm runs in time exponential in k whereas the antichain algorithm runs in time polynomial in k.*

Proof. Consider the family of games G_k over the alphabet $\Sigma = \{0, 1\}$ depicted in Figure 1. For any $k \geq 2$ the set L_k of locations of G_k consists of $2k + 1$ locations, $\ell_0, \ldots, \ell_k$ and $\ell'_1, \ldots, \ell'_k$, the initial location is ℓ_0. The observations are $\{\ell_k, \ell'_k\}$ and $L_k \setminus \{\ell_k, \ell'_k\}$, and the goal is to reach the set $T = \{\ell_k, \ell'_k\}$. Clearly, there exists a winning strategy in G_k for all $k \geq 2$, consisting in playing any $\{0, 1\}$-word starting with 1.

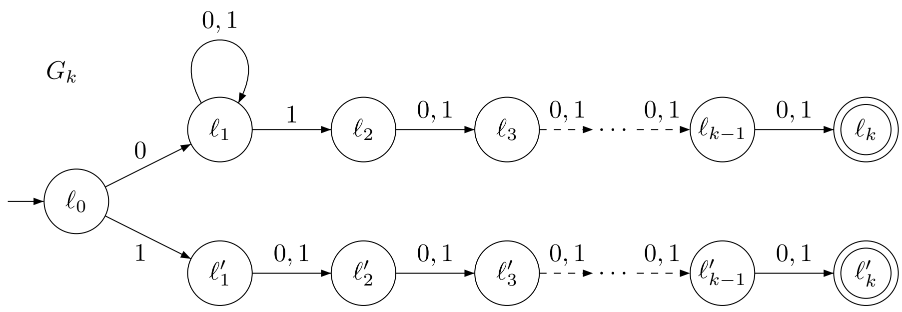

Figure 1: A family of games G_k, $k \geq 2$, for Proposition 1.

It is easy to see that the subset-construction algorithm encounters an exponential blow-up on G_k as there are $O(2^k)$ cells in the perfect-information version of the subgame $\{\ell_1, \dots, \ell_k\}$.

However, the antichain algorithm terminates in polynomial time, as the sequence defined by $q_0 = \{\{\ell_k, \ell'_k\}\}$, and $q_{i+1} = \mathsf{CPre}^{G_k}(q_i) \sqcup q_0$ for $i \geq 0$, stabilizes after k iterations with $q_i = \{\{\ell_{k-i}, \dots, \ell_k\} \cup \{\ell'_{k-i}, \dots, \ell'_k\}\}$ for $i < k$, $q_k = \{Q_k\}$, and $q_{k+1} = q_k$. ■

The antichain algorithm computes a compact representation of the set of winning cells. However, it does not produce a winning strategy. We point out that, already for parity games with perfect information, no polynomial-time algorithm is known to construct a winning strategy, even when the set of winning locations for each player is given. In fact, such an algorithm would show that parity games with perfect information can be solved in polynomial time, as the problem of verifying a winning partition for a game is as hard as solving the game itself.

Proposition 2. *The following two problems on parity games with perfect information are polynomial-time equivalent.*

(i) Given a game, decide whether Player 1 *is winning from the initial location.*
(ii) Given a game and a set W of locations, decide whether W is the set of all winning locations for Player 1.

Proof. Clearly, Problem (ii) can be solved by solving Problem (i) successively, for all locations.

For the reverse direction, consider an instance of problem (i) – a game G over a set L of locations with initial location l_0. We construct in polynomial time a game G' such that (G', L) is a positive instance of problem (ii) if, and only if, G is a positive instance of problem (i).

Without loss of generality, we may assume that no priority in G is less than 2. The game G' is obtained by adding to G a "reset" location z of priority 1, with

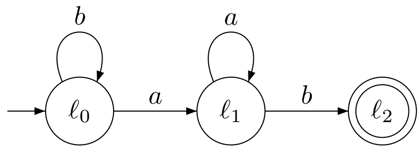

Figure 2: A reachability game G.

transitions that allow Player 1 to reach z from any location of G where he moves, and with one transition from z back to l_0. If Player 1 wins in G from l_0, then he will win in the new game from any location by first moving via z to l_0 and then following the winning strategy he has in G. Thus, G' together with the set of all locations is a positive instance of Problem (*ii*); obviously this can be constructed in polynomial time. Conversely, suppose Player 1 wins from every location in G', and let α be a memoryless winning strategy from l_0. No play starting from l_0 that follows α can reach z, otherwise Player 1 loses. Thus, the strategy α readily witnesses that Player 1 wins in the original game G from l_0. ■

We also argue that, in games with imperfect information, even for simple reachability objectives the antichain representation of the set of winning cells may not be sufficient to construct a winning strategy. Consider the game G depicted in Figure 2, with reachability objective $\mathsf{Reach}(\{\ell_2\})$. The observations are $\{\ell_0, \ell_1\}$ and $\{\ell_2\}$. Since $\mathsf{CPre}(\{\{\ell_2\}\}) = \{\{\ell_1\}\}$ (by playing action b) and $\mathsf{CPre}(\{\{\ell_1\}, \{\ell_2\}\}) = \{\{\ell_0, \ell_1\}\}$ (by playing action a), the fixed point computed by the antichain algorithm is $\{\{\ell_2\}, \{\ell_0, \ell_1\}\}$. However, from $\{\ell_0, \ell_1\}$, after playing a, Player 1 reaches the cell $\{\ell_1\}$ which is not in the fixed point (however, it is subsumed by the cell $\{\ell_0, \ell_1\}$). Intuitively, the antichain algorithm has forgotten which action is to be played next. Notice that playing a again, and thus forever, is not a winning strategy. The next proposition formalizes this intuition.

Proposition 3. *There exists a family of games G_k with $O(p(k))$ many locations for a polynomial p, and a reachability objective ϕ, such that the fixed point computed by the antichain algorithm for (G_k, ϕ) is of polynomial size in k, whereas any finite-memory winning strategy for (G_k, ϕ) is of exponential size in k.*

We first present the ideas of the proof informally. Let $p_1, p_2, \dots$ be the list of prime numbers in increasing order. The action alphabet of the game is $\Sigma = \{\mathsf{tick}, \#, \bot\}$. The game is composed of subgames H_i, each consisting of a loop over p_i locations $\ell_1, \dots, \ell_{p_i}$. From a location ℓ_j, action tick leads to ℓ_{j+1} and from the last location ℓ_{p_i} to the initial location ℓ_1. Formally, for all $1 \leq i \leq k$, we define the subgame H_i with location space $L_i = \{\ell_1, \dots, \ell_{p_i}\}$, initial location ℓ_1, and transition relation $E_i = \{(\ell_j, \mathsf{tick}, \ell_{j+1}) \mid 1 \leq j \leq p_i - 1\} \cup \{(\ell_{p_i}, \mathsf{tick}, \ell_1)\}$. In the sequel, we assume that the location spaces of all H_i are disjoint, *e.g.* by adding a superscript i to the locations of L_i ($L_i = \{\ell_1^i, \dots, \ell_{p_i}^i\}$).

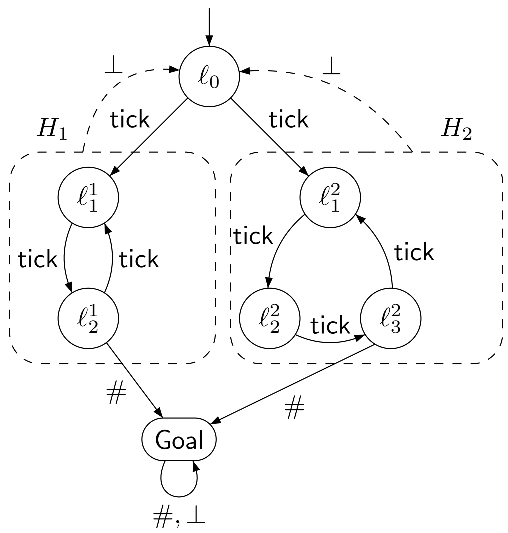

Figure 3: The game G_2.

Figure 3 shows the game G_k for $k = 2$. In general, in G_k, there is a unique trivial observation, so it is a blind game. We also assume that playing a particular action in a location where it is not allowed leads to a sink location from which Goal is not reachable. The plays start in location ℓ_0 where Player 1 should play tick, allowing Player 2 to choose a subgame H_i. As Player 1 does not know in which of the H_i the play is, he should avoid playing action # whenever his knowledge set contains other locations than one of the $\ell^i_{p_i}$ (i.e., the last locations of the subgames). However, after a certain number of steps — $p^*_k = \prod_{i=1}^k p_i$ many, to be precise — the current location of the game will for sure be one of the $\ell^i_{p_i}$. Then, playing # necessarily leads to Goal. The action # is not allowed in any other location, so that Player 1 needs to count the first p^*_k steps before playing that action. Notice that after the first round, Player 1 could play $\perp$, but this would not reduce the amount of memory needed to win. However, it shows that he is winning uniformly from all locations of the subgames H_i and thus the antichain of winning positions is the singleton $\mathsf{Win} = \{L\}$ where L is the set of locations of the game. Since the size p^*_k of the strategy is exponential in the size $\sum_{i=1}^k p_i$ of the antichain Win, the proposition follows.

Proof of Proposition 3. The location space of G_k is the disjoint union of $L_1, \dots, L_k$ and $\{\ell_0, \mathsf{Goal}, \mathsf{Bad}\}$. The initial location is ℓ_0, the target observation consists of Goal, and the sink location is Bad. The transition relation contains each set E_i, the transitions $(\ell^i_j, \perp, \ell_0)$, the transitions $(\ell_0, \mathsf{tick}, \ell^i_1)$, and the transitions $(\ell^i_{p_i}, \#, \mathsf{Goal})$ for all $1 \le i \le k$, $1 \le j \le p_i$. The transition relation is made total by adding the transitions $(q, \sigma, \mathsf{Bad})$ for each location ℓ of G_n and $\sigma \in \{\mathsf{tick}, \#\}$ such that there is no transition of the form (q, σ, q') for $q' \neq \mathsf{Bad}$.

There is only one trivial observation, i.e., the observation alphabet Γ is a singleton.

First we show that Player 1 wins G_k (from ℓ_0). As there is exactly one observation, a strategy for Player 1 corresponds to a function $\alpha : \mathbb{N} \to \Sigma$. Let $\alpha(j) = \mathsf{tick}$ for all $0 \leq j < p_k^*$ and $\alpha(j) = \#$ for all $j \geq p_k^*$. It is easy to check that this strategy is winning for Player 1.

For the second part of the statement assume, towards a contradiction, that there exists a finite-state winning strategy β with less than p_k^* states. Clearly, β cannot play # before the (p_k^*+1)-th round since Player 2 can ensure that one of the subgames H_i is not in $\ell_{p_i}^i$ (by his initial choice). Note that the state reached by the automaton defining β after p_k^* rounds has necessarily been visited in a previous round. Since β has to play # eventually to reach Goal, this means that # must have been played in some round $j < p_k^*$, when at least one of the subgames subgames H_i was not in location $\ell_{p_i}^i$, so that Player 1 would have already lost. This is in contradiction with our assumption that β is a winning strategy. ■

Note that for safety games with imperfect information, there always exists a winning strategy of the size of the antichain representation of the winning cells [14].

Finally, we show that it is not trivial to compute $\mathsf{CPre}(\cdot)$ efficiently. In the antichain representation, the controllable predecessor operator is defined as

$$\mathsf{CPre}(q) = \lceil \{s \subseteq L \mid \exists \sigma \in \Sigma \cdot \forall o \in \Gamma \cdot \exists s' \in q : \mathsf{post}_\sigma(s) \cap \gamma(o) \subseteq s'\} \rceil, \tag{2}$$

or equivalently as

$$\mathsf{CPre}(q) = \bigsqcup_{\sigma \in \Sigma} \bigsqcap_{o \in \Gamma} \bigsqcup_{s' \in q} \{\widetilde{\mathsf{pre}}_\sigma(s' \cup \overline{\gamma(o)})\}, \tag{3}$$

where $\widetilde{\mathsf{pre}}_\sigma(s) = \{s' \in S \mid \mathsf{post}_\sigma(\{s'\}) \subseteq s\}$ and $\overline{\gamma(o)} = L \setminus \gamma(o)$.

Notice that the least upper bound of a set $\{\ell_1, \ldots, \ell_k\}$ of antichains can be computed in polynomial time, whereas a naive algorithm for the greatest lower bound is exponential. The next proposition shows that, assuming a reasonable representation of antichains which allows to decide in polynomial time whether an antichain contains a set larger than n, it is unlikely that $\mathsf{CPre}(\cdot)$ is computable in polynomial time.

Proposition 4. *The following problem is* NP-HARD*: given a game with imperfect information G, an antichain q and an integer n, decide whether there exists a set $B \in \mathsf{CPre}(q)$ with $|B| \geq n$.*

Proof. We reduce the NP-COMPLETE problem 3SAT to our problem. Let P be a finite set of propositions. We denote by $\bar{P} = \{\bar{p} \mid p \in P\}$ the set of negated propositions and by $L = P \cup \bar{P}$ the set of literals. An instance of 3SAT is a set $C = \{c_1, \ldots, c_k\}$ of clauses $c_i \in L \times L \times L$ for $1 \leq i \leq k$, corresponding to the Boolean formula $c_1 \wedge \cdots \wedge c_k$ where each clause is the disjunction of its literals.

Given an instance C of 3SAT, we construct an instance of our decision problem consisting of a game with imperfect information G (see Figure 4), an antichain q and an integer n. The components of the game $G = (S, \Sigma, \rightarrow, \Gamma)$ are:

- $S = L \cup \{u_x, v_x, w_x \mid x \in P \cup C\}$;
- $\Sigma = \{\sigma\}$;
- The transition relation $\rightarrow$ is the union of the following sets:
  - $\{(p, \sigma, u_p), (p, \sigma, w_p), (\bar{p}, \sigma, v_p), (\bar{p}, \sigma, w_p) \mid p \in P\}$,
  - $\{(\bar{p}, \sigma, u_c), (\bar{q}, \sigma, v_c), (\bar{r}, \sigma, w_c) \mid c = (p, q, r) \in C\}$,
  - $\{u_x, v_x, w_x \mid x \in P \cup C\} \times \{\sigma\} \times \{u_y, v_y, w_y \mid y \in P \cup C\}$;
- $\Gamma = \{\{u_x, v_x, w_x\} \mid x \in P \cup C\} \cup \{L\}$.

Notice that the transition relation is total and that the observations cover the location space. Further, let

$$q = \{\{v_p, w_p \mid p \in P \cup C\}, \{u_p, w_p \mid p \in P \cup C\}, \{u_p, v_p \mid p \in P \cup C\}\},$$

and set $n = |P|$.

At this point, we have

$$\begin{aligned}
\mathsf{CPre}(q) &= \bigsqcap_{o \in \Gamma} \bigsqcup_{s' \in q} \{\widetilde{\mathsf{pre}}_\sigma(s' \cup \overline{o})\} \\
&= \bigsqcap_{x \in P \cup C} \bigsqcup_{s' \in q} \{\widetilde{\mathsf{pre}}_\sigma(s' \cup \overline{\{u_x, v_x, w_x\}})\} \sqcap \bigsqcup_{s' \in q} \{\widetilde{\mathsf{pre}}_\sigma(s' \cup \bar{L})\} \\
&= \bigsqcap_{x \in P \cup C} \lceil \{\widetilde{\mathsf{pre}}_\sigma(S \setminus \{u_x\}), \widetilde{\mathsf{pre}}_\sigma(S \setminus \{v_x\}), \widetilde{\mathsf{pre}}_\sigma(S \setminus \{w_x\})\} \rceil \\
&\quad \sqcap \bigsqcup_{s' \in q} \{\widetilde{\mathsf{pre}}_\sigma(\bar{L})\} \\
&= \bigsqcap_{p \in P} \lceil \{L \setminus \{p\}, L \setminus \{\bar{p}\}, L \setminus \{p, \bar{p}\}\} \rceil \\
&\quad \sqcap \bigsqcap_{(p,q,r) \in C} \lceil \{L \setminus \{\bar{p}\}, L \setminus \{\bar{q}\}, L \setminus \{\bar{r}\}\} \rceil \sqcap \{S\} \\
&= \bigsqcap_{p \in P} \underbrace{\{L \setminus \{p\}, L \setminus \{\bar{p}\}\}}_{A_p} \sqcap \bigsqcap_{c = (p,q,r) \in C} \underbrace{\lceil \{L \setminus \{\bar{p}\}, L \setminus \{\bar{q}\}, L \setminus \{\bar{r}\}\} \rceil}_{A_c}.
\end{aligned}$$

We show that $C = \{c_1, \ldots, c_k\}$ is satisfiable if and only if there exists a set $B \in \mathsf{CPre}(q)$ with $|B| \geq n$.

First, assume that C is satisfiable and let $f : P \rightarrow \{\mathsf{true}, \mathsf{false}\}$ be a truth assignment satisfying C. Set

$$B' = \{p \in P \mid f(p) = \mathsf{true}\} \cup \{\bar{p} \in \bar{P} \mid f(p) = \mathsf{false}\}.$$

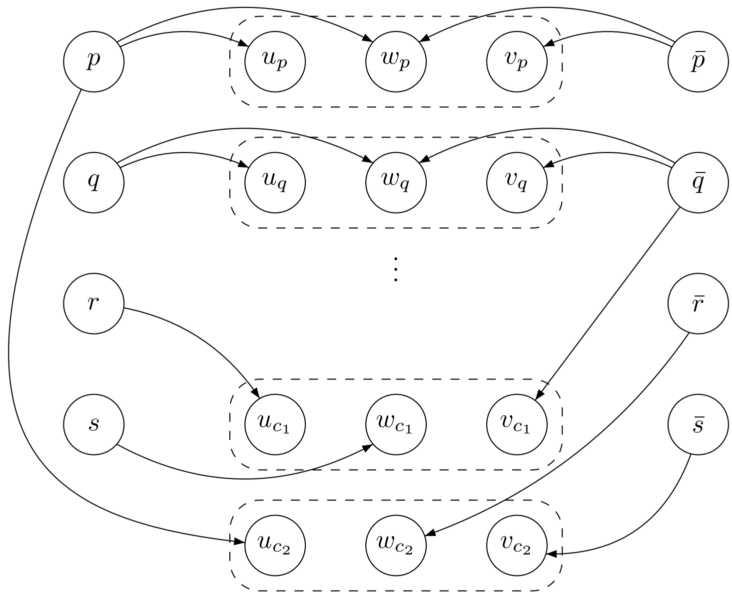

Figure 4:
Reduction for the formula $\underbrace{(\bar{r} \vee q \vee \bar{s})}_{c_1} \wedge \underbrace{(\bar{p} \vee s \vee r)}_{c_2}$.

Then, (i) $|B'| = n$, (ii) for all propositions $p \in P$, if $f(p) = \mathsf{true}$ then $B' \subseteq L \setminus \{\bar{p}\}$, and if $f(p) = \mathsf{false}$ then $B' \subseteq L \setminus \{p\}$, so that there exists $B \in A_p$ such that $B' \subseteq B$, and (iii) as each clause $c \in C$ is satisfied by f, there exists a literal φ_c in c such that either $\varphi_c = p$ and $f(p) = \mathsf{true}$, or $\varphi_c = \bar{p}$ and $f(p) = \mathsf{false}$. Hence, either there exists a proposition p in c and $B' \subseteq L \setminus \{\bar{p}\}$, or there exists a negated proposition $\bar{p}$ in c and $B' \subseteq L \setminus \{p\}$, so that there exists $B \in A_c$ such that $B' \subseteq B$. In either case, there must exist a set $B \in \mathsf{CPre}(q)$ with $|B| \geq n$.

Second, assume that $B \in \mathsf{CPre}(q)$ and $|B| \geq n$. Then for all $p \in P$, there exists $B' \in A_p$ such that $B \subseteq B'$ $(\star)$. In particular, this entails that $B \subseteq L$. Let us show that $p \in B$ iff $\bar{p} \notin B$ for all $p \in P$. Towards a contradiction, assume that $p \in B$ and $\bar{p} \in B$ for some $p \in P$. Then, $\{p, \bar{p}\} \subseteq B$ but $\{p, \bar{p}\} \not\subseteq L \setminus \{p\}$ and $\{p, \bar{p}\} \not\subseteq L \setminus \{\bar{p}\}$. Hence $B \not\subseteq L \setminus \{p\}$ and $B \not\subseteq L \setminus \{\bar{p}\}$, which contradicts $(\star)$. Similarly, it is impossible that $p \notin B$ and $\bar{p} \notin B$, because $|B| \geq n$ would imply that there exists $q \in P$ such that $\{q, \bar{q}\} \subseteq B$. Now, take $f(p) = \mathsf{true}$ iff $p \in B$. For each clause $c \in C$, there exists $B' \in A_c$ such that $B \subseteq B'$, hence there exists a literal φ_c in c such that $\varphi_c \in B$, and thus f satisfies c. ■

4. Strategy Construction with Antichains

We present a procedure for constructing a winning strategy for a parity game of imperfect information $G = (L, l_0, \Delta, \gamma)$ over the alphabets Σ and Γ. It will sometimes be convenient to reason in terms of the equivalent perfect-information game G^{K} obtained via the subset construction in Section 3. Let $\mathcal{C}$ denote the set of all cells s such that $s \subseteq \gamma(o)$ for some $o \in \Gamma$. Thus, $\mathcal{C}$ contains all locations of G^{K}. For $\mathcal{R} \subseteq \mathcal{C}$, a *cell strategy* on $\mathcal{R}$ is a memoryless strategy $\alpha : \mathcal{R} \to \Sigma$ for Player 1 in G^{K}. Given an objective $\phi \subseteq \mathcal{C}^\omega$ in G^{K}, we define

$$\begin{aligned}\mathsf{Win}^{\mathcal{R}}(\phi) := \{\, s \in \mathcal{R} \mid\ & \text{there exists a cell strategy } \alpha \text{ such that} \\ & \mathrm{Outcome}(G^{\mathsf{K}}_s, \alpha) \subseteq \phi \cap \mathsf{Safe}(\mathcal{R}) \,\}.\end{aligned}$$

In words, $\mathsf{Win}^{\mathcal{R}}(\phi)$ consists of cells s such that there exists a winning cell strategy for Player 1 to ensure ϕ starting from cell s and maintaining the play of G^{K} in $\mathcal{R}$.

In Algorithm 1, we present a procedure to construct a winning cell strategy in G^{K} for objectives of the form

$$\mathsf{Reach}(\mathcal{T}) \cup (\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F})),$$

where $\mathcal{T}, \mathcal{F} \subseteq \mathcal{C}$ are downward-closed sets of cells and $p : \Gamma \to \mathbb{N}$ is a priority function over observations. With p extended naturally to cells, the set $\mathsf{Parity}(p)$ contains the sequence of cells such that the minimal priority of a cell appearing infinitely often is even. The parity objective $\mathsf{Parity}(p)$ corresponds to the special case where $\mathcal{F} = \mathcal{C}$ and $\mathcal{T} = \emptyset$. Note that a winning strategy does not need to be defined on $\mathcal{T}$ since $\mathsf{Reach}(\mathcal{T})$ is satisfied for all cells in $\mathcal{T}$. Memoryless strategies are sufficient for this kind of objective in games with perfect information. Thus, we can restrict our attention, without loss of generality, to memoryless cell strategies.

Informal description. We first present an informal description of Algorithm 1. The algorithm is based on two elementary procedures $\mathsf{ReachOrSafe}(\mathcal{T}, \mathcal{F})$ and $\mathsf{ReachAndSafe}(\mathcal{T}, \mathcal{F})$ that use antichains to compute the set of winning cells and a winning strategy for the objectives $\mathsf{Reach}(\mathcal{T}) \cup \mathsf{Safe}(\mathcal{F})$ and $\mathsf{Reach}(\mathcal{T}) \cap \mathsf{Safe}(\mathcal{F})$, respectively, for given downward-closed sets of cells $\mathcal{T} \subseteq \mathcal{C}$ and $\mathcal{F} \subseteq \mathcal{C}$. In games with perfect information, it is known that memoryless winning strategies exist for such combinations of safety and reachability objectives.

The algorithm works recursively, reducing the number of priorities. Given a parity function p, we denote by $p-2$ the parity function such that, for all $o \in \Gamma$, we have $(p-2)(o) = p(o)$ if $p(o) \leq 1$, and $(p-2)(o) = p(o) - 2$ otherwise. For $i \geq 0$, we denote by $\mathcal{C}_p(i) = \{\, s \in \mathcal{C} \mid s \subseteq \gamma(o), o \in \Gamma, p(o) = i \,\}$ the set of cells with priority i. Let W_1 and W_2 be disjoint sets of cells, and let α_1 be a cell strategy on W_1 and α_2 be a cell strategy on W_2. We denote by $\alpha_1 \cup \alpha_2$ the cell strategy on $W_1 \cup W_2$ such that for all $s \in W_1 \cup W_2$, we have $(\alpha_1 \cup \alpha_2)(s) = \alpha_1(s)$ if $s \in W_1$, and $(\alpha_1 \cup \alpha_2)(s) = \alpha_2(s)$ otherwise.

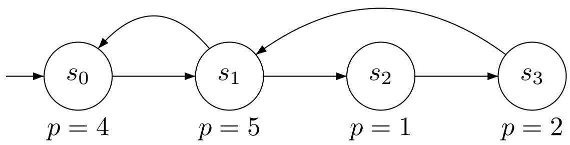

Figure 5: A parity game with perfect information to illustrate Algorithm 1.

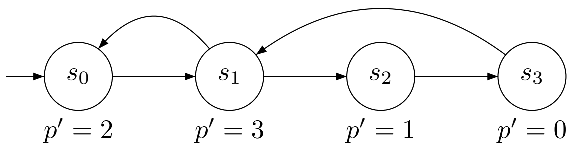

Figure 6: The parity game of Figure 5, with parity function $p' = p - 2$.

Without loss of generality we assume that the cells in the target set $\mathcal{T}$ are *absorbing*, that is, they only have self-loops and no other out-going transitions. In line 1 of Algorithm 1, we compute $W = \mathsf{Win}^{\mathcal{C}}(\phi)$ using the antichain algorithm of [6]. As we assume that cells in $\mathcal{T}$ are absorbing, a winning cell strategy for the objective ϕ ensures that the set W is never left. In the rest of the algorithm and in the arguments below, we consider the sub-game induced by W. In line 2 of Algorithm 1, the set W^* of winning cells and a winning cell strategy α^* on $W^* \setminus \mathcal{T}$ for the objective $\mathsf{Reach}(\mathcal{T})$ is computed by invoking the procedure $\mathsf{ReachAndSafe}$ with target $\mathcal{T}$ and safe set W. Then, the set W_0 of cells is obtained along with a cell strategy α_0 ensuring that either W^* or the set of priority 0 cells in W is reached. After this, the algorithm iterates a loop as follows: at iteration $i+1$, let W_i be the set of cells already obtained in the previous iteration and let $A_i = W \setminus W_i$. The algorithm is invoked recursively with W_i as target set, $A_i \setminus \mathcal{C}_p(1)$ as the safe set, and $p-2$ as the priority function to obtain a set W_{i+1} as a result. In the base case, where W consists of priorities 0, 1, and 2 only, since A_i has no priority 0 cells, the objective $\mathsf{Reach}(W_i) \cup (\mathsf{Parity}(p-2) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1))$ can be equivalently written as $\mathsf{Reach}(W_i) \cup \mathsf{Safe}(A_i \cap \mathcal{C}_p(2))$. Therefore, in the base case, the recursive call is replaced by $\mathsf{ReachOrSafe}(W_i, A_i \cap \mathcal{C}_p(2))$. Notice that $W_i \subseteq W_{i+1}$. The algorithm proceeds until it reaches a fixed point $W_i = W_{i+1}$.

Example. To illustrate how Algorithm 1 works, let us consider a simple example involving only one player with perfect information. In the game structure depicted in Figure 5, all out-going edges from a state are controlled by Player 1. The safe set is $\mathcal{F} = \{ s_0, s_1, s_3 \}$ and the target set $\mathcal{T}$ is empty. The parity function p is as follows: $p(s_0) = 4$, $p(s_1) = 5$, $p(s_2) = 1$, and $p(s_3) = 2$. In the game shown, the winning set W for the objective $\mathsf{Reach}(\mathcal{T}) \cup (\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}))$ is the set $\{ s_0, s_1, s_3 \}$. The main steps of the computation of Algorithm 1 are as follows: the algorithm enters the recursion for the first step with the parity

Algorithm 1: Imperfect-Information Game Solver - **Solve**(G, $\mathcal{T}$, $\mathcal{F}$, p)

Input : A game structure G with target $\mathcal{T} \subseteq \mathcal{C}$, safe set $\mathcal{F} \subseteq \mathcal{C}$ and parity function p on Γ.

Output: $W = \mathsf{Win}^{\mathcal{C}}(\phi)$ where $\phi := \mathsf{Reach}(\mathcal{T}) \cup (\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}))$, and a winning cell strategy α on $W \setminus \mathcal{T}$ for ϕ.

begin
- $W \leftarrow \mathsf{Win}^{\mathcal{C}}(\phi)$
- $(W^*, \alpha^*) \leftarrow \mathsf{ReachAndSafe}(\mathcal{T}, W)$
- $(W_0, \alpha_0) \leftarrow \mathsf{ReachAndSafe}(W^* \cup (\mathcal{C}_p(0) \cap W), W))$
- Let α_0' be a cell strategy on $(\mathcal{C}_p(0) \cap W) \setminus W^*$ such that
  $\mathsf{post}_{\alpha_0'(s)}(s) \cap \gamma(o) \in W$ for all $o \in \Gamma$ and $s \in (\mathcal{C}_p(0) \cap W) \setminus W^*$
- $\overline{\alpha}_0 \leftarrow \alpha_0 \cup \alpha_0' \cup \alpha^*$
- $i \leftarrow 0$
- **repeat**
  - $A_i \leftarrow W \setminus W_i$
  - **if** $W \subseteq \mathcal{C}_p(0) \cup \mathcal{C}_p(1) \cup \mathcal{C}_p(2)$ **then**
    - $(W_{i+1}, \alpha_{i+1}) \leftarrow \mathsf{ReachOrSafe}(W_i, A_i \cap \mathcal{C}_p(2))$
  - **else**
    - $(W_{i+1}, \alpha_{i+1}) \leftarrow$ **Solve**(G, W_i, $A_i \setminus \mathcal{C}_p(1)$, $p - 2$)
  - $\overline{\alpha}_{i+1} \leftarrow \overline{\alpha}_i \cup \alpha_{i+1}$
  - $i \leftarrow i + 1$
- **until** $W_i = W_{i-1}$
- **return** $(W_i, \overline{\alpha}_i)$

end

function $p-2$ on the set W, the safe set $\mathcal{F}$ and the empty target set (the modified parity function is shown in Figure 6). The computation of the set W^* in step 1 of this recursive call yields the empty set. The state s_3 (with priority 0 under the parity function $p-2$) cannot be reached from s_0 or s_1 by staying safe in the set $W = \{ s_0, s_1, s_3 \}$ (since to reach the state s_3 from s_0 or s_1 the state s_2 must be reached). Hence the set $W_0 = \mathsf{ReachAndSafe}(\{ s_3 \}, W)$ computed in step 2 of this recursive call is $\{ s_3 \}$. The algorithm then enters the following recursive step with the set $\{ s_0, s_1 \}$ of states, and this recursive call returns the set $\{ s_0, s_1 \}$ as the winning set. Then in the iterations of the first recursive step, the set $\{ s_0, s_1 \}$ acts as the target set W_i. Since the state s_2 can reach the target set $W_i = \{ s_0, s_1 \}$ by staying safe in W, the state s_2 is included in the winning set. This illustrates the main computation steps of Algorithm 1.

Correctness of the iteration. First, we have $W \setminus W^* \subseteq \mathcal{F}$. This holds essentially because Player 1 cannot reach $\mathcal{T}$ from $W \setminus W^*$. More precisely, if we suppose that a cell $s \in W \setminus W^*$ does not belong to $\mathcal{F}$, then against every cell strategy for Player 1, there is a Player 2 strategy which ensures that the set $\mathcal{T}$ is not reached from s. Hence from s, against every cell strategy for Player 1, there is a Player 2 strategy to ensure that the condition $\mathsf{Reach}(\mathcal{T}) \cup \mathsf{Safe}(\mathcal{F})$, and thus $\phi = \mathsf{Reach}(\mathcal{T}) \cup (\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}))$ is violated, in contradiction with $s \in W = \mathsf{Win}^{\mathcal{C}}(\phi)$. The significance of the claim is that, if W^* is reached, then Player 1 can ensure that $\mathcal{T}$ is reached, and since $W \setminus W^* \subseteq \mathcal{F}$ it follows that, if W^* is not reached, then the game stays safe in $\mathcal{F}$.

To establish the correctness of the iterative step, we claim that from the set W_{i+1} the cell strategy α_{i+1} on $W_{i+1} \setminus W_i$ which ensures

$$\mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p-2) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1))\big),$$

also ensures that

$$\mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F} \setminus \mathcal{C}_p(1))\big).$$

Notice that in $A_i \setminus \mathcal{C}_p(1)$, there is no cell with priority 0 or priority 1 for the priority function p, since $\mathcal{C}_p(0) \cap W \subseteq W_0 \subseteq W_i$. Hence, we have

$$\mathsf{Parity}(p-2) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1)) = \mathsf{Parity}(p) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1)).$$

Since $A_i \subseteq W \setminus W_0 \subseteq W \setminus W^* \subseteq \mathcal{F}$, it follows that the cell strategy α_{i+1} on $W_{i+1} \setminus W_i$ which ensures

$$\mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p-2) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1))\big),$$

also ensures that

$$\mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F} \setminus \mathcal{C}_p(1))\big)$$

holds from all cells in W_{i+1}. By induction on i, composing the cell strategies (i.e., by taking the union of strategies obtained in the iteration) we obtain that from W_{i+1}, the cell strategy $\overline{\alpha}_{i+1}$ on $W_{i+1} \setminus \mathcal{T}$ for Player 1 ensures the condition $\mathsf{Reach}(W_0) \cup \big(\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}) \cap \mathsf{coBuchi}(\mathcal{F} \setminus \mathcal{C}_p(1))\big)$. Note that, to apply the induction step for i times, one may visit cells in $\mathcal{C}_p(1)$, but only finitely many times.

Termination. We claim that, upon termination, we have $W_i = W$. Assume towards a contradiction that the algorithm terminates with $W_i = W_{i+1}$ and $W_{i+1} \neq W$. Then the following assertions hold. The set $A_i = W \setminus W_i$ is nonempty and

$$W_{i+1} = W_i = \mathsf{Win}^W\Big(\mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p-2) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1))\big)\Big),$$

that is, in the whole set A_i against all Player 1 cell strategies, Player 2 can ensure the complementary objective, i.e.,

$$\mathsf{Safe}(A_i) \cap \big(\mathsf{coParity}(p-2) \cup \mathsf{Reach}(A_i \cap \mathcal{C}_p(1))\big).$$

Now, we show that satisfying the above objective also implies satisfying the objective $\mathsf{Safe}(A_i) \cap \mathsf{coParity}(p)$. Consider a cell strategy for Player 1, and consider the counter-strategy for Player 2 that ensures that the game stays in A_i, and also ensures that $\mathsf{coParity}(p-2) \cup \mathsf{Reach}(A_i \cap \mathcal{C}_p(1))$ is satisfied. If a play visits $A_i \cap \mathcal{C}_p(1)$ only finitely many times, then from some point onwards, it only visits cells in A_i that do not have priority p equal to 1 or 0, and hence $\mathsf{coParity}(p-2) = \mathsf{coParity}(p)$. Otherwise, the set $A_i \cap \mathcal{C}_p(1)$ is visited infinitely often and A_i is never left. Since A_i has cells of priority p equal to 0, it means that Player 2 satisfies the $\mathsf{coParity}(p)$ objective. It follows that in A_i against all Player 1 cell strategies, Player 2 can ensure $\mathsf{Safe}(A_i) \cap \mathsf{coParity}(p)$. This is a contradiction to the fact that $A_i \subseteq W = \mathsf{Win}^W(\phi)$ and $\mathsf{Safe}(A_i) \cap \mathsf{coParity}(p) \subseteq \Gamma^\omega \setminus \phi$.

The insights formulated above lead to the following theorem.

Theorem 1. *Given an imperfect-information game G with target $\mathcal{T} \subseteq \mathcal{C}$, safe set $\mathcal{F} \subseteq \mathcal{C}$ and a parity function p on Γ, Algorithm 1 computes $W = \mathsf{Win}^{\mathcal{C}}(\phi)$, where $\phi = \mathsf{Reach}(\mathcal{T}) \cup (\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}))$, and a winning cell strategy α on $W \setminus \mathcal{T}$ for ϕ.*

Proof. The statement follows from the correctness of the iteration, and the fact that $W = W_i$ for some i. From our previous analysis, it also follows that from all locations in W, the obtained cell strategy ensures

$$\mathsf{Reach}(W_0) \cup (\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}) \cap \mathsf{coBuchi}(\mathcal{F} \setminus \mathcal{C}_p(1)).$$

We now complete the argument by showing that the cell strategy is winning for ϕ. The cell strategy on W_0 ensures that $\mathcal{T}$ is reached from cells in W^*, from cells in $\mathcal{C}_p(0) \cap W$ it ensures to stay in W, and in all remaining cells in W_0 it ensures to reach $W^* \cup (\mathcal{C}_p(0) \cap W)$. The following case analysis completes the proof.

1. If the set W_0 is visited infinitely often, then (a) if W^* is reached, then $\mathcal{T}$ is reached; (b) otherwise $\mathcal{C}_p(0) \cap W$ is visited infinitely often and the game always stays safe in $W \setminus W^* \subseteq \mathcal{F}$. This ensures that $\mathsf{Parity}(p)$ is also satisfied.
2. If W_0 is visited only finitely often, then the play never reaches W^*, otherwise it would reach $\mathcal{T}$ and stay in $\mathcal{T}$ forever, and hence $\mathsf{Safe}(\mathcal{F})$ is satisfied, such that the objective $\mathsf{Parity}(p) \cap \mathsf{Safe}(\mathcal{F}) \cap \mathsf{coBuchi}(\mathcal{F} \setminus \mathcal{C}_p(1))$ is attained. Overall, it follows the objective ϕ is satisfied.

■

Antichain algorithm. To turn Algorithm 1 into an antichain algorithm, all set operations must preserve the downward-closed property. The union and intersection operations on sets preserve the downward-closed property of sets, but the complementation operation does not. Observe that Algorithm 1 performs complementation in line 9 ($A_i \leftarrow W \setminus W_i$) and uses the set A_i in lines 11 and 12. This was done for the ease of correctness proof of the algorithm. To see that the complementation step is not necessary, observe that

$$\mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p-2) \cap \mathsf{Safe}(A_i \setminus \mathcal{C}_p(1))\big) = \mathsf{Reach}(W_i) \cup \big(\mathsf{Parity}(p-2) \cap \mathsf{Safe}(W \setminus \mathcal{C}_p(1))\big).$$

Indeed, if a play never visits W_i, then the play is in $\mathsf{Safe}(A_i \setminus \mathcal{C}_p(1))$ if, and only if, it is in $\mathsf{Safe}(W \setminus \mathcal{C}_p(1))$. Also, note that the expression $\mathsf{Parity}(p-2) \cap \mathsf{Safe}(W \setminus \mathcal{C}_p(1))$ can be equivalently written as $\mathsf{Parity}(p-2) \cap \mathsf{Safe}(W \cap \bigcup_{i \geq 2} \mathcal{C}_p(i))$. It follows that every set operation in Algorithm 1 preserves downward-closed property. Since the algorithm of [6] is antichain based, it follows that step 1 of Algorithm 1 is compatible with antichain representation. This demonstrates the following statement.

Theorem 2. *Algorithm 1 is compatible with the antichain representation.*

We remark that the explicit construction of the strategies takes place only in few steps of the algorithm: at line 2 and 3 of each recursive call where cell strategies are computed for reachability objectives, and in the base case (parity games with priorities $0, 1$ and 2) in line 11 where cell strategies are computed for union of safety and reachability objectives. Also note that we never need to compute strategies for the target set $\mathcal{T}$, and therefore in line 10, we would obtain strategies for the set $W_{i+1} \setminus W_i$. Hence, once the strategy is computed for a set, it will never be modified in any subsequent iteration.

5. Implementation

We have implemented Algorithm 1 in a tool called Alpaga [15] written in Python, except for the BDD package which is written in C. We use the CUDD BDD library [16] with its PYCUDD Python binding. There is a certain performance overhead in using Python, but we chose it to enhance readability and to make the code easy to change. We believe this is important in the context of academic research, as we expect other researchers to experiment with the tool, tweak the existing algorithms and add their own ones. In the same spirit, the code architecture is designed in a modular way and offers possibilities to reuse utility functions like, for example, the successors of a set of states, the controllable predecessors, or antichain-handling functions.

The building blocks of the algorithm are the computation of the controllable predecessor operator $\mathsf{CPre}(\cdot)$, and the two basic procedures $\mathsf{ReachOrSafe}$ and $\mathsf{ReachAndSafe}$.

Controllable predecessor. According to Proposition 4, computing $\mathsf{CPre}(\cdot)$ is likely to require time exponential in the number of observations. Therefore, it is natural to let the BDD machinery evaluate the quantifications 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 $n \cdot |\Sigma|$ many BDDs $T_\sigma(\ell_i)$, for $\sigma \in \Sigma$ and $1 \leq i \leq n$, with logarithmic encoding over $\bar{y}$. Thus, $T_\sigma(\ell_i)$

represents the set $\{\ell_j \mid (\ell_i, \sigma, \ell_j) \in \Delta\}$. The observations $\Gamma = \{o_1, \dots, o_p\}$ are encoded by $\lceil \log_2 p \rceil$ many Boolean variables $b_0, b_1, \dots$ in the BDD B_Γ defined by

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

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

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

$$\mathsf{CP}_\sigma \equiv \forall \bar{b} \cdot \bigvee_{1 \le k \le t} \bigwedge_{1 \le i \le n} x_i \to \big[\forall \bar{y} \cdot \big(T_\sigma(\ell_i) \wedge B_\Gamma\big) \to S_k\big].$$

Then, we define $\mathsf{CP} \equiv \bigvee_{\sigma \in \Sigma} \mathsf{CP}_\sigma(q)$, 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 $\subset$:

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

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

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

Strategy construction. To compute $\mathsf{ReachOrSafe}(\mathcal{T}, \mathcal{F})$, we evaluate the following fixed-point formula in the lattice of antichains: $\varphi_1 \equiv \nu X.(\mathcal{F} \sqcap \mathsf{CPre}(X)) \sqcup \mathcal{T}^*$ where $\mathcal{T}^* = \mu X.\mathsf{CPre}(X) \sqcup \mathcal{T}$. To compute $\mathsf{ReachAndSafe}(\mathcal{T}, \mathcal{F})$, we evaluate $\varphi_2 \equiv \mu X.\mathcal{F} \sqcap (\mathsf{CPre}(X) \sqcup \mathcal{T})$.

While computing $q' = \mathsf{CPre}(q)$, we associate with each cell s in the antichain q' the action σ to be played in order to ensure reaching a cell in q. Each such pair (s, σ) is output with a rank (a natural number) that is incremented at each iteration of $\mathsf{CPre}()$. Thus, a strategy is represented by a set $\Pi = \{(s_1, \mathsf{Rank}_1, \sigma_1), \dots, (s_n, \mathsf{Rank}_n, \sigma_n)\}$ of triples $(s_i, \mathsf{Rank}_i, \sigma_i) \in 2^L \times \mathbb{N} \times \Sigma$ where s_i is a cell, and σ_i is an action. The action to be played in a given cell s is the action a_i associated with the cell s_i with minimal rank that contains s. Formally, given the current knowledge s of Player 1, let $(s_i, \mathsf{Rank}_i, \sigma_i)$ 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 σ_i in s.

Our implementation applies the following rules to simplify the strategies and to 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: $(s, \mathsf{Rank}, \sigma) \succeq (s', \mathsf{Rank}', \sigma')$ if $\mathsf{Rank} \le \mathsf{Rank}'$ and $s' \subseteq s$. Intuitively, the rule specifies that we can delete $(s', \mathsf{Rank}', \sigma')$ whenever all cells contained in s' are also contained in s; since $\mathsf{Rank} \leq \mathsf{Rank}'$, the strategy can always choose $(s, \mathsf{Rank}, \sigma)$ and play σ.

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

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 that 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). Details about the tool features and options can be found in [15]. Alpaga is available for download at `http://www.antichains.be/alpaga` for Linux-operated systems. For convenience, the tool can also be tested through a web interface. We provide the source code, the executable, an online demo, and several examples.

6. Examples

We illustrate the use of imperfect-information games for the synthesis of distributed programs. Currently, the input language of Alpaga does not allow to describe the transition graph in a symbolic way via variables. To solve our examples, we had to manually translate the programs into game graphs, enumerating the valuations of the variables. This is manageable for small examples, but for larger ones Alpaga needs to be extended to handle symbolic input.

6.1. An example with locks

Consider the program in Figure 7 that acquires and releases a lock. The `if`-statement in line 1 corresponds to a nondeterministic choice that abstracts away any concrete branching condition. The program has a local variable `got_lock` used to ensure that the functions `lock()` and `unlock()` are called in strict alternation. The program is partially specified in that lines 3 and 6 are non-deterministic assignments. We provide a set of possible choices to assign the variable: `inc`, `dec`, `s0`, or `s1`. We use a game formulation to find the correct assignment, in which Player 1 resolves the choices in line 3 and 6, while Player 2 controls the branching choice in line 1. Solving this game and constructing a winning strategy for Player 1 provides a way to synthesize a correct program. This setting was used in [17] as an example of program repair. There, a diagnosis is performed on a concrete version of Figure 7, and a fault is localized in

```
    int got_lock = 0;
    do {
1.      if (*) {
2.              lock();
3.              | got_lock++ (inc);
                | got_lock-- (dec);
                | got_lock=1 (s1);
                | got_lock=0 (s0);
        }
4.      if (got_lock != 0) {
5.              unlock();
        }
6.      | got_lock++ (inc);
        | got_lock-- (dec);
        | got_lock=1 (s1);
        | got_lock=0 (s0);
    } while(true)
```

```
void lock() {
   assert(L == 0);
   L=1;
}

void unlock() {
   assert(L == 1);
   L=0;
}
```

Figure 7: Example with locks.

the assignment of `got_lock`. The repair consists in fixing the correct value of the variable.

The actual status of the lock is tracked by the variable `L` which is not visible to the main program. Assertions over `L` are used to enforce the alternation of lock holds and releases, encoding the requirement as a safety objective (namely, that the assertions are never violated).

In [17], the game is solved under the assumption of perfect information (that is, `L` is visible), thus assuming that `L` is a global variable. A correct repair is obtained by choosing an assignment that is common to a memoryless winning strategy for `L=0` and `L=1`, universally quantifying `L` in a permissive strategy (i.e., a strategy that allows multiple actions to be played). Note that maximally permissive strategies exist only for safety objectives [18]. This approach is sound but not complete, as it may not find a winning strategy even if there exists one.

In the setting of imperfect information, states that differ only by the value of `L` display the same observation and are thus indistinguishable to Player 1. The observation-based strategy constructed by Alpaga is then guaranteed to provide assignments that do not depend on the value of `L`. The computed strategy corresponds to a memoryless winning strategy in game structure G^{K} obtained from the original game G via subset construction (see Section 3). When used in the game G, the strategy may need (finite) memory for tracking the cell that represents the knowledge of Player 1. To implement the strategy in the program, we may thus need additional variables for tracking this information. In the locking example, this is not the case, as Alpaga constructs a strategy that plays s_0 in line 6, while in line 3, all actions except s_0 can be played.

```
do {
unbounded_wait;
flag[1]:=true;
turn:=2;

| while(flag[1]) nop;           (C1)
| while(flag[2]) nop;           (C2)
| while(turn=1) nop;            (C3)
| while(turn=2) nop;            (C4)
| while(flag[1] & turn=2) nop;  (C5)
| while(flag[1] & turn=1) nop;  (C6)
| while(flag[2] & turn=1) nop;  (C7)
| while(flag[2] & turn=2) nop;  (C8)

fin_wait;  // Critical section
flag[1]:=false;
} while(true)
```

```
do {
unbounded_wait;
flag[2]:=true;
turn:=1;

while(flag[1] & turn=1) nop;

fin_wait;  // Critical section
flag[2]:=false;
} while(true)
```

Figure 8: Mutual-exclusion protocol synthesis.

6.2. Mutual-exclusion protocol

We consider the design of a mutual-exclusion protocol for two processes, following the lines of [19]. We assume that one process (on the right in Figure 8) is completely specified. The second process (on the left in Figure 8) has freedom of choice in line 4. It can use one of 8 possible conditions `C1`–`C8` to guard the entry to its critical section in line 5. The Boolean variables `flag`[1] and `flag`[2] are used to place a request to enter the critical section. They are both visible to each process. The variable `turn` is visible and can be written by the two processes. Thus, all variables are visible to the left process, except the program counter of the right process.

There is also some nondeterminism in the length of the delays in lines 1 and 5 of the two processes. The processes are free to request or not the critical section and thus may wait for an arbitrary amount of time in line 1 (as indicated by `unbounded_wait`), but they have to leave the critical section within a finite amount of time (as indicated by `fin_wait`). In the game model, the length of the delay is chosen by the adversary.

Finally, each computation step is assigned to one of the two processes by a *scheduler*. We require that the scheduler is fair, i.e., it assigns computation steps to both processes infinitely often. In our game model, we encode all fair schedulers by allowing each process to execute an arbitrary finite number of steps, before releasing the turn to the other process. Again, the actual number of computation steps assigned to each process is chosen by the adversary.

The mutual-exclusion requirement (that the processes are never simultaneously in their critical section) and the starvation-freedom requirement (that whenever the left process requests to enter the critical section, it will eventually

	Locations	Observations	Priorities	Execution time (s)
Game1	4	4	Reach.	0.1
Game2	3	2	Reach.	0.1
Game3	6	3	3	0.1
Game4	8	5	5	1.4
Game5	8	5	7	9.4
Game6	11	9	10	50.7
Game7	11	8	10	579.0
Locks	22	14	Safety	0.6
Mutex	50	28	3	57.7

Table 1: Experimental results with Alpaga.

enter) can be encoded using three priorities.

When solving this game with our tool, we find that Player 1 is winning, and that choosing C_8 is a winning strategy.

6.3. Experimental results

We have run Alpaga on the previous examples (locks and mutual exclusion), as well as on seven toy examples (Game1-7) that we have set up to test our tool while programming. We provide those examples as representative of different levels of difficulty for our tool, and also for the purpose of comparing with future versions of our tool, or later with other tools. They are included in the tool package at `http://www.antichains.be/alpaga`. Table 1 shows the size of the games (in terms of number of locations of the game structure, number of observations, and number of priorities). The experiments were conducted on a Intel Dual-processor P8400 (2.3GHz) with 2Gb of RAM. Execution times are given in Table 1, and include strategy construction.

7. Conclusion

We conclude with some discussion and remarks. The antichain approach has been applied to several problems in automata and game theory, always in a way to avoid exponential subset constructions for solving decision problems [12, 13, 20, 21]. In this paper, we go beyond decision problems and consider the construction of a witness, namely, a winning strategy. We show that winning strategies may have size exponentially larger than the antichain representation of winning positions (see Proposition 3). Therefore, a natural question is to search for an alternative (and more compact) representation of strategies, or to show that certain classes of strategies that can be represented compactly are sufficient to win. For instance, in reachability games, a winning strategy does not necessarily need to enforce the objective within minimal number of steps, and this may give enough freedom for reducing the size of winning strategies.

Concerning the implementation of Alpaga, we identify two issues where improvements can be made. First, the game graph has to be provided explicitly which is not convenient for dealing with symbolic (variable-based) games as in the mutex and locks examples. Second, it is not clear whether BDDs are the best data-structure for antichains. We use BDDs because they provide highly optimized algorithms for existential and universal quantification. However, experiments made with other data-structures in automata theory (see above) tend to show that simple data-structures (like arrays) can outperform BDDs. This however holds in applications where the main operations are computable in polynomial time, which is not the case of the controllable predecessor operator for games with imperfect information (see Proposition 4).

Other future works include the extension of the approach and implementation to non-visible objectives, and to weaker notions of winning condition such as almost-sure winning, i.e., winning with probability 1.

References

[1] W. Thomas, On the Synthesis of Strategies in Infinite Games., in: Proc. of STACS 1995, Springer-Verlag, 1–13, 1995.

[2] E. Grädel, W. Thomas, T. Wilke (Eds.), Automata, Logics, and Infinite Games, LNCS 2500, Springer-Verlag, 2002.

[3] E. A. Emerson, C. S. Jutla, Tree Automata, Mu-Calculus and Determinacy, in: Proc. of FoCS 1991, IEEE, 368–377, 1991.

[4] D. Fudenberg, J. Tirole, Game Theory, MIT Press, 1991.

[5] J. Reif, The complexity of two-player games of incomplete information, Journal of Computer and System Sciences 29 (1984) 274–301.

[6] K. Chatterjee, L. Doyen, T. A. Henzinger, J.-F. Raskin, Algorithms for Omega-regular Games of Incomplete Information, Logical Methods in Computer Science 3 (3:4).

[7] T. A. Henzinger, R. Jhala, R. Majumdar, Counterexample-guided control, in: Proc. of ICALP 2003, LNCS 2719, Springer-Verlag, 886–902, 2003.

[8] R. McNaughton, Infinite Games Played on Finite Graphs., Annals of Pure and Applied Logic 65 (2) (1993) 149–184.

[9] W. Zielonka, Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees, Theoretical Computer Science 200 (1998) 135–183.

[10] W. Thomas, Languages, automata, and logic, Handbook of Formal Languages 3 (1997) 389–455.

[11] D. Berwanger, L. Doyen, On the power of imperfect information, in: Proc. of FSTTCS 2008, Dagstuhl Seminar Proceedings 08004, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), 2008.

[12] M. De Wulf, L. Doyen, T. A. Henzinger, J.-F. Raskin, Antichains: A New Algorithm for Checking Universality of Finite Automata, in: Proc. of CAV 2006, LNCS 4144, Springer-Verlag, 17–30, 2006.

[13] M. De Wulf, L. Doyen, N. Maquet, J.-F. Raskin, Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking, in: Proc. of TACAS 2008, LNCS 4693, Springer-Verlag, 63–77, 2008.

[14] M. De Wulf, L. Doyen, J.-F. Raskin, A Lattice Theory for Solving Games of Imperfect Information, in: Proc. of HSCC 2006, LNCS 3927, Springer-Verlag, ISBN 3-540-33170-0, 153–168, 2006.

[15] D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, T. A. Henzinger, Alpaga: A Tool for Solving Parity Games with Imperfect Information, in: Proc. of TACAS 2009, LNCS 5505, Springer, 58–61, 2009.

[16] F. Somenzi, CUDD: Colorado University Decision Diagram Package, http://vlsi.colorado.edu/~fabio/CUDD/, 2005.

[17] B. Jobstmann, A. Griesmayer, R. Bloem, Program Repair as a Game, in: Proc. of CAV: Computer-Aided Verification, LNCS 3576, Springer, 226–238, 2005.

[18] J. Bernet, D. Janin, I. Walukiewicz, Permissive strategies: from parity games to safety games, Inf. Théorique et Applications 36 (3) (2002) 261–275.

[19] K. Chatterjee, T. A. Henzinger, Assume-guarantee synthesis, in: Proc. of TACAS 2007, LNCS 4424, Springer, 261–275, 2007.

[20] A. Bouajjani, P. Habermehl, L. Holík, T. Touili, T. Vojnar, Antichain-Based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata, in: Proc. of CIAA 2008, LNCS 5148, Springer, 57–67, 2008.

[21] L. Doyen, J.-F. Raskin, Antichains for the Automata-Based Approach to Model-Checking, Logical Methods in Computer Science 5 (1:5).