

# Solving Games via Three-Valued Abstraction Refinement\*

Luca de Alfaro and Pritam Roy

Computer Engineering Department  
University of California, Santa Cruz, USA

**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. Given a 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. We present the results for reachability and safety properties, as well as for fully general  $\omega$ -regular properties.

## 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. 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

---

\* This research was supported in part by the NSF grant CCR-0132780.

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 [2, 5, 3]. According to this technique, given a system abstraction, we check whether the abstraction 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* [12].

We propose here an alternative technique to CEGAR for refining game abstractions: namely, we propose to use *three-valued* analysis [16, 17, 9] in order to guide abstraction refinement for games. Our proposed 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 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, where the goal is to reach a set of target states, 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, where the goal is to stay always in a set of “safe” states, the refinement occurs at the may-never border. We show that the proposed abstraction refinement scheme can be uniformly extended to games with parity objectives: this enables the solution of games with arbitrary  $\omega$ -regular objectives, via automata-theoretic transformations [19].

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 [13, 8]. 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 [16, 17]. Differently from these approaches, however, we avoid the explicit construction of the tree-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 [17, 9, 18]; 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 for game analysis is superior to the CEGAR technique of [12], 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 [12] 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.

## 2 Preliminary Definitions

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

- A state space  $S$ .
- A turn function  $\lambda : S \rightarrow \{1, 2\}$ , 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  $\delta : S \mapsto 2^S \setminus \emptyset$ , 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  $\bar{s} = s_0, s_1, s_2, \dots$  of states in  $S$  such that for all  $k \geq 0$ , we have  $s_{k+1} \in \delta(s_k)$ . We denote by  $\Omega$  the set of all paths.

### 2.1 Game Objectives

An *objective*  $\Phi$  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*  $(G, \Phi)$  consists of a game structure  $G$  together with an objective  $\Phi$  for a player. We consider winning objectives that consist in  $\omega$ -regular conditions [19]; in particular, we will present algorithms for reachability, safety, and parity objectives. We often use linear-time temporal logic (LTL) notation [14] when defining objectives. Given a subset  $T \subseteq S$  of states, the *reachability* objective  $\diamond T = \{s_0, s_1, s_2, \dots \in S^\omega \mid \exists k \geq 0. s_k \in T\}$  consists of all paths that reach  $T$ ; the *safety* objective  $\square T = \{s_0, s_1, s_2, \dots \in S^\omega \mid \forall k \geq 0. s_k \in T\}$  consists of all paths that stay in  $T$  forever. We also consider *parity* objectives: the ability to solve games with parity objectives suffices for solving games with arbitrary LTL (or omega-regular) winning objectives [19]. A parity objective is specified via a partition  $\langle B_0, B_1, \dots, B_n \rangle$  of  $S$ , for some  $n \geq 0$ . Given a path  $\bar{s}$ , let  $\text{Infi}(\bar{s}) \subseteq S$  be the set of states that occur infinitely often along  $\bar{s}$ , and let  $\text{MaxCol}(\bar{s}) = \max\{i \in \{0, \dots, n\} \mid B_i \cap \text{Infi}(\bar{s}) \neq \emptyset\}$  be the index of the largest partition visited infinitely often along the path. Then,  $\varphi_n = \{\bar{s} \in \Omega \mid \text{MaxCol}(\bar{s}) \text{ is even}\}$ .

## 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 nonempty finite sequence  $\sigma$  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  $\pi_1, \pi_2$  for players 1 and 2 uniquely determine a sequence of states  $\text{Outcome}(s_0, \pi_1, \pi_2) = s_0, s_1, s_2, \dots$ , where for  $k > 0$  we have  $s_{k+1} = \pi_1(s_0, \dots, s_k)$  if  $s_k \in S_1$ , and  $s_{k+1} = \pi_2(s_0, \dots, s_k)$  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  $\pi_i$  such that, for all player  $\sim i$  strategies  $\pi_{\sim i}$ , we have  $\text{Outcome}(s_0, \pi_1, \pi_2) \in \Phi$ . 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 [11], as well as the determinacy result of [15], ensures that for all  $\omega$ -regular goals  $\Phi$  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  $V \subseteq 2^{S \setminus \emptyset}$  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:

$$U \downarrow = \bigcup_{u \in U} u \quad T \uparrow_V^m = \{v \in V \mid v \cap T \neq \emptyset\} \quad T \uparrow_V^M = \{v \in V \mid v \subseteq T\}$$

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  $T$ . 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  $\omega$ -regular winning conditions are commonly solved using *controllable predecessor operators*. We define the *player-1 controllable predecessor operator*  $\text{Cpre}_1 : 2^S \mapsto 2^S$  as follows, for all  $X \subseteq S$  and  $i \in \{1, 2\}$ :

$$\text{Cpre}_i(X) = \{s \in S_i \mid \delta(s) \cap X \neq \emptyset\} \cup \{s \in S_{\sim i} \mid \delta(s) \subseteq X\}. \quad (1)$$

Intuitively, for  $i \in \{1, 2\}$ , the set  $\text{Cpre}_i(X)$  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  $\text{Cpre}_i$ . As multiple concrete states may correspond to the same abstract state, we cannot compute, on the abstract state space, a precise analogous of  $\text{Cpre}_i$ . Thus, for player  $i \in \{1, 2\}$ , we define two abstract operators: the *may* operator  $\text{Cpre}_i^{V,m} : 2^V \mapsto 2^V$ , which constitutes an over-approximation of  $\text{Cpre}_i$ , and the *must* operator  $\text{Cpre}_i^{V,M} : 2^V \mapsto 2^V$ , which constitutes an under-approximation of  $\text{Cpre}_i$  [9]. We let, for  $U \subseteq V$  and  $i \in \{1, 2\}$ :

$$\text{Cpre}_i^{V,m}(U) = \text{Cpre}_i(U \downarrow) \uparrow_V^m \quad \text{Cpre}_i^{V,M}(U) = \text{Cpre}_i(U \downarrow) \uparrow_V^M. \quad (2)$$

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

$$\text{Cpre}_i^{V,M}(U) = V \setminus \text{Cpre}_{\sim_i}^{V,m}(V \setminus U).$$

The fact that  $\text{Cpre}_i^{V,m}$  and  $\text{Cpre}_i^{V,M}$  are over and under-approximations of the concrete predecessor operator is made precise by the following observation: for all  $U \subseteq V$  and  $i \in \{1, 2\}$ , we have  $\text{Cpre}_i^{V,M}(U) \downarrow \subseteq \text{Cpre}_i(U \downarrow) \subseteq \text{Cpre}_i^{V,m}(U) \downarrow$ .

## 2.5 $\mu$ -Calculus

We will express our algorithms for solving games on the abstract state space in  $\mu$ -calculus notation [11]. 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  $\gamma$ , 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:  $\mu Z.\gamma(Z) = \lim_{n \rightarrow \infty} \gamma^n(\emptyset)$  and  $\nu Z.\gamma(Z) = \lim_{n \rightarrow \infty} \gamma^n(V)$ . In the solution of parity games we will make use of nested fixpoint operators, which can be evaluated by nested Picard iteration [11].

## 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  $\theta$  and  $T$ . 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, the algorithm picks an abstract state  $v$  such that

$$v \in (W_1^m \setminus W_1^M) \cap \text{Cpre}_1^{V,m}(W_1^M). \quad (3)$$

Such a state lies at the border between  $W_1^M$  and  $W_1^m$ . The state  $v$  is split into two abstract states  $v_1$  and  $v_2$ , where:

$$v_1 = v \cap \text{Cpre}_1(W_1^M \downarrow) \quad v_2 = v \setminus \text{Cpre}_1(W_1^M \downarrow).$$

As a consequence of (3), we have 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 1** *At Step 4 of Algorithm 1, we have  $W_1^M \downarrow \subseteq \langle 1 \rangle \diamond T \subseteq W_1^m \downarrow$ .*

---

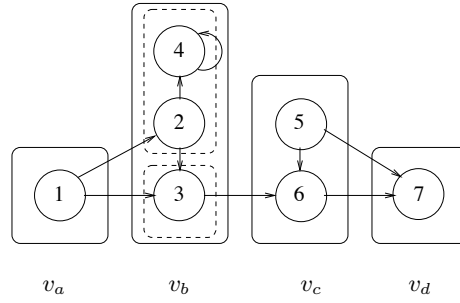
**Algorithm 1** 3-valued Abstraction Refinement for Reachability Games

---

**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  $V \subseteq 2^{2^S \setminus \theta}$  that is precise for  $\theta$  and  $T$ .

**Output:** Yes if  $\theta \cap \langle 1 \rangle \diamond T \neq \emptyset$ , and No otherwise.

1. **while true do**
2.      $W_1^M := \mu Y. (T \uparrow_V^M \cup \text{Cpre}_1^{V,M}(Y))$
3.      $W_1^m := \mu Y. (T \uparrow_V^m \cup \text{Cpre}_1^{V,m}(Y))$
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  $v \in (W_1^m \setminus W_1^M) \cap \text{Cpre}_1^{V,m}(W_1^M)$
8.         let  $v_1 := v \cap \text{Cpre}_1(W_1^M \downarrow)$  and  $v_2 := v \setminus v_1$
9.         let  $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
10.     **end if**
11. **end while**



**Fig. 1.** Three-Valued Abstraction Refinement in Reachability Game

**Theorem 1 (partial correctness)** *If Algorithm 1 terminates, it returns the correct answer.*

*Example 1.* As an example, consider the game  $G$  illustrated in Figure 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  $v_1 = v_b \cap \text{Cpre}_1(W_1^M \downarrow) = \{3\}$ , and  $v_2 = v_b \setminus v_1 = \{2, 4\}$ . 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. ■

### 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  $T$  and  $\theta$ , the goal is to

---

**Algorithm 2** 3-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  $\theta$  and  $T$ .

**Output:** Yes if  $\theta \cap \langle 1 \rangle \square T \neq \emptyset$ , 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**
- 

answer the question of whether  $\theta \cap \langle 1 \rangle \square T = \emptyset$ . As for reachability 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  $\square T$ , 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). Thus, to solve safety games, we split abstract states at the may-losing boundary, that is, at the boundary between  $W_1^m$  and its complement. This can be explained by the fact that  $\langle 1 \rangle \square T = S \setminus \langle 2 \rangle \diamond \neg T$ : the objectives  $\square T$  and  $\diamond \neg T$  are dual. Therefore, we adopt for  $\square T$  the same refinement method we would adopt for  $\diamond \neg T$ , and the may-must boundary for  $\langle 2 \rangle \diamond \neg T$  is the may-losing boundary for  $\langle 1 \rangle \square T$ . The algorithm is given below.

**Lemma 2** *At Step 4 of Algorithm 2, we have  $W_1^M \downarrow \subseteq \langle 1 \rangle \square T \subseteq W_1^m \downarrow$ .*

**Theorem 2 (partial correctness)** *If Algorithm 2 terminates, it returns the correct answer.*

### 3.3 Termination

We present a condition that ensures termination of Algorithms 1 and 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 sets of initial and target states, then (i) Algorithms 1 and 2 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  $\text{Cpre}_1(u) \in U$  and  $\text{Cpre}_2(u) \in U$ .

**Theorem 3 (termination)** *Consider a game  $G$  with a finite region algebra  $U$ . Assume that Algorithm 1 or 2 are called with arguments  $G, \theta, T$ , with  $\theta, T \in U$ , 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.*

The proof of the results is immediate. Many games, including timed games, have the finite region algebras mentioned in the above theorem [13, 10].

### 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  $\text{Cpre}_*^{V,m}$  and  $\text{Cpre}_*^{V,M}$  of (2) 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 2.

In fact, both assumptions can be related, 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  $\text{Cpre}_i^{V,m+}, \text{Cpre}_i^{V,M-} : 2^V \mapsto 2^V$  of the operators  $\text{Cpre}_i^{V,m}, \text{Cpre}_i^{V,M}$ , for  $i \in \{1, 2\}$ . We require that, for all  $U \subseteq V$  and  $i \in \{1, 2\}$ , we have:

$$\text{Cpre}_i^{V,m}(U) \subseteq \text{Cpre}_i^{V,m+}(U) \quad \text{Cpre}_i^{V,M-}(U) \subseteq \text{Cpre}_i^{V,M}(U) . \quad (4)$$

With these operators, we can phrase a new, approximate abstraction scheme for reachability, given in Algorithm 3. 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 2.

**Theorem 4** *The following assertions hold.*

1. *Correctness. If Algorithm 3 terminates, it returns the correct answer.*
2. *Termination. Assume that Algorithm 3 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  $\text{Cpre}_i^{V,M-}$  and  $\text{Cpre}_i^{V,m+}$ , for  $i \in \{1, 2\}$ , and that Step 10 of Algorithm 3 splits the abstract states in regions in  $U$ . Then, Algorithm 3 terminates, and it produces only abstract states in  $U$  in the course of its execution.*

---

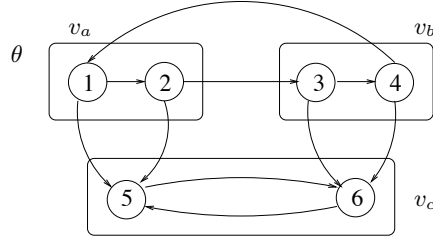
**Algorithm 3** Approximate 3-valued Abstraction Refinement for Reachability Games

---

**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  $V \subseteq 2^{2^S \setminus \theta}$  that is precise for  $\theta$  and  $T$ .

**Output:** Yes if  $\theta \cap \langle 1 \rangle \diamond T \neq \emptyset$ , and No otherwise.

1. **while true do**
2.      $W_1^{M-} := \mu Y. (T \uparrow_V^M \cup \text{Cpre}_1^{V, M-}(Y))$
3.      $W_1^{m+} := \mu Y. (T \uparrow_V^m \cup \text{Cpre}_1^{V, m+}(Y))$
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  $v \in (W_1^{m+} \setminus W_1^{M-}) \cap \text{Cpre}_1^{V, m+}(W_1^{M-})$
8.         let  $v_1 := v \cap \text{Cpre}_1(W_1^{M-} \downarrow)$
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**



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

### 3.5 Comparison with Counterexample-Guided Control

It is instructive to compare our three-valued refinement approach with the *counterexample-guided control* approach of [12]. In [12], 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 [17, 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 precision, and more abstraction refinement steps may be needed than with our proposed three-valued approach. This is best illustrated with an example.

*Example 2.* Consider the game structure depicted in Figure 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  $\theta$  and  $T$ , as depicted. In the counterexample-guided control approach of [12], 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  $v_a \rightarrow v_c$ ; ruling it out requires splitting  $v_a$  into its constituent states 1 and 2. Once this is done, there is another spurious abstract counterexample  $2 \rightarrow v_b \rightarrow v_c$ ; 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  $v_a, v_b \in \text{Cpre}_1^{V,M}(\{v_a, v_b\})$ , so that no abstraction refinement is required. ■

The above example illustrates that the counterexample-guided control approach of [12] 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., [6, 7]) 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  $\neg, \wedge, \vee, \rightarrow$ . We denote with  $\phi[\psi/x]$  the result of replacing all occurrences of the variable  $x$  in  $\phi$  with a formula  $\psi$ . For  $\phi \in \mathcal{F}(X)$  and  $x \in X$ , we write  $\{\exists\}x.\phi$  for  $\phi[\text{true}/x]\{\wedge\}\phi[\text{false}/x]$ . We extend this notation to sets  $Y = \{y_1, y_2, \dots, y_n\}$  of variables, writing  $\forall Y.\phi$  for  $\forall y_1.\forall y_2.\dots.\forall y_n.\phi$ , 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  $\phi'$  the formula obtained by replacing every  $x \in X$  with  $x'$ .

A *state*  $s$  over a set  $X$  of variables is a truth-assignment  $s : X \mapsto \{\text{T}, \text{F}\}$  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  $\phi$  holds when the variables in  $X$  are interpreted as prescribed by  $s$ , and we let  $\llbracket \phi \rrbracket_X = \{s \in S[X] \mid s \models \phi\}$ . Given  $\phi \in \mathcal{F}(X \cup X')$  and  $s, t \in S[X]$ , we write  $(s, t) \models \phi$  if  $\phi$  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 [4]. 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

$$\text{Cpre}_1(\llbracket \phi \rrbracket_X) = \llbracket (\Lambda_1 \wedge \exists X'. (\Delta \wedge \phi')) \vee (\neg\Lambda_1 \wedge \forall X'. (\Delta \rightarrow \phi')) \rrbracket_X.$$

## 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  $X^c = X \setminus X^a$  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  $\text{Cpre}_1^{V,m}$  and  $\text{Cpre}_1^{V,M}$  can be computed symbolically via the corresponding operators  $\text{SCpre}_1^{V,m}$  and  $\text{SCpre}_1^{V,M}$ , defined as follows. For  $\phi \in \mathcal{F}(X^a)$ ,

$$\text{SCpre}_1^{V,m}(\phi) = \exists X^c. \left( (\Lambda_1 \wedge \exists X'. (\Delta \wedge \phi')) \vee (\Lambda_2 \wedge \forall X'. (\Delta \rightarrow \phi')) \right) \quad (5)$$

$$\text{SCpre}_1^{V,M}(\phi) = \forall X^c. \left( (\Lambda_1 \wedge \exists X'. (\Delta \wedge \phi')) \vee (\Lambda_2 \wedge \forall X'. (\Delta \rightarrow \phi')) \right) \quad (6)$$

The above operators correspond exactly to (2). Alternatively, we can abstract the transition formula  $\Delta$ , defining:

$$\Delta_{X^a}^m = \exists X^c. \exists X^{c'}. \Delta \quad \Delta_{X^a}^M = \forall X^c. \exists X^{c'}. \Delta.$$

These abstract transition relations can be used to compute approximate versions  $\text{SCpre}_1^{V,m+}$  and  $\text{SCpre}_1^{V,M-}$  of the controllable predecessor operators of (5), (6):

$$\text{SCpre}_1^{V,m+}(\phi) = \left( (\Lambda_1 \wedge \exists X^{a'}. (\Delta_{X^a}^m \wedge \phi')) \vee (\Lambda_2 \wedge \forall X^{a'}. (\Delta_{X^a}^m \rightarrow \phi')) \right)$$

$$\text{SCpre}_1^{V,M-}(\phi) = \left( (\Lambda_1 \wedge \exists X^{a'}. (\Delta_{X^a}^M \wedge \phi')) \vee (\Lambda_2 \wedge \forall X^{a'}. (\Delta_{X^a}^M \rightarrow \phi')) \right)$$

These operators, while approximate, satisfy the conditions (4), and can thus be used to implement symbolically Algorithm 3.

---

**Algorithm 4** 3-valued Abstraction Refinement for Parity Games

---

**Input:** A concrete game structure  $G = \langle S, \lambda, \delta \rangle$ , a set of initial states  $\theta \subseteq S$ , a parity condition  $\varphi = \langle B_0, B_1, \dots, B_n \rangle$ , and an abstraction  $V \subseteq 2^{2^S \setminus \emptyset}$  that is precise for  $\theta, B_0, \dots, B_n$ .

**Output:** Yes if  $\theta \cap \langle 1 \rangle \varphi \neq \emptyset$ , and No otherwise.

1. **while true do**
  2.      $W_1^M := \text{Win}(\text{Cpre}_1^{V,M}, \emptyset, n)$
  3.      $W_1^m := \text{Win}(\text{Cpre}_1^{V,m}, \emptyset, n)$
  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  $(v, v_1, v_2)$  from  $\text{Split}(V, W_1^m, W_1^M, n)$
  11.          $V = (V \setminus \{v\}) \cup \{v_1, v_2\}$
  13.     **end if**
  14. **end while**
- 

### 4.3 Symbolic Abstraction Refinement

We replace the abstraction refinement step of Algorithms 1, 2, and 3 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 [5].

Denote by  $v \in S[X^a]$  the abstract state that Algorithms 3 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  $s_1 \models \text{SCpre}_1^{V,m+}(\psi_1^{M-})$  and  $s_2 \not\models \text{SCpre}_1^{V,m+}(\psi_1^{M-})$ . Thus, the symbolic abstraction refinement algorithm first searches for a variable  $x \in X^c$  for which the following formula is true:

$$\exists (X^c \setminus x). \left( \left( \chi_v \rightarrow (x \equiv \text{SCpre}_1^{V,m+}(\psi_1^{M-})) \right) \vee \left( \chi_v \rightarrow (x \not\equiv \text{SCpre}_1^{V,m+}(\psi_1^{M-})) \right) \right),$$

where  $\chi_v$  is the *characteristic formula* of  $v$ :

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

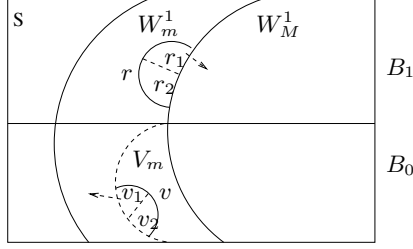
If no such variable can be found, due to the approximate computation of  $\text{SCpre}_1^{V,m+}$  and  $\text{SCpre}_1^{V,M-}$ , then  $x \in X^c$  is chosen arbitrarily. The choice of variable for Algorithm 2 can be obtained by reasoning in dual fashion.

## 5 Abstraction Refinement for Parity Games

We now present a general abstraction-refinement algorithm to solve a  $n$ -color parity game where the state-space  $S$  is partitioned into  $n$  disjoint subsets  $B_0, B_1, \dots, B_n$ . Denoting the parity condition  $\langle B_0, \dots, B_n \rangle$  by  $\varphi$ , the winning states can be computed as follows [11]:

$$\langle 1 \rangle \varphi = \Upsilon_n Y_n \dots \nu Y_0. \left( (B_0 \cap \text{Cpre}_1(Y_0)) \cup \dots \cup (B_n \cap \text{Cpre}_1(Y_n)) \right),$$





**Fig. 3.** Abstraction refinement for co-Büchi games

To compute the candidate splits in  $B_0$ , the algorithm considers a safety game with goal  $\square B_0$ , with  $W_1^M$  as set of states that are already considered to be winning; the may-winning states in this game are  $V_m = \nu Y_0. (W_1^M \cup (B_0 \cap \text{Cpre}_1^{V,m}(Y_0)))$ . Thus, the algorithm computes the following candidate splits in  $B_0$ :

$$P_0 = \{(v, v_1, v_2) \mid v \in B_0 \cap (V_m \setminus W_1^M) \cap \text{Cpre}_2^{V,m}(V \setminus V_m), \\ v_1 = v \cap \text{Cpre}_2(V \setminus V_m), v_2 = v \setminus v_1\}.$$

The function Split returns  $P_1 \cup P_0$  as the set of candidate splits for the given co-Büchi game. ■

**Lemma 3** *At Step 4 of Algorithm 4, we have  $W_1^M \downarrow \subseteq \langle 1 \rangle \varphi \subseteq W_1^m \downarrow$ .*

**Theorem 5** *If Algorithm 4 terminates, it returns the correct answer. Moreover, consider a game  $G$  with a finite region algebra  $U$ . Assume that Algorithm 4 is called with an initial abstraction  $V \subseteq U$ . Then, the algorithm terminates, and during its execution, it produces abstract states that are all members of the algebra  $U$ .*

## 6 Conclusion and Future Work

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, being based entirely on the evaluation of predecessor operators, is simple both to present and to implement. We plan to implement the approach as part of the Ticc toolset of interface composition and analysis [1], applying it both to the untimed interface composition problem (which requires solving safety games), and to the timed interface composition problem (which requires solving 3-color parity games).

## References

1. B. Adler, L. de Alfaro, L. D. D. Silva, M. Faella, A. Legay, V. Raman, and P. Roy. TICC: a tool for interface compatibility and composition. In *CAV 06: Proc. of 18th Conf. on Computer Aided Verification*, volume 4144 of *Lect. Notes in Comp. Sci.*, pages 59–62. Springer-Verlag, 2006.

2. R. Alur, A. Itai, R. P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Inf. Comput.*, 118(1):142–157, 1995.
3. T. Ball and S. Rajamani. The SLAM project: Debugging system software via static analysis. In *Proceedings of the 29th Annual Symposium on Principles of Programming Languages*, pages 1–3. ACM Press, 2002.
4. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.
5. E. Clarke, O. Grumberg, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV 00: Proc. of 12th Conf. on Computer Aided Verification*, Lect. Notes in Comp. Sci. Springer-Verlag, 2000.
6. L. de Alfaro, R. Alur, R. Grosu, T. Henzinger, M. Kang, R. Majumdar, F. Mang, C. Meyer-Kirsch, and B. Wang. Mocha: A model checking tool that exploits design structure. In *ICSE 01: Proceedings of the 23rd International Conference on Software Engineering*, pages 835–836, 2001.
7. L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea. Sociable interfaces. In *FRODOS: Frontiers of Combining Systems, Proc. of the 5th Intl. Workshop*, volume 3717 of *Lect. Notes in Comp. Sci.*, pages 81–105. Springer-Verlag, 2005.
8. L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR 03: Concurrency Theory. 14th Int. Conf.*, volume 2761 of *Lect. Notes in Comp. Sci.*, pages 144–158. Springer-Verlag, 2003.
9. L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *Proc. 19th IEEE Symp. Logic in Comp. Sci.*, pages 170–179, 2004.
10. L. de Alfaro, T. Henzinger, and R. Majumdar. Symbolic algorithms for infinite-state games. In *CONCUR 01: Concurrency Theory. 12th Int. Conf.*, Lect. Notes in Comp. Sci. Springer-Verlag, 2001.
11. E. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proc. 32nd IEEE Symp. Found. of Comp. Sci.*, pages 368–377. IEEE Computer Society Press, 1991.
12. T. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *30th Int. Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2719, pages 886–902. LNCS, 2003.
13. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proc. of 12th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 900 of *Lect. Notes in Comp. Sci.*, pages 229–242. Springer-Verlag, 1995.
14. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
15. D. Martin. An extension of Borel determinacy. *Annals of Pure and Applied Logic*, 49:279–293, 1990.
16. S. Shoham. A game-based framework for CTL counter-examples and 3-valued abstraction-refinement. In *CAV 03: Proc. of 15th Conf. on Computer Aided Verification*, Lect. Notes in Comp. Sci., pages 275–287. Springer-Verlag, 2003.
17. S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *TACAS*, volume 2988 of *Lect. Notes in Comp. Sci.*, pages 546–560. Springer-Verlag, 2004.
18. S. Shoham and O. Grumberg. 3-valued abstraction: More precision at less cost. In *Proc. 21st IEEE Symp. Logic in Comp. Sci.*, pages 399–410, 2006.
19. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 135–191. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.