Solving Games Via Three-Valued Abstraction Refinement

Georg Neis
Advisor: Rayna Dimitrova

July 22, 2008

1 Introduction

(Infinite) games have many applications in verification and related areas. For instance, they are often used to model the interaction between a software component and its environment. Unfortunately, the state space of game structures for real life problems may be so huge that it is no longer feasible to work on them directly. The usual approach to tackle this is to work on abstractions of the original game structures. A popular technique for solving infinite two-player games according to this principle is counter-example guided abstraction refinement (CEGAR), as presented in [1]. [2] proposes an alternative approach for abstraction refinement, using a technique called *three-valued analysis*.

2 Games

Before talking about abstractions we recapitulate the basic terminology for infinite two-player games.

Definition 1. *A* game structure *is a triple $G = (S, \lambda, \delta)$ where S is the set of states (the* state space*), $\lambda : S \to \{1, 2\}$ the* turn function*, and $\delta : S \to 2^S \setminus \emptyset$ the* transition function*.*

Definition 2. *A* game objective *is an ω-regular language $\Phi \subseteq S^\omega$ over the alphabet S.*

Definition 3. *A* game *is a triple (G, I, Φ) consisting of a game structure, a set of initial states, and a game objective for player 1.*

The turn function determines which of the two players' turn it is when the game is in a particular state. That is, it partitions the state space into a set of player-1 states and a set of player-2 states: $S = S_1 \uplus S_2$.

Intuitively, a game starts by player 1 placing a token on an initial state. Then the player whose turn it is moves the token to one of the successor states, which in turn will determine who will do the next move. Since every state has a successor, this will go on forever. Player 1 wins if the infinite sequence of states that are visited satisfies (is an element of) the game objective. Otherwise player 2 wins.

Here we consider three particular classes of game objectives.

- The *reachability objective* for $T \subseteq S$ is written as $\Diamond T$ and denotes $\{\sigma \in S^\omega \mid \exists k \geq 0.\sigma[k] \in T\}$. Here the goal is to eventually visit a state in T.
- The *safety objective* for $T \subseteq S$ is written as $\Box T$ and denotes $\{\sigma \in S^\omega \mid \forall k \geq 0.\sigma[k] \in T\}$. Here the goal is never to visit a state not in T.
- The *parity objective* for a partition $(B_1, ..., B_n)$ of S is written as $\varphi_{(B_1,...,B_n)}$ and denotes $\{\sigma \in S^\omega \mid \max\{i \mid B_i \cap \inf(\sigma) \neq \emptyset\} \text{ is even}\}$.

The following definitions are relative to a particular given game structure (S, λ, δ).

Definition 4. *A* strategy *for player $i \in \{1, 2\}$ is a function $\pi_i : S^* \times S_i \to S$.*

Definition 5. *The* outcome *of a state s and strategies π_1, π_2, denoted by $outcome(s, \pi_1, \pi_2)$, is the uniquely determined sequence of states $\sigma \in S^\omega$ such that $\forall k \geq 0.\ \sigma[k] \in S_i \implies \sigma[k+1] = \pi_i(\sigma[0..k])$. It results from starting at state s and then following the strategies.*

Definition 6. *A state s is* winning *for player i with objective Φ iff*

$$\exists \pi_i.\forall \pi_{\bar{i}}.outcome(s, \pi_1, \pi_2) \in \Phi.$$

where $\bar{1} \overset{def}{=} 2$ and $\bar{2} \overset{def}{=} 1$. The set of player i's winning states is denoted by

$$\langle i \rangle \Phi := \{s \in S \mid s \textit{ is winning for player } i \textit{ with objective } \Phi\}.$$

Definition 7. *The* controllable predecessors operator *for player $i \in \{1, 2\}$ is the function $\mathrm{cpre}_i : 2^S \to 2^S$ defined as follows:*

$$\mathrm{cpre}_i(T) \overset{def}{=} \{s \in S_i \mid \delta(s) \cap T \neq \emptyset\} \cup \{s \in S_{\bar{i}} \mid \delta(s) \subseteq T\}$$

$\mathrm{cpre}_i(T)$ is the set of those states from which player i can force the game in one step to a state in T. It consists of the predecessors of T that belong to player i and of the states of his opponent that have no successor outside T. In the first case player i can just move to a successor in T because it is his turn. In the second case it's player 2's turn but he cannot avoid moving to a state in T.

Given a game (G, I, Φ), we want to know whether or not player 1 can win, i.e., we want to decide whether $I \cap \langle 1 \rangle \Phi$ is empty.

3 Abstractions

Definition 8. *An* abstraction *of a game structure $G = (S, \lambda, \delta)$ is a set $V \subseteq 2^S \setminus \{\emptyset\}$ of* abstract states *such that $\bigcup V = S$.*

In other words, each abstract state is a nonempty set of concrete states and every concrete state is contained in at least one abstract state.

The following definitions are relative to a particular given abstraction V.

Definition 9. *Given a set $U \subseteq V$ of abstract states, the* conretization *of U is the set of concrete states corresponding to U. It is written as $U\!\downarrow$ and defined as $\bigcup_{u \in U} u$.*

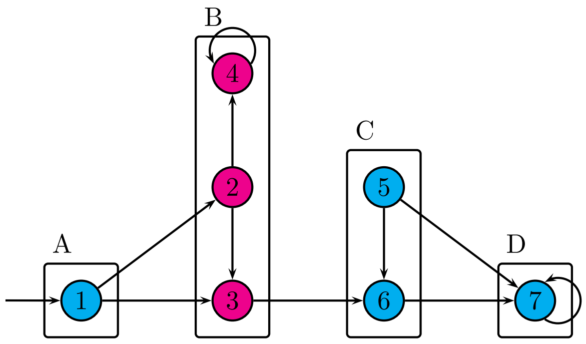

Figure 1: An abstraction of a game structure

Definition 10. *Given a set of concrete states $T \subseteq V$, the* under-approximation *of T is written T^{under} and defined as $\{v \in V \mid v \subseteq T\}$.*

Definition 11. *Given a set of concrete states $T \subseteq V$, the* over-approximation *of T is written T^{over} and defined as $\{v \in V \mid v \cap T \neq \emptyset\}$.*

Definition 12. *An abstraction is* precise *for $T \subseteq S$ iff $T^{\text{under}} = T^{\text{over}}$.*

For any $T \subseteq S$ we have $T^{\text{under}}\!\downarrow \subseteq T \subseteq T^{\text{over}}\!\downarrow$. Consequently, if V is precise for T, then $T^{\text{under}}\!\downarrow = T = T^{\text{over}}\!\downarrow$.

Figure 1 gives a sample game structure and one possible abstraction of it. The concrete state space is $\{1, 2, 3, 4, 5, 6, 7\}$, where $\{2, 3, 4\}$ is controlled by player 2 and the rest by player 1. Its abstraction consists of four states: $A = \{1\}$, $B = \{5, 6\}$, $C = \{2, 3, 4\}$, and $D = \{7\}$. It is precise for the set of initial states $\{1\}$ and for $\{7\}$. For instance, we have $\text{cpre}_1(\{5, 6, 7\}) = \{3, 5, 6, 7\}$, $\langle 1\rangle\Diamond\{7\} = \{1, 3, 5, 6, 7\}$, $\{B, C\}\!\downarrow = \{2, 3, 4, 5, 6\}$, $\{6\}^{\text{under}} = \emptyset$, $\{3, 5, 6\}^{\text{under}} = \{C\}$, $\{6\}^{\text{over}} = \{C\}$, and $\{3, 5, 6\}^{\text{over}} = \{B, C\}$.

4 Three-valued Abstraction Refinement

How to obtain an abstraction that is *precise enough* to answer a question like "can player 1 win"? The general approach is *abstraction refinement*, that is, the use of an algorithm that takes an initial abstraction and refines it until it becomes sufficiently precise. One popular technique for this is *counter-example guided control* (see [3]), where a counter-example in the abstract game is analyzed to yield either a counter-example in the concrete game (in which case the property in question does not hold) or a refined abstraction. If the refined abstraction contains no other counter-example, then the property holds. Otherwise, this counter-example will be analyzed in the same way.

In [2], de Alfaro and Roy propose an alternative approach, *three-valued abstraction refinement*, which works as follows. Given an abstraction that is precise for the set of initial states, we compute the (abstract) *must-win* states, *may-win* states, and *never-win* states for player 1. If the may-win states do not contain

```
while true do
    $W_{\text{must}} := \mu Y.(T^{\text{under}} \cup \text{cpre}_1^{\text{under}}(Y))$
    $W_{\text{may}} := \mu Y.(T^{\text{over}} \cup \text{cpre}_1^{\text{over}}(Y))$
    if $W_{\text{may}} \cap I^{\text{over}} = \emptyset$ then return NO
    if $W_{\text{must}} \cap I^{\text{under}} \neq \emptyset$ then return YES
    choose $v \in (W_{\text{may}} \setminus W_{\text{must}}) \cap \text{cpre}_1^{\text{over}}(W_{\text{must}})$
    let $v_1 = v \cap \text{cpre}_1(W_{\text{must}}\downarrow)$
    let $v_2 = v \setminus v_1$
    $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
done
```

Figure 2: Three-valued abstraction refinement for reachability games

an initial state, then player 1 has no chance to win[1]. On the other hand, if one of the must-win states is initial, then we know that player 1 can win the game. If neither is the case, then the abstraction is not yet precise enough and must be refined. The refinement, which depends on the game objective, will turn one of the may-win states that is not a must-win state into either a must-win or a never-win state. Afterwards the process is repeated.

4.1 Algorithm for Reachability Games

In a concrete game structure, the set of player 1's winning states for a reachability objective $\Diamond T$ is computed by a fixed point iteration involving the controllable predecessors operator. On the abstract level, we can only approximate the controllable predecessors. Accordingly, under-approximating will yield the must-win states and over-approximating will yield the may-win states, which always include the former. The set of never-win states simply is the complementation of the set of may-win states. The algorithm[2] is given in figure 2. If the abstraction is not sufficiently precise, an abstract state v at the may/must border, that is $v \in (W_{\text{may}} \setminus W_{\text{must}}) \cap \text{cpre}_1(W_{\text{must}}\downarrow)^{\text{over}}$, is chosen and split into two, one of which will contain exactly those concrete states in v that are controllable predecessors (for player 1) of the must-win concretization.

Let us apply the algorithm to the abstraction in figure 1. We have $W_{\text{must}} = \{C, D\}$ and $W_{\text{may}} = \{A, B, C, D\}$, so $W_{\text{may}} \setminus W_{\text{must}} = \{A, B\}$. Then, since $\text{cpre}_1^{\text{over}}(\{C, D\}) = \{B, C, D\}$, B will be chosen and split into $\{2, 4\}$ and the new must-win state $\{3\}$. Because of this, in the next iteration of the loop, $I^{\text{under}} = A$ will be a must-win state, so the abstraction does not need to be refined further and the algorithm will return yes.

4.2 Algorithm for Safety Games

Safety games are somewhat dual to reachability games. Player 1 loses a game with objective $\Box T$ iff player 2 wins the game with objective $\Diamond(S \setminus T)$. That is:

$$\langle 1 \rangle \Box T = S \setminus \langle 2 \rangle \Diamond (S \setminus T)$$

[1]We always assume that player 2 behaves "intelligent".

[2]We abbreviate $\text{cpre}_1(Y\downarrow)^{\text{under}}$ by $\text{cpre}_1^{\text{under}}(Y)$ and $\text{cpre}_1(Y\downarrow)^{\text{over}}$ by $\text{cpre}_1^{\text{over}}(Y)$.

```
while true do
    $W_{\text{must}} := \nu Y.(T^{\text{under}} \cap \text{cpre}_1^{\text{under}}(Y))$
    $W_{\text{may}} := \nu Y.(T^{\text{over}} \cap \text{cpre}_1^{\text{over}}(Y))$
    if $W_{\text{may}} \cap I^{\text{over}} = \emptyset$ then return NO
    if $W_{\text{must}} \cap I^{\text{under}} \neq \emptyset$ then return YES
    choose $v \in (W_{\text{may}} \setminus W_{\text{must}}) \cap \text{cpre}_2^{\text{over}}(V \setminus W_{\text{may}})$
    let $v_1 = v \cap \text{cpre}_2(S \setminus W_{\text{may}}\downarrow)$
    let $v_2 = v \setminus v_1$
    $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
done
```

Figure 3: Three-valued abstraction refinement for safety games

The algorithm (see figure 3) makes use of this fact and therefore is very similar to the previous one. Since we are dealing with a safety objective, the refinement takes place at the border between may- and never-win states. Because of the duality, this is $(W_{\text{may}} \setminus W_{\text{must}}) \cap \text{cpre}_2^{\text{over}}(V \setminus W_{\text{may}})$.

4.3 Algorithm for Parity Games

The algorithm for parity games, given in figure 4, is much more complex than the other two, but the general scheme is still the same. The must-win and may-win states are computed using nested fixed point iterations, where the algorithm assumes the following definition:

- $F_i \overset{\text{def}}{=} \begin{cases} \mu & \text{if } i \text{ is odd} \\ \nu & \text{otherwise} \end{cases}$

- $\text{win}(op, U, k) \overset{\text{def}}{=} F_k Y_k \ldots \nu Y_0 . U \cup (B_k^{\text{under}} \cap op(Y_k)) \cup \cdots \cup (B_0^{\text{under}} \cap op(Y_0))$

Furthermore, the algorithm relies on a recursive procedure for computing the abstract states that may be split.

4.4 Termination

The three given algorithms produce a correct answer—if they terminate. Termination is guaranteed if there exists a finite region algebra for the game structure, i.e., an abstraction that is

- closed under boolean operations, and
- closed under controllable predecessor operators.

For instance, this is the case for timed games.

4.5 Comparison with CEGAR

The presented algorithms never produce a finer abstraction than the counter-example guided approach in order to prove a property. However, the latter may require more refinement that the former. The three-valued algorithms do

```
procedure split($V, U_{\text{may}}, U_{\text{must}}, k$) = begin
    if $k$ odd then
        $P := \{(v, v_1, v_2) \mid v \in \{B_k \cap (U_{\text{may}} \setminus U_{\text{must}}) \cap \text{cpre}_1^{\text{over}}(U_{\text{must}})\},$
                                   $v_1 = v \cap \text{cpre}_1(U_{\text{must}}{\downarrow}), v_2 = v \setminus v_1, v_1 \neq \emptyset, v_2 \neq \emptyset\}$
    else
        $P := \{(v, v_1, v_2) \mid v \in \{B_k \cap (U_{\text{may}} \setminus U_{\text{must}}) \cap \text{cpre}_2^{\text{over}}((V \setminus U_{\text{may}}))\},$
                                   $v_1 = v \cap \text{cpre}_2((V \setminus U_{\text{may}}){\downarrow}), v_2 = v \setminus v_1, v_1 \neq \emptyset, v_2 \neq \emptyset\}$
    if $k = 0$ then return $P$
    let $U'_{\text{may}} = \text{win}(\text{cpre}^{\text{over}}, U_{\text{must}}, k-1)$
    return $P \cup$ split($V, U'_{\text{may}}, U_{\text{must}}, k-1$)
end

while true do
    $W_{\text{must}} := \text{win}(\text{cpre}_1^{\text{under}}, \emptyset, n)$
    $W_{\text{may}} := \text{win}(\text{cpre}_1^{\text{over}}, \emptyset, n)$
    if $W_{\text{may}} \cap I^{\text{over}} = \emptyset$ then return NO
    if $W_{\text{must}} \cap I^{\text{under}} \neq \emptyset$ then return YES
    choose $(v, v_1, v_2) \in$ split($V, W_{\text{may}}, W_{\text{must}}, n$)
    $V := (V \setminus \{v\}) \cup \{v_1, v_2\}$
done
```

Figure 4: Three-valued abstraction refinement for parity games

not explicitly construct the abstract game but heavily use approximations of the controllable predecessors operator. CEGAR, however, works with abstract *must* transitions for player 1 and abstract *may* transitions for player 2. To be precise they should be represented as *hyper-edges* but this is not done to save space, causing a loss of precision.

References

[1] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.

[2] L. de Alfaro and P. Roy. Solving games via three-valued abstraction refinement. In L. Caires and V. T. Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 2007.

[3] T. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *Proc. 30th Int. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 886–902, 2003.