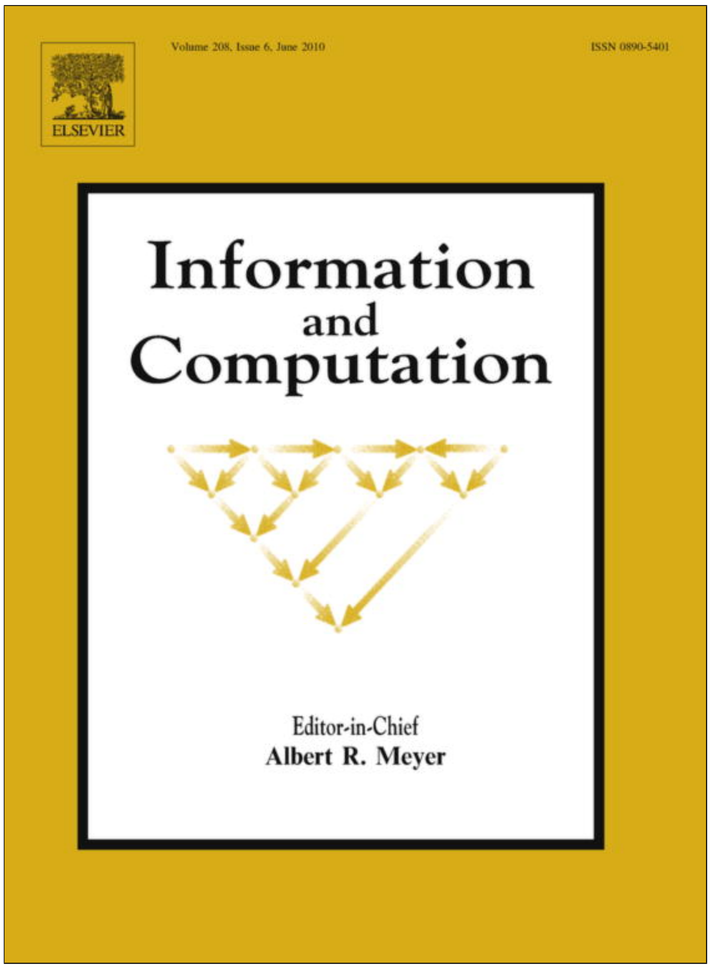
Volume 208, Issue 6, June 2010
ISSN 0890-5401
ELSEVIER
Information and Computation
Editor-in-Chief
Albert R. Meyer

Information and Computation 208 (2010) 666–676

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Solving games via three-valued abstraction refinement[☆]

Luca de Alfaro[a,*], Pritam Roy[b]

[a] Computer Science Department, UC Santa Cruz, USA
[b] Computer Engineering Department, UC Santa Cruz, USA

ARTICLE INFO

Article history:
Received 19 April 2008
Revised 28 April 2009
Available online 29 January 2010

Keywords:
Abstraction
Refinement
Three valued
Model checking
Safety
Reachability

ABSTRACT

Games that model realistic systems can have very large state spaces, making their direct solution difficult. We present a symbolic abstraction-refinement approach to the solution of two-player games with reachability or safety goals. Given a reachability or safety property, an initial set of states, and a game representation, our approach starts by constructing a simple abstraction of the game, guided by the predicates present in the property and in the initial set. The abstraction is then refined, until it is possible to either prove, or disprove, the property over the initial states. Specifically, we evaluate the property on the abstract game in three-valued fashion, computing an over-approximation (the *may* states), and an under-approximation (the *must* states), of the states that satisfy the property. If this computation fails to yield a certain yes/no answer to the validity of the property on the initial states, our algorithm refines the abstraction by splitting *uncertain* abstract states (states that are may-states, but not must-states). The approach lends itself to an efficient symbolic implementation. We discuss the property required of the abstraction scheme in order to achieve convergence and termination of our technique.

1. Introduction

Games provide a computational model that is widely used in applications ranging from controller design, to modular verification, to system design and analysis. The main obstacle to the practical application of games to design and control problems lies in very large state space of games modeling real-life problems. In system verification, one of the main methods for coping with large-size problems is *abstraction.* An abstraction is a simplification of the original system model. The foundations for the use of abstraction in verification have been laid in the works on *abstract interpretation* [1], which has been extended to games in [2].

To be useful, an abstraction should contain sufficient detail to enable the derivation of the desired system properties, while being succinct enough to allow for efficient analysis. Finding an abstraction that is simultaneously informative and succinct is a difficult task, and the most successful approaches rely on the automated construction, and gradual refinement, of abstractions. Given a system and the property, a coarse initial abstraction is constructed: this initial abstraction typically preserves only the information about the system that is most immediately involved in the property, such as the values of the state variables mentioned in the property. This initial abstraction is then gradually, and automatically, refined, until the property can be proved or disproved, in the case of a verification problem, or until the property can be analyzed to the desired level of accuracy, in case of a quantitative problem.

One of the most successful techniques for automated abstraction refinement is the technique of *counterexample-guided refinement,* or CEGAR [3–5]. According to this technique, given a system abstraction, we check whether the abstraction

[☆] This research was supported in part by the NSF Grants CCR-0132780, CNS-0720884, and CNS-0834812.
[*] Corresponding author. Fax: +1 831 529 4829.
Email address: luca@soc.ucsc.edu (L. de Alfaro)

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2009.05.007

satisfies the property. If the answer is affirmative, we are done. Otherwise, the check yields an *abstract counterexample,* encoding a set of "suspect" system behaviors. The abstract counterexample is then further analyzed, either yielding a concrete counterexample (a proof that the property does not hold), or yielding a refined abstraction, in which that particular abstract counterexample is no longer present. The process continues until either a concrete counterexample is found, or until the property can be shown to hold (i.e., no abstract counterexamples are left). The appeal of CEGAR lies in the fact that it is a fully automatic technique, and that the abstraction is refined on-demand, in a property-driven fashion, adding just enough detail as is necessary to perform the analysis. The CEGAR technique has been extended to games in *counterexample-guided control* [6].

We propose here an alternative technique to CEGAR for refining game abstractions: namely, we propose to use *three-valued* analysis [7–9] in order to guide abstraction refinement for games. The technique is suited to *reachability games,* where the goal is to reach a set of target states, and to *safety properties,* where the goal is to stay always in a set of "safe" states. The technique works as follows. Given a game abstraction, we analyze it in three-valued fashion, computing the set of *must-win* states, which are known to satisfy the reachability or safety property, and the set of *never-win* states, which are known not to satisfy the property; the remaining states, for which the satisfaction is unknown, are called the *may-win* states. If this three-valued analysis yields the desired information (for example, showing the existence of an initial state with a given property), the analysis terminates. Otherwise, we refine the abstraction in a way that reduces the number of *may-win* states. The abstraction refinement proceeds in a property-dependent way. For reachability properties, we refine the abstraction at the may-must border, splitting a may-win abstract state into two parts, one of which is known to satisfy the property (and that will become a must-win state). For the dual case of safety properties, the refinement occurs at the may-never border.

Our proposed three-valued abstraction-refinement technique can be implemented in fully symbolic fashion, and it can be applied to games with both finite and infinite state spaces. The technique terminates whenever the game has a finite *region algebra* (a partition of the state space) that is closed with respect to Boolean and controllable-predecessor operators [10]: this is the case for many important classes of games, among which timed games [11,12]. Furthermore, we show that the technique never performs unnecessary refinements: the final abstraction is never finer than a region algebra that suffices for proving the property.

In its aim of reducing the number of may-states, our technique is related to the three-valued abstraction-refinement schemes proposed for CTL and transition systems in [7,8]. Differently from these approaches, however, we avoid the explicit construction of the three-valued transition relation of the abstraction, relying instead on *may* and *must* versions of the controllable-predecessor operators. Our approach provides precision and efficiency benefits. In fact, to retain full precision, the must-transitions of a three-valued model need to be represented as hyper-edges, rather than normal edges [8,9,13]; in turn, hyper-edges are computationally expensive both to derive and to represent. The may and must predecessor operators we use provide the same precision as the hyper-edges, without the associated computational penalty. For a similar reason, we show that our three-valued abstraction-refinement technique is superior to the CEGAR technique of [6], in the sense that it can prove a given property with an abstraction that never needs to be finer, and that can often be coarser. Again, the advantage is due to the fact that [6] represents player-1 moves in the abstract model via must-edges, rather than must hyper-edges. A final benefit of avoiding the explicit construction of the abstract model, relying instead on predecessor operators, is that the resulting technique is simpler to present, and simpler to implement. On the other side, we remark that the techniques of [6] extend easily to parity goals, whereas the refinement scheme we propose can be extended, but only at the price of cumbersome bookkeeping.

While we present the technique for games, the technique also yields a three-valued abstraction-refinement scheme for the verification of safety and reachability properties of transition systems.

2. Preliminary definitions

A two-player game structure $G = \langle S, \lambda, \delta \rangle$ consists of:

- A state space S.
- A turn function $\lambda : S \to \{1, 2\}$, associating with each state $s \in S$ the player $\lambda(s)$ whose turn it is to play at the state. We write $\sim 1 = 2, \sim 2 = 1$, and we let $S_1 = \{s \in S \mid \lambda(s) = 1\}$ and $S_2 = \{s \in S \mid \lambda(s) = 2\}$.
- A transition function $\delta : S \mapsto 2^S \backslash \emptyset$, associating with every state $s \in S$ a non-empty set $\delta(s) \subseteq S$ of possible successors.

The game takes place over the state space S, and proceeds in an infinite sequence of rounds. At every round, from the current state $s \in S$, player $\lambda(s) \in \{1, 2\}$ chooses a successor state $s' \in \delta(s)$, and the game proceeds to s'. The infinite sequence of rounds gives rise to a *path* $\bar{s} \in S^\omega$: precisely, a *path* of G is an infinite sequence $\bar{s} = s_0, s_1, s_2, \ldots$ of states in S such that for all $k \geq 0$, we have $s_{k+1} \in \delta(s_k)$. We denote by Ω the set of all paths.

2.1. Game objectives

An *objective* Φ for a game structure $G = \langle S, \lambda, \delta \rangle$ is a subset $\Phi \subseteq S^\omega$ of the sequences of states of G. A *game* (G,Φ) consists of a game structure G together with an objective Φ for a player. We consider winning objectives that consist in reachability and safety conditions. Given a subset $T \subseteq S$ of states, the *reachability* objective $\diamond T = \{s_0, s_1, s_2, \ldots \in S^\omega \mid \exists k \geq 0. s_k \in T\}$ consists of all paths that reach T; the *safety* objective $\Box T = \{s_0, s_1, s_2, \ldots \in S^\omega \mid \forall k \geq 0. s_k \in T\}$ consists of all paths that stay in T forever. Games with reachability or safety objectives are called reachability and safety games, respectively.

2.2. Strategies and winning states

A *strategy* for player $i \in \{1, 2\}$ in a game $G = \langle S, \lambda, \delta \rangle$ is a mapping $\pi_i : S^* \times S_i \mapsto S$ that associates with every non-empty finite sequence σ of states ending in S_i, representing the past history of the game, a successor state. We require that, for all $\sigma \in S^\omega$ and all $s \in S_i$, we have $\pi_i(\sigma s) \in \delta(s)$. An initial state $s_0 \in S$ and two strategies π_1, π_2 for players 1 and 2 uniquely determine a sequence of states $Outcome(s_0, \pi_1, \pi_2) = s_0, s_1, s_2, \ldots$, where for $k > 0$ we have $s_{k+1} = \pi_1(s_0, \ldots, s_k)$ if $s_k \in S_1$, and $s_{k+1} = \pi_2(s_0, \ldots, s_k)$ if $s_k \in S_2$.

Given an initial state s_0 and a winning objective $\Phi \subseteq S^\omega$ for player $i \in \{1, 2\}$, we say that state $s \in S$ is *winning* for player i if there is a player-i strategy π_i such that, for all player $\sim i$ strategies $\pi_{\sim i}$, we have $Outcome(s_0, \pi_1, \pi_2) \in \Phi$. We denote by $\langle i \rangle \Phi \subseteq S$ the set of winning states for player i for objective $\Phi \subseteq S^\omega$. A result by [14], as well as the determinacy result of [15], ensures that for all ω-regular goals Φ we have $\langle 1 \rangle \Phi = S \setminus \langle 2 \rangle \neg \Phi$, where $\neg \Phi = S \setminus \Phi$. Given a set $\theta \subseteq S$ of initial states, and a property $\Phi \subseteq S^\omega$, we will present algorithms for deciding whether $\theta \cap \langle i \rangle \Phi \neq \emptyset$ or, equivalently, whether $\theta \subseteq \langle i \rangle \Phi$, for $i \in \{1, 2\}$.

2.3. Game abstractions

An *abstraction* V of a game structure $G = \langle S, \lambda, \delta \rangle$ consists of a set $V \subseteq 2^{2^S \setminus \emptyset}$ of *abstract states:* each abstract state $v \in V$ is a non-empty subset $v \subseteq S$ of concrete states. We require $\bigcup V = S$. For subsets $T \subseteq S$ and $U \subseteq V$, we write:

$$U\!\downarrow\, = \textstyle\bigcup_{u \in U} u \qquad T\!\uparrow_V^m = \{v \in V \mid v \cap T \neq \emptyset\} \qquad T\!\uparrow_V^M = \{v \in V \mid v \subseteq T\}. \tag{1}$$

Thus, for a set $U \subseteq V$ of abstract states, $U\!\downarrow$ is the corresponding set of concrete states. For a set $T \subseteq S$ of concrete states, $T\!\uparrow_V^m$ and $T\!\uparrow_V^M$ are the set of abstract states that constitute over and under-approximations of the concrete set T. The following result follows immediately from the definitions (1).

Lemma 1. *For all sets $T \subseteq S$, we have:*

$$T\!\uparrow_V^M \subseteq T\!\uparrow_V^m, \qquad (T\!\uparrow_V^M)\!\downarrow\, \subseteq T \subseteq (T\!\uparrow_V^m)\!\downarrow .$$

We say that the abstraction V of a state space S is *precise* for a set $T \subseteq S$ of states if $T\!\uparrow_V^m = T\!\uparrow_V^M$.

2.4. Controllable-predecessor operators

Two-player games with reachability, safety, or ω-regular winning conditions are commonly solved using *controllable-predecessor operators.* We define the *player-i controllable-predecessor operator* $\mathrm{Cpre}_i : 2^S \mapsto 2^S$ as follows, for all $X \subseteq S$ and $i \in \{1, 2\}$:

$$\mathrm{Cpre}_i(X) = \{s \in S_i \mid \delta(s) \cap X \neq \emptyset\} \cup \{s \in S_{\sim i} \mid \delta(s) \subseteq X\}. \tag{2}$$

Intuitively, for $i \in \{1, 2\}$, the set $\mathrm{Cpre}_i(X)$ consists of the states from which player i can force the game to X in one step. In order to allow the solution of games on the abstract state space V, we introduce abstract versions of $\mathrm{Cpre}_\cdot$ [2]. As multiple concrete states may correspond to the same abstract state, we cannot compute, on the abstract state space, a precise analogous of $\mathrm{Cpre}_\cdot$. Thus, for player $i \in \{1, 2\}$, we define two abstract operators [9]:

- the *may* operator $\mathrm{Cpre}_i^{V,m} : 2^V \mapsto 2^V$, which constitutes an over-approximation of Cpre_i;
- the *must* operator $\mathrm{Cpre}_i^{V,M} : 2^V \mapsto 2^V$, which constitutes an under-approximation of Cpre_i.

We let, for $U \subseteq V$ and $i \in \{1, 2\}$:

$$\mathrm{Cpre}_i^{V,m}(U) = \mathrm{Cpre}_i(U\!\downarrow)\!\uparrow_V^m \qquad\qquad \mathrm{Cpre}_i^{V,M}(U) = \mathrm{Cpre}_i(U\!\downarrow)\!\uparrow_V^M. \tag{3}$$

By the results of [9], we have the duality

$$\mathrm{Cpre}_i^{V,M}(U) = V \setminus \mathrm{Cpre}_{\sim i}^{V,m}(V \setminus U). \tag{4}$$

The fact that $\mathrm{Cpre}_\cdot^{V,m}$ and $\mathrm{Cpre}_\cdot^{V,M}$ are over and under-approximations of the concrete predecessor operator is made precise by the following observation, which follows directly from Lemma 1: for all $U \subseteq V$ and $i \in \{1, 2\}$, we have

$$\mathrm{Cpre}_i^{V,M}(U)\!\downarrow\, \subseteq \mathrm{Cpre}^i(U\!\downarrow) \subseteq \mathrm{Cpre}_i^{V,m}(U)\!\downarrow . \tag{5}$$

2.5. μ-Calculus

We will express our algorithms for solving games on the abstract state space in μ-calculus notation [14]. Consider a function $\gamma : 2^V \mapsto 2^V$, monotone when 2^V is considered as a lattice with the usual subset ordering. We denote by $\mu Z.\gamma(Z)$ (resp. $\nu Z.\gamma(Z)$) the *least* (resp. *greatest*) *fixpoint* of γ, that is, the least (resp. greatest) set $Z \subseteq V$ such that $Z = \gamma(Z)$. As is well known, since V is finite, these fixpoints can be computed via Picard iteration: $\mu Z.\gamma(Z) = \lim_{n\to\infty} \gamma^n(\emptyset)$ and $\nu Z.\gamma(Z) = \lim_{n\to\infty} \gamma^n(V)$. In the solution of parity games we will make use of nested fixpoint operators, which can be evaluated by nested Picard iteration [14].

3. Reachability and safety games

We present our three-valued abstraction-refinement technique by applying it first to the simplest games: reachability and safety games. It is convenient to present the arguments first for reachability games; the results for safety games are then obtained by duality.

3.1. Reachability games

Our three-valued abstraction-refinement scheme for reachability proceeds as follows. We assume we are given a game $G = \langle S, \lambda, \delta \rangle$, together with an initial set $\theta \subseteq S$ and a final set $T \subseteq S$, and an abstraction V for G that is precise for T. The question to be decided is: $\theta \cap \langle 1 \rangle \diamond T = \emptyset$?

The algorithm proceeds as follows. Using the may and must predecessor operators, we compute respectively the set W_1^m of *may-winning* abstract states, and the set W_1^M of *must-winning* abstract states. If $W_1^m \cap \theta{\uparrow}_V^m = \emptyset$, then the algorithm answers the question No; if $W_1^M \cap \theta{\uparrow}_V^m \neq \emptyset$, then the algorithm answers the question Yes. Otherwise, we will show later in Lemma 3 that there is at least one abstract state v such that:

$$v \in (W_1^m \backslash W_1^M) \cap \mathrm{Cpre}_1^{V,m}(W_1^M). \tag{6}$$

Such a state lies at the border between W_1^M and W_1^m. Precisely, such a v it is in $\mathrm{Cpre}_1^{V,m}(W_1^M)$, but outside of $W_1^M = \mathrm{Cpre}_1^{V,M}(W_1^M)$, and it is thus a state in W_1^m that is just one $\mathrm{Cpre}_1^{V,m}$-step removed from W_1^M. The algorithm picks such an abstract state v satisfying (6), and splits it into two abstract states v_1 and v_2, where:

$$v_1 = v \cap \mathrm{Cpre}_1(W_1^M{\downarrow}) \qquad v_2 = v \backslash \mathrm{Cpre}_1(W_1^M{\downarrow}).$$

As a consequence of (6), we will show that $v_1, v_2 \neq \emptyset$. The algorithm is given in detail as Algorithm 1. We first state the partial correctness of the algorithm, postponing the analysis of its termination to Section 3.3.

Lemma 2. *After Step 3 of Algorithm 1, we have* $W_1^M{\downarrow} \subseteq \langle 1 \rangle \diamond T \subseteq W_1^m{\downarrow}$.

Proof. The result follows from (5), and from the monotonicity of the μ-calculus operators appearing in Steps 2 and 3 of Algorithm 1. □

Lemma 3. *If Step 7 of Algorithm 1 is reached, there is at least one region* $v \in (W_1^m \setminus W_1^M) \cap Cpre_1^{V,m}(W_1^M)$.

Algorithm 1 Three-valued abstraction refinement for reachability games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \backslash \emptyset}$ that is precise for T.
Output: Yes if $\theta \cap \langle 1 \rangle \diamond T \neq \emptyset$, and No otherwise.

```
1.  while true do
2.      $W_1^M := \mu Y.(T{\uparrow}_V^M \cup \mathrm{Cpre}_1^{V,M}(Y))$
3.      $W_1^m := \mu Y.(T{\uparrow}_V^m \cup \mathrm{Cpre}_1^{V,m}(Y))$
4.      if $W_1^m \cap \theta{\uparrow}_V^m = \emptyset$ then return No
5.      else if $W_1^M \cap \theta{\uparrow}_V^m \neq \emptyset$ then return Yes
6.      else
7.          choose $v \in (W_1^m \setminus W_1^M) \cap \mathrm{Cpre}_1^{V,m}(W_1^M)$
8.          let $v_1 := v \cap \mathrm{Cpre}_1(W_1^M{\downarrow})$ and $v_2 := v \setminus v_1$
9.          $V := (V \backslash \{v\}) \cup \{v_1, v_2\}$
10.     end if
11. end while
```

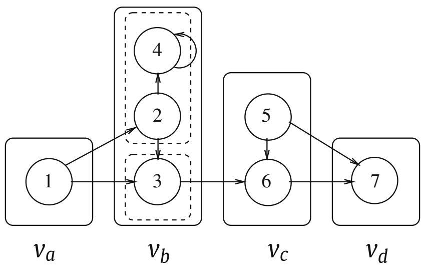

Fig. 1. Three-valued abstraction refinement in reachability game.

Proof. First, notice that since the algorithm did not terminate at Step 4 or Step 5, it must be $W_1^m \cap \theta{\uparrow}_V^m \neq \emptyset$ and $W_1^M \cap \theta{\uparrow}_V^m = \emptyset$, which by the previous lemma implies $W_1^M \subsetneq W_1^m$. From the fact that W_1^m is a least fixpoint, we have $W_1^m = \mu Y.(W_1^M \cup \mathrm{Cpre}_1^{V,m}(Y))$. Thus, there must be some $v \in W_1^m \backslash W_1^M$ with $v \in \mathrm{Cpre}_1^{V,m}(W_1^M)$. □

Lemma 4. *The sets v_1 and v_2 computed at Step 8 of Algorithm 1 are both non-empty.*

Proof. Consider $v \in W_1^m \backslash W_1^M$ with $v \in \mathrm{Cpre}_1^{V,m}(W_1^M)$. For $v_1 = v \cap \mathrm{Cpre}_1(W_1^M{\downarrow})$, we have $v_1 \neq \emptyset$, for otherwise $v_1 \notin \mathrm{Cpre}_1^{V,m}(W_1^M)$. Furthermore, we have $v_1 \subsetneq v$, for else we would have $v \in \mathrm{Cpre}_1^{V,M}(W_1^M)$, contradicting the fact that W_1^M is the fixpoint computed at Step 2. □

Theorem 5 (Partial correctness)**.** *Algorithm 1 can be executed without errors. Moreover:*

1. *if the algorithm terminates with answer Yes, then $\theta \cap \langle 1\rangle \diamond T \neq \emptyset$;*
2. *if the algorithm terminates with answer No, then $\theta \cap \langle 1\rangle \diamond T = \emptyset$.*

Proof. The only statement that could result in an error in the execution of Algorithm 1 is the choice of v at Step 7; Lemma 3 ensures that the error never arises. If the algorithm terminates at Step 4, the result follows from Lemma 2. If the algorithm terminates at Step 5, the result follows from the fact that $W_1^M{\downarrow} \cap \theta \neq \emptyset$, together with Lemma 2. □

Sufficient conditions for the termination of the algorithm are presented later, in Section 3.3.

Example. As an example, consider the game G illustrated in Fig. 1. The state space of the game is $S = \{1, 2, 3, 4, 5, 6, 7\}$, and the abstract state space is $V = \{v_a, v_b, v_c, v_d\}$, as indicated in the figure; the player-2 states are $S_2 = \{2, 3, 4\}$. We consider $\theta = \{1\}$ and $T = \{7\}$. After Steps 2 and 3 of Algorithm 1, we have $W_1^m = \{v_a, v_b, v_c, v_d\}$, and $W_1^M = \{v_c, v_d\}$. Therefore, the algorithm can answer neither No in Steps 4, nor Yes in Step 5, and proceeds to refine the abstraction. In Step 7, the only candidate for splitting is $v = v_b$, which is split into $v_1 = v_b \cap \mathrm{Cpre}_1(W_1^M{\downarrow}) = \{3\}$, and $v_2 = v_b \backslash v_1 = \{2, 4\}$. It is easy to see that at the next iteration of the analysis, v_1 and v_a are added to W_1^M, and the algorithm returns the answer Yes.

Discussion. We have assumed that the initial abstraction is precise for T. In practice, this is not a restrictive assumption, since the initial abstraction is usually obtained as one that can represent precisely the set of target states. Removing the assumption that T is represented precisely in the initial abstraction invalidates Lemma 3. Thus, Algorithm 1 can stop with $W_1^m \backslash W_1^M \neq \emptyset$, yet no region v as stipulated in Lemma 3 may exist. In this case, the abstraction of the target states T must be refined by splitting some region in $T{\uparrow}_V^m \backslash T{\uparrow}_V^M$.

An improved algorithm for reachability. Algorithm 1 can be improved by avoiding the full recomputation of the sets W_1^M and W_1^m at each abstraction refinement. Once we obtain v_1 and v_2 as in Step 8, we can set $W := W_1^M \cup \{v_1\}$, and we can compute for the next iteration:

$$W_1^M := \mu Y.(W \cup \mathrm{Cpre}_1^{V,M}(Y)) \qquad W_1^m := \mu Y.(W \cup \mathrm{Cpre}_1^{V,m}(Y))$$

The resulting algorithm is presented as Algorithm 2.

3.2. Safety games

We next consider a safety game specified by a target $T \subseteq S$, together with an initial condition $\theta \subseteq S$. Given an abstraction V that is precise for T, the goal is to answer the question of whether $\theta \cap \langle 1\rangle \Box T = \emptyset$. We note that the objectives $\Box T$ and $\diamond\neg T$ are dual, so that due to the determinacy of such games, we have $\langle 1\rangle \Box T = S \backslash \langle 2\rangle \diamond \neg T$.

Algorithm 2 Improved three-valued abstraction refinement for reachability games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for T.
Output: Yes if $\theta \cap \langle 1 \rangle \diamond T \neq \emptyset$, and No otherwise.

```
1.  $W := T\uparrow_V^M$
2.  **while** true **do**
3.      $W_1^M := \mu Y.(W \cup \text{Cpre}_1^{V,M}(Y))$
4.      $W_1^m := \mu Y.(W \cup \text{Cpre}_1^{V,m}(Y))$
5.      **if** $W_1^m \cap \theta\uparrow_V^m = \emptyset$ **then return** No
6.      **else if** $W_1^M \cap \theta\uparrow_V^m \neq \emptyset$ **then return** Yes
7.      **else**
8.          choose $v \in (W_1^m \setminus W_1^M) \cap \text{Cpre}_1^{V,m}(W_1^M)$
9.          let $v_1 := v \cap \text{Cpre}_1(W_1^M\downarrow)$ and $v_2 := v \setminus v_1$
10.         $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
11.         $W := W \cup \{v_1\}$
12.     **end if**
13. **end while**
```

As for reachability games, in safety games we begin by computing the set W_1^m of may-winning states, and the set W_1^M of must-winning states. Again, if $W_1^m \cap \theta\uparrow_V^m = \emptyset$, we answer No, and if $W_1^M \cap \theta\uparrow_V^m \neq \emptyset$, we answer Yes. In safety games, unlike in reachability games, we cannot split abstract states at the may-must boundary. For reachability games, a may-state can only win by reaching the goal T, which is contained in $W_1^M\downarrow$: hence, we refine the may-must border. In a safety game with objective $\Box T$, on the other hand, we have $W_1^m\downarrow \subseteq T$, and a state in $W_1^m\downarrow$ can be winning even if it never reaches $W_1^M\downarrow$ (which indeed can be empty if the abstraction is too coarse). To choose the location where to split, we exploit the duality between the safety goal $\Box T$ and reachability goal $\diamond\neg T$. In the game with goal $\diamond\neg T$, splitting occurs at the may-must boundary for $\langle 2 \rangle \diamond \neg T$, which is also the may-losing boundary $\langle 1 \rangle \Box T$. Hence, in safety games the splitting occurs at the may-losing boundary. This yields Algorithm 3.

Theorem 6 (Partial correctness). *Algorithm 3 can be executed without errors. Moreover:*

1. *if the algorithm terminates with answer Yes, then $\theta \cap \langle 1 \rangle \Box T \neq \emptyset$;*
2. *if the algorithm terminates with answer No, then $\theta \cap \langle 1 \rangle \Box T = \emptyset$.*

Proof. The theorem can be proved by noting that the goals $\Box T$ and $\diamond\neg T$ are dual, and by noting that from (4) we have:

$$\nu Y.(T\uparrow_V^M \cap \text{Cpre}_1^{V,M}(Y)) = V \setminus \mu Y.((S \setminus T)\uparrow_V^M \cup \text{Cpre}_1^{V,M}(Y))$$
$$\nu Y.(T\uparrow_V^m \cap \text{Cpre}_1^{V,m}(Y)) = V \setminus \mu Y.((S \setminus T)\uparrow_V^m \cup \text{Cpre}_1^{V,m}(Y)) \ .$$

Thus, the Algorithm 3 is the dual of Algorithm 1, and its correctness can be proved in analogous fashion. □

Algorithm 3 Three-valued abstraction refinement for safety games

Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for T.
Output: Yes if $\theta \cap \langle 1 \rangle \Box T \neq \emptyset$, and No otherwise.

```
1.  **while** true **do**
2.      $W_1^M := \nu Y.(T\uparrow_V^M \cap \text{Cpre}_1^{V,M}(Y))$
3.      $W_1^m := \nu Y.(T\uparrow_V^m \cap \text{Cpre}_1^{V,m}(Y))$
4.      **if** $W_1^m \cap \theta\uparrow_V^m = \emptyset$ **then return** No
5.      **else if** $W_1^M \cap \theta\uparrow_V^m \neq \emptyset$ **then return** Yes
6.      **else**
7.          choose $v \in (W_1^m \setminus W_1^M) \cap \text{Cpre}_2^{V,m}(V \setminus W_1^m)$
8.          let $v_1 := v \cap \text{Cpre}_2(S \setminus (W_1^m\downarrow))$ and $v_2 := v \setminus v_1$
9.          let $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
10.     **end if**
11. **end while**
```

We note that it is possible to obtain a more efficient version of Algorithm 3 by performing a dual transformation to the one that yielded Algorithm 2. Precisely, before Step 1, we let $W := (T\uparrow_V^m)$; the fixpoints at Steps 2 and 3 are computed via $W_1^M := \nu Y.(W \cap \mathrm{Cpre}_1^{V,M}(Y))$ and $W_1^m := \nu Y.(W \cap \mathrm{Cpre}_1^{V,m}(Y))$; and after Step 8 we set $W := W_1^m \setminus \{v_1\}$.

3.3. Termination

We present a condition that ensures termination of Algorithms 1 and 3 (and thus also Algorithm 2). The condition states that, if there is a finite algebra of regions (sets of concrete states) that is closed under Boolean operations and controllable-predecessor operators, and that is precise for the set of target states, then (i) Algorithms 1 and 3 terminate and (ii) the algorithms never produce abstract states that are finer than the regions of the algebra (guaranteeing that the algorithms do not perform unnecessary work). Formally, a *region algebra* for a game $G = \langle S, \lambda, \delta\rangle$ is an abstraction U such that:

- U is closed under Boolean operations: for all $u_1, u_2 \in U$, we have $u_1 \cup u_2 \in U$ and $S\setminus u_1 \in U$.
- U is closed under controllable-predecessor operators: for all $u \in U$, we have $\mathrm{Cpre}_1(u) \in U$ and $\mathrm{Cpre}_2(u) \in U$.

Theorem 7 (Termination)**.** *Consider a game G with a finite region algebra U. Assume that Algorithm 1 or 3 are called with arguments G, θ, T, with $T \in U$, and with an initial abstraction $V \subseteq U$. Then, the following assertions hold for both algorithms:*

1. *The algorithms, during their executions, produce abstract states that are all members of the algebra U.*
2. *The algorithms terminate.*

Proof. Let us prove the theorem for the reachability game. The proof for safety game can be easily obtained by duality.

First, note that due to the closure properties of the region algebra U, the algorithm computes entirely with regions in U: precisely, variables are only assigned regions of U. This yields the first assertion of the theorem.

The termination of Algorithm 1 can be proved by the following argument. At each refinement loop, the algorithm decreases the size of the uncertainty region $W_1^m \setminus W_1^M$, since the set v_1 computed in Step 8 will belong to W_1^M in the following iteration. As the region algebra U is finite, within a finite number of refinements the uncertainty region will be empty, and the algorithm will return either Yes or No. $\square$

Many games, including timed games, have the finite region algebras mentioned in the above theorem [10–12].

3.4. Approximate abstraction-refinement schemes

While the abstraction-refinement scheme above is fairly general, it makes two assumptions that may not hold in a practical implementation:

- it assumes that we can compute $\mathrm{Cpre}_i^{V,m}$ and $\mathrm{Cpre}_i^{V,M}$ of (3) precisely;
- it assumes that, once we pick an abstract state v to split, we can split it into v_1 and v_2 precisely, as outlined in Algorithms 1 and 3.

In fact, both assumptions can be relaxed, yielding a more widely applicable abstraction-refinement algorithm for two-player games. We present the modified algorithm for the reachability case only; the results can be easily extended to the dual case of safety objectives. Our starting point consists in approximate versions $\mathrm{Cpre}_i^{V,m+}, \mathrm{Cpre}_i^{V,M-} : 2^V \mapsto 2^V$ of the operators $\mathrm{Cpre}_i^{V,m}, \mathrm{Cpre}_i^{V,M}$, for $i \in \{1, 2\}$. We require that, for all $U \subseteq V$ and $i \in \{1, 2\}$, we have:

$$\mathrm{Cpre}_i^{V,m}(U) \subseteq \mathrm{Cpre}_i^{V,m+}(U) \qquad \mathrm{Cpre}_i^{V,M-}(U) \subseteq \mathrm{Cpre}_i^{V,M}(U)\ . \tag{7}$$

With these operators, we can phrase a new, approximate abstraction scheme for reachability, given in Algorithm 4. The use of the approximate operators means that, in Step 8, we can be no longer sure that both $v_1 \neq \emptyset$ and $v\setminus v_1 \neq \emptyset$. If the "precise" split of Step 8 fails, we resort instead to an arbitrary split (Step 10). The following theorem states that the algorithm essentially enjoys the same properties of the "precise" Algorithms 1 and 3.

Lemma 8. *At Step 4 of Algorithm 4, we have $W_1^{M-}\downarrow \subseteq \langle 1\rangle\diamond T \subseteq W_1^{m+}\downarrow$.*

Proof. From Eq. (7), we have : $\mathrm{Cpre}_i^{V,m}(U) \subseteq \mathrm{Cpre}_i^{V,m+}(U)$ and $\mathrm{Cpre}_i^{V,M-}(U) \subseteq \mathrm{Cpre}_i^{V,M}(U)$ for all $U \subseteq V$ and $i \in \{1, 2\}$. For reachability game the approximate and accurate must-winning set is obtained by the respective fixpoint formulas $W_1^{M-} = \mu Y.(T\uparrow_V^M \cup \mathrm{Cpre}_1^{V,M-}(Y)$ and $W_1^M = \mu Y.(T\uparrow_V^M \cup \mathrm{Cpre}_1^{V,M}(Y)$. Since we start with the same initial set and in each iteration $\mathrm{Cpre}_i^{V,M-}(Y) \subseteq \mathrm{Cpre}_i^{V,M}(Y)$ holds, it is obvious that $W_1^{M-} \subseteq W_1^M$. Similar arguments will prove that $W_1^m \subseteq W_1^{m+}$. After we combine these two results with Lemma 2, we obtain $W_1^{M-}\downarrow \subseteq \langle 1\rangle\diamond T \subseteq W_1^{m+}\downarrow$. $\square$

Algorithm 4 Approximate three-valued abstraction refinement for reachability games

```
Input: A concrete game structure $G = \langle S, \lambda, \delta \rangle$, a set of initial states $\theta \subseteq S$, a set of target states $T \subseteq S$, and an abstraction $V \subseteq 2^{2^S \setminus \emptyset}$ that is precise for $T$.
Output: Yes if $\theta \cap \langle 1 \rangle \diamond T \neq \emptyset$, and No otherwise.
1.  while true do
2.      $W_1^{M-} := \mu Y.(T{\uparrow}_V^M \cup \mathrm{Cpre}_1^{V,M-}(Y))$
3.      $W_1^{m+} := \mu Y.(T{\uparrow}_V^m \cup \mathrm{Cpre}_1^{V,m+}(Y))$
4.      if $W_1^{m+} \cap \theta{\uparrow}_V^m = \emptyset$ then return No
5.      else if $W_1^{M-} \cap \theta{\uparrow}_V^m \neq \emptyset$ then return Yes
6.      else
7.          choose $v \in (W_1^{m+} \setminus W_1^{M-}) \cap \mathrm{Cpre}_1^{V,m+}(W_1^{M-})$
8.          let $v_1 := v \cap \mathrm{Cpre}_1(W_1^{M-}{\downarrow})$
9.          if $v_1 = \emptyset$ or $v_1 = v$
10.             then split $v$ arbitrarily into non-empty $v_1$ and $v_2$
11.             else $v_2 = v \setminus v_1$
12.         end if
13.         let $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
14.     end if
15. end while
```

Theorem 9. *The following assertions hold.*

1. *Correctness. If Algorithm 4 terminates, it returns the correct answer.*
2. *Termination. Assume that Algorithm 4 is given as input a game G with a finite region algebra U, and arguments $\theta, T \in U$, as well as with an initial abstraction $V \subseteq U$. Assume also that the region algebra U is closed with respect to the operators $\mathrm{Cpre}_i^{V,M-}$ and $\mathrm{Cpre}_i^{V,m+}$, for $i \in \{1, 2\}$, and that Step 10 of Algorithm 4 splits the abstract states in regions in U. Then, (a) Algorithm 4 produces only abstract states in U in the course of its execution and (b) it terminates within finite number of refinements.*

Proof

1. The correctness can be proved as in Theorem 5, using Lemma 8.
2. (a) The fact that Algorithm 4 produces only regions in U follows from the closure of U, and by inspection of the operations performed by the algorithm. (b) The termination of Algorithm 4 follows again from the finiteness of U, and from the fact that at each iteration, the uncertainty region shrinks. $\square$

3.5. Comparison with counterexample-guided control

It is instructive to compare our three-valued refinement approach with the *counterexample-guided control* approach of [6]. In [6], an abstract game structure is constructed and analyzed. The abstract game contains *must* transitions for player 1, and *may* transitions for player 2. Every counterexample to the property (spoiling strategy for player 2) found in the abstract game is analyzed in the concrete game. If the counterexample is real, the property is disproved; If the counterexample is spurious, it is ruled out by refining the abstraction. The process continues until either the property is disproved, or no abstract counterexamples is found, proving the property.

The main advantage of our proposed three-valued approach over counterexample-guided control is, somewhat paradoxically, that we do not explicitly construct the abstract game. It was shown in [8,9] that, for a game abstraction to be fully precise, the *must* transitions should be represented as hyper-edges (an expensive representation, space-wise). In the counterexample-guided approach, instead, normal *must* edges are used: the abstract game representation incurs a loss of

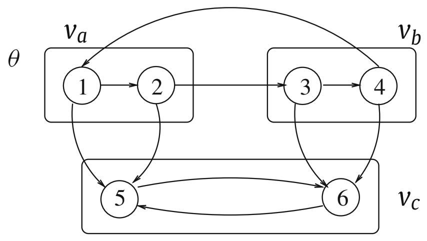

Fig. 2. Safety game, with objective $\square T$ for $T = \{1, 2, 3, 4\}$.

precision, and more abstraction-refinement steps may be needed than with our proposed three-valued approach. This is best illustrated with an example.

Example. Consider the game structure depicted in Fig. 2. The state space is $S = \{1, 2, 3, 4, 5, 6\}$, with $S_1 = \{1, 2, 3, 4\}$ and $S_2 = \{5, 6\}$; the initial states are $\theta = \{1, 2\}$. We consider the safety objective $\Box T$ for $T = \{1, 2, 3, 4\}$. We construct the abstraction $V = \{v_a, v_b, v_c\}$ precise for θ and T, as depicted. In the counterexample-guided control approach of [6], hyper-must transitions are not considered in the construction of the abstract model, and the transitions between v_a and v_b are lost: the only transitions from v_a and v_b lead to v_c. Therefore, there is a spurious abstract counterexample tree $v_a \to v_c$; ruling it out requires splitting v_a into its constituent states 1 and 2. Once this is done, there is another spurious abstract counterexample $2 \to v_b \to v_c$; ruling it out requires splitting v_b in its constituent states. In contrast, in our approach we have immediately $W_1^M = \{v_a, v_b\}$ and $v_a, v_b \in \mathrm{Cpre}_1^{V,M}(\{v_a, v_b\})$, so that no abstraction refinement is required.

The above example illustrates the fact that the counterexample-guided control approach of [6] may require a finer abstraction than our three-valued refinement approach, to prove a given property. On the other hand, it is easy to see that if an abstraction suffices to prove a property in the counterexample-guided control approach, it also suffices in our three-valued approach: the absence of abstract counterexamples translates directly in the fact that the states of interest are must-winning.

4. Symbolic implementation

We now present a concrete symbolic implementation of our abstraction scheme. We chose a simple symbolic representation for two-player games; while the symbolic game representations encountered in real verification systems (see, e.g. [16,17]) are usually more complex, the same principles apply.

4.1. Symbolic game structures

To simplify the presentation, we assume that all variables are Boolean. For a set X of Boolean variables, we denote by $\mathcal{F}(X)$ the set of propositional formulas constructed from the variables in X, the constants *true* and *false*, and the propositional connectives $\neg, \wedge, \vee, \to$. We denote with $\phi[\psi/x]$ the result of replacing all occurrences of the variable x in ϕ with a formula ψ. For $\phi \in \mathcal{F}(X)$ and $x \in X$, we write $\left\{ \begin{smallmatrix} \forall \\ \exists \end{smallmatrix} \right\} x.\phi$ for $\phi[true/x] \left\{ \begin{smallmatrix} \wedge \\ \vee \end{smallmatrix} \right\} \phi[false/x]$. We extend this notation to sets $Y = \{y_1, y_2, \ldots, y_n\}$ of variables, writing $\forall Y.\phi$ for $\forall y_1.\forall y_2.\cdots\forall y_n.\phi$, and similarly for $\exists Y.\phi$. For a set X of variables, we also denote by $X' = \{x' \mid x \in X\}$ the corresponding set of *primed* variables; for $\phi \in \mathcal{F}(X)$, we denote ϕ' the formula obtained by replacing every $x \in X$ with x'.

A *state* s over a set X of variables is a truth-assignment $s : X \mapsto \{\mathrm{T}, \mathrm{F}\}$ for the variables in X; we denote with $S[X]$ the set of all such truth assignments. Given $\phi \in \mathcal{F}(X)$ and $s \in S[X]$, we write $s \models \phi$ if ϕ holds when the variables in X are interpreted as prescribed by s, and we let $[\![\phi]\!]_X = \{s \in S[X] \mid s \models \phi\}$. Given $\phi \in \mathcal{F}(X \cup X')$ and $s, t \in S[X]$, we write $(s, t) \models \phi$ if ϕ holds when $x \in X$ has value $s(x)$, and $x' \in X'$ has value $t(x)$. When X, and thus the state space $S[X]$, are clear from the context, we equate informally formulas and sets of states. These formulas, or sets of states, can be manipulated with the help of symbolic representations such as BDDs [18]. A *symbolic game structure* $G_S = \langle X, \Lambda_1, \Delta \rangle$ consists of the following components:

- A set of Boolean variables X.
- A predicate $\Lambda_1 \in \mathcal{F}(X)$ defining when it is player 1's turn to play. We define $\Lambda_2 = \neg\Lambda_1$.
- A transition function $\Delta \in \mathcal{F}(X \cup X')$, such that for all $s \in S[X]$, there is some $t \in S[X]$ such that $(s, t) \models \Delta$.

A symbolic game structure $G_S = \langle X, \Lambda_1, \Delta \rangle$ induces a (concrete) game structure $G = \langle S, \lambda, \delta \rangle$ via $S = S[X]$, and for $s, t \in S$, $\lambda(s) = 1$ iff $s \models \Lambda_1$, and $t \in \delta(s)$ iff $(s, t) \models \Delta$. Given a formula $\phi \in \mathcal{F}(X)$, we have

$$\mathrm{Cpre}_1([\![\phi]\!]_X) = [\![(\Lambda_1 \wedge \exists X'.(\Delta \wedge \phi')) \vee (\neg\Lambda_1 \wedge \forall X'.(\Delta \to \phi'))]\!]_X.$$

4.2. Symbolic abstractions

We specify an abstraction for a symbolic game structure $G_S = \langle X, \Lambda_1, \Delta \rangle$ via a subset $X^a \subseteq X$ of its variables: the idea is that the abstraction keeps track only of the values of the variables in X^a; we denote by $X^c = X \backslash X^a$ the concrete-only variables. We assume that $\Lambda_1 \in \mathcal{F}(X^a)$, so that in each abstract state, only one of the two players can move (in other words, we consider *turn-preserving* abstractions [9]). With slight abuse of notation, we identify the abstract state space V with $S[X^a]$, where, for $s \in S[X]$ and $v \in V$, we let $s \in v$ iff $s(x) = v(x)$ for all $x \in X^a$. On this abstract state space, the operators $\mathrm{Cpre}_1^{V,m}$ and $\mathrm{Cpre}_1^{V,M}$ can be computed symbolically via the corresponding operators $\mathrm{SCpre}_1^{V,m}$ and $\mathrm{SCpre}_1^{V,M}$, defined as follows. For $\phi \in \mathcal{F}(X^a)$,

$$\mathrm{SCpre}_1^{V,m}(\phi) = \exists X^c.\Big((\Lambda_1 \wedge \exists X'.(\Delta \wedge \phi')) \vee (\Lambda_2 \wedge \forall X'.(\Delta \to \phi'))\Big) \quad (8)$$

$$\mathrm{SCpre}_1^{V,M}(\phi) = \forall X^c.\Big((\Lambda_1 \wedge \exists X'.(\Delta \wedge \phi')) \vee (\Lambda_2 \wedge \forall X'.(\Delta \to \phi'))\Big) \quad (9)$$

The above operators correspond exactly to (3). Alternatively, we can abstract the transition formula Δ, defining:

$$\Delta^m_{X^a} = \exists X^c.\exists X^{c\prime}.\Delta \qquad \Delta^M_{X^a} = \forall X^c.\exists X^{c\prime}.\Delta \ .$$

These abstract transition relations can be used to compute approximate versions $\mathrm{SCpre}_1^{V,m+}$ and $\mathrm{SCpre}_1^{V,M-}$ of the controllable-predecessor operators of (8), (9):

$$\mathrm{SCpre}_1^{V,m+}(\phi) = \Big((\Lambda_1 \wedge \exists X^{a\prime}.(\Delta^m_{X^a} \wedge \phi')) \vee (\Lambda_2 \wedge \forall X^{a\prime}.(\Delta^M_{X^a} \to \phi'))\Big)$$

$$\mathrm{SCpre}_1^{V,M-}(\phi) = \Big((\Lambda_1 \wedge \exists X^{a\prime}.(\Delta^M_{X^a} \wedge \phi')) \vee (\Lambda_2 \wedge \forall X^{a\prime}.(\Delta^m_{X^a} \to \phi'))\Big)$$

These operators, while approximate, satisfy the conditions (7), and can thus be used to implement symbolically Algorithm 4.

4.3. Symbolic abstraction refinement

We replace the abstraction-refinement step of Algorithms 1, 3, and 4 with a step that adds a variable $x \in X^c$ to the set X^a of variables present in the abstraction. The challenge is to choose a variable x that increases the precision of the abstraction in a useful way. To this end, we follow an approach inspired directly by [4].

Denote by $v \in S[X^a]$ the abstract state that Algorithms 4 chooses for splitting at Step 7, and let $\psi_1^{M-} \in \mathcal{F}(X^a)$ be the formula defining the set W_1^{M-} in the same algorithm. We choose $x \in X^c$ so that there are at least two states $s_1, s_2 \in v$ that differ only for the value of x, and such that $s_1 \models \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-})$ and $s_2 \not\models \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-})$. Thus, the symbolic abstraction-refinement algorithm first searches for a variable $x \in X^c$ for which the following formula is true:

$$\exists (X^c \backslash x).\Big(\big(\chi_v \to (x \equiv \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-}))\big) \vee \big(\chi_v \to (x \not\equiv \mathrm{SCpre}_1^{V,m+}(\psi_1^{M-}))\big)\Big),$$

where χ_v is the *characteristic formula* of v:

$$\chi_v = \bigwedge\{x \mid x \in X^a.v(x) = \mathrm{T}\} \ \wedge \ \bigwedge\{\neg x \mid x \in X^a.v(x) = \mathrm{F}\} \ .$$

If no such variable can be found, due to the approximate computation of $\mathrm{SCpre}_1^{V,m+}$ and $\mathrm{SCpre}_1^{V,M-}$, then $x \in X^c$ is chosen arbitrarily. The choice of variable for Algorithm 3 can be obtained by reasoning in dual fashion.

5. Conclusions

We have presented a technique for the verification of game properties based on the construction, three-valued analysis, and refinement of game abstractions. The approach is suitable for symbolic implementation, and can be implemented in a relatively straightforward manner. The key insight of the approach consists on relying on three-valued versions of the usual predecessor operators to analyze a system, avoiding the construction of a three-valued transition relation, which would require an exponential blow-up in the size of the abstract system to achieve comparable precision. The method, presented here for games, is equally suited to transition systems, where it constitutes an alternative to the classical counterexample-guided refinement technique of CEGAR [3–5].

References

[1] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints , in: Proceedings of the 4th ACM Symposium on Principles of Programming Languages, ACM Press, 1977, pp. 238–252.

[2] T. Henzinger, R. Majumdar, F. Mang, J.-F. Raskin, Abstract interpretation of game properties, in: SAS 00: Static Analysis, Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 220–239.

[3] R. Alur, A. Itai, R.P. Kurshan, M. Yannakakis, Timing verification by successive approximation, Information and Computation 118 (1) (1995) 142–157.

[4] E. Clarke, O. Grumberg, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: CAV 00: Proceedings of the 12th Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 154–169.

[5] T. Ball, S. Rajamani, The SLAM project: debugging system software via static analysis, in: Proceedings of the 29th Annual Symposium on Principles of Programming Languages, ACM Press, 2002, pp. 1–3.

[6] T. Henzinger, R. Jhala, R. Majumdar, Counterexample-guided control, in: 30th International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Computer Science, vol. 2719, 2003, pp. 886–902.

[7] S. Shoham, A game-based framework for CTL counter-examples and 3-valued abstraction-refinement, in: CAV 03: Proceedings of the 15th Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 275–287.

[8] S. Shoham, O. Grumberg, Monotonic abstraction-refinement for CTL, in: TACAS, Lecture Notes in Computer Science, vol. 2988, Springer-Verlag, 2004, pp. 546–560.
[9] L. de Alfaro, P. Godefroid, R. Jagadeesan, Three-valued abstractions of games: uncertainty, but with precision, in: Proceedings of the 19th IEEE Symposium Logic in Computer Science, 2004, pp. 170–179.
[10] L. de Alfaro, T. Henzinger, R. Majumdar, Symbolic algorithms for infinite-state games, in: CONCUR 01: 12th International Conference on Concurrency Theory, Lecture Notes in Computer Science, , Springer-Verlag, 2001, pp. 536–550.
[11] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 900, Springer-Verlag, 1995, pp. 229–242.
[12] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, M. Stoelinga, The element of surprise in timed games, in: CONCUR 03: 14th International Conference on Concurrency Theory, Lecture Notes in Computer Science, vol. 2761, Springer-Verlag, 2003, pp. 144–158.
[13] S. Shoham, O. Grumberg, 3-Valued abstraction: more precision at less cost, in: Proceedings of the 21st IEEE Symposium Logic in Computer Science, 2006, pp. 399–410.
[14] E. Emerson, C. Jutla, Tree automata, mu-calculus and determinacy (extended abstract), in: Proceedings of the 32nd IEEE Symposium Foundation of Computer Science, IEEE Computer Society Press, 1991, pp. 368–377.
[15] D. Martin, An extension of Borel determinacy, Annals of Pure and Applied Logic 49 (1990) 279–293.
[16] L. de Alfaro, R. Alur, R. Grosu, T. Henzinger, M. Kang, R. Majumdar, F. Mang, C. Meyer-Kirsch, B. Wang, Mocha: a model checking tool that exploits design structure, in: ICSE 01: Proceedings of the 23rd International Conference on Software Engineering, 2001, pp. 835–836.
[17] L. de Alfaro, L.D. da Silva, M. Faella, A. Legay, P. Roy, M. Sorea, Sociable interfaces, in: FROCOS: Proceedings of the 5th International Workshop on Frontiers of Combining Systems, Lecture Notes in Computer Science, vol. 3717, Springer-Verlag, 2005, pp. 81–105.
[18] R. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers C-35 (8) (1986) 677–691.