Summary: Counterexample Guided Control

Daniel Dahrendorf

2008-07-17

1 Introduction

The solving of large games may be practically infeasible. Also we cannot apply algorithms which are designed for finite-state games to infinite game structures directly. A key to the success of algorithmic methods for the control of complex systems may be abstraction. This paper [1] presents a fully automatic way of finding suitable abstract models. An abstract model should be coarse enough to reduce the state space but fine enough to exhibit controllability of the concrete model. The algorithm starts with a very coarse abstraction and refines it if it is not controllable and the abstract counterexample does not correspond to a concrete one. This spurious counterexample is used to guide the refinement. The algorithm is presented for safety games but can be extended to ω-regular games

2 Games and Abstraction

Game structure and semantics Let Λ be a set of labels and Φ a set of propositions. A two-player game structure $\mathcal{G} = (V_1, V_2, \delta, P)$ (V_1 and V_2 are disjoint sets and $V = V_1 \cup V_2$) consists of:

- V_1: player 1 nodes
- V_2: player 2 nodes
- δ: labeled transition relation function $\delta \subseteq V \times \Lambda \times V$
- P: function which maps every state to a set of propositions $P : V \to 2^{\Phi}$

We denote the set of available moves in $v \in V$ with $L(v) = \{l \in \Lambda \mid \exists w : (v, l, w) \in \delta\}$. For $i \in \{0, 1\}$ player i chooses in a state $v \in V_i$ a move $l \in L(v)$ and the game proceeds nondeterministically to some state w satisfying $(v, l, w) \in \delta$. For every player 2 state v holds $L(v) \neq \emptyset$.

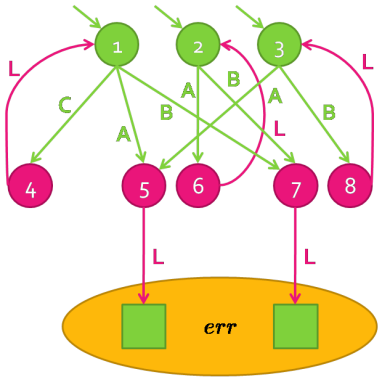

Figure 1: Example of a two-player safety game

Run, strategy and outcome A run is a finite or infinite sequence $v_0v_1v_2...$ of states $v_j \in V$ such that for all $j \geq 0$, if v_j is not the last state, then there is a move $l_j \in \lambda$ with $(v_j, l_j, v_{j+1}) \in \delta$. A strategy of player i is a partial function $f_i : V^* \cdot V_i \rightarrow \lambda$. When possible a player i strategy suggests a move for player i for a given sequence of states. For two strategies f_1 and f_2 the set of possible outcomes $\Omega_{f_1,f_2}(v)$ from a state $v \in V$ is the set of runs defined as follows. A run $v_0v_1v_2...$ is in $\Omega_{f_1,f_2}(v)$ iff $v = v_0$ and for every $j \geq 0$ $v_j \in V_i$ and $(v_j, f_i(v_0...v_j), v_{j+1}) \in \delta$ or v_j is the last state and $L(v_j) = \emptyset$.

Winning conditions We will only consider safety games. A two-player safety game consists of a game structure $\mathcal{G}$ and an objective ϕ. ϕ is a LTL-formula over Φ of the form $\Box\overline{err}$. $[err] \subseteq V$ specify a set of error states. Player 1 will loose if the game will be in a state where err is true or a dead-end state will be encountered. Let Π_1 denote the infinite runs which never visit an error state and are therefore winning for player 1.

- A strategy f_1 is winning for player 1 if for all strategies f_2 of player 2 and all initial states v: $\Omega_{f_1,f_2}(v) \subseteq \Pi_1$
- A strategy f_2 is spoiling for player 2 if for all strategies f_1 of player 1 there is an initial state v such that: $\Omega_{f_1,f_2}(v) \not\subseteq \Pi_1$

Abstraction We want to construct a simplification of the concrete game which is less expensive to solve and sound. This means the abstraction should have a smaller state space and if player 1 wins the abstraction she also wins the concrete game. To ensure soundness we decrease the power of player 1 and increase the power of player 2. Therefore in the abstraction player 1 has fewer moves available and player 2 more moves. An abstraction consists of a game structure $\mathcal{G}^\alpha = (V_1^\alpha, V_2^\alpha, \delta^\alpha, P^\alpha)$ and a concretization function $[\![\cdot]\!] : V^\alpha \rightarrow 2^V$ such that following conditions hold:

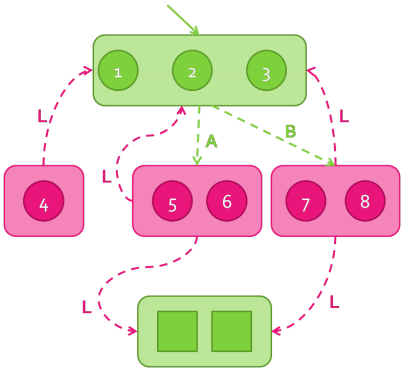

Figure 2: A possible abstraction of the example

1. The player structure is preserved: for $i \in \{1, 2\}$ and all $v^\alpha \in V_i^\alpha$: $[\![v^\alpha]\!] \subseteq V_i$.
2. Propositions are preserved: for all $v^\alpha \in V^\alpha$, if $v, v' \in [\![v^\alpha]\!]$: $P(v) = P(v')$ and $P^\alpha(v^\alpha) = P(v)$
3. The abstract states cover the whole concrete state space: $\bigcup_{v^\alpha \in V^\alpha} [\![v^\alpha]\!] = V$
4. From each abstract player 1 node $v^\alpha \in V_1^\alpha$ only moves are allowed which could be played from each concrete state $v \in [\![v^\alpha]\!]$: $L^\alpha(v^\alpha) = \bigcap_{v \in [\![v^\alpha]\!]} L(v)$
5. From each abstract player 2 node $v^\alpha \in V_2^\alpha$ all moves are allowed which could be played from some concrete state $v \in [\![v^\alpha]\!]$: $L^\alpha(v^\alpha) = \bigcup_{v \in [\![v^\alpha]\!]} L(v)$

An abstraction is uniquely defined by the abstract state space V^α and the concretization function $[\![\cdot]\!]$. We will consider only abstractions with finite state space.

3 Counterexample-Guided Abstraction Refinement

A counterexample is a spoiling strategy for player 2 which shows that player 1 cannot win the game. We divide abstract counterexample in genuine counterexamples which corresponds to one in the concrete game and spurious counterexamples which arises due coarseness of the abstraction. The first step is to determine if an abstract counterexample is genuine and if not we will refine the abstraction.

Abstract counterexample trees As we only consider finite-state safety games we can represent an abstract counterexample as a rooted, directed, labeled, finite tree. Each node n is labeled by an abstract state v^α ($n : v^\alpha$) and possibly a set $r \subseteq V$ of concrete nodes. The edges are labeled with moves. Node n' is an l-child of node n iff $n \xrightarrow{l} n'$ is a edge. An abstract counterexample tree (ACT) T^α is a tree where following conditions hold:

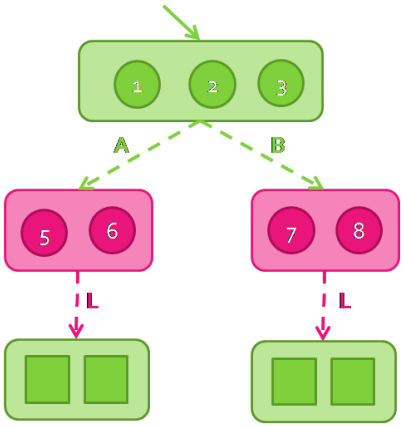

Figure 3: Abstract counterexample tree

1. The root is labeled by $v^\alpha \Rightarrow [\![v^\alpha]\!] \subseteq [init]$
2. $n' : w^\alpha$ is a l-child of $n : v^\alpha \Rightarrow (v^\alpha, l, w^\alpha) \in \delta^\alpha$
3. $n' : v^\alpha$ is a non-leaf player 1 node $\Rightarrow$ for each $l \in L^\alpha(v^\alpha)$ n has at least one l-child
4. $n' : v^\alpha$ is a non-leaf player 2 node $\Rightarrow$ for some $l \in L^\alpha(v^\alpha)$ n has at least one l-child
5. A leaf is labeled by $v^\alpha \Rightarrow L^\alpha(v^\alpha) = \emptyset$ or $[\![v^\alpha]\!] \subseteq [err]$

Concretizing abstract counterexamples To determine whether an abstract counterexample is spurious we will analyze the ACT. A concrete node $v \in [\![v^\alpha]\!]$ can be only part of a spoiling strategy iff a successor of v is part of a spoiling strategy or if it is an error state or a dead-end. We will annotate each node of the ACT with a good set which contains concrete nodes which are part of the spoiling strategy and a bad set which contains the nodes which are not part of the spoiling strategy. Initially for all nodes in a ACT holds $r = [\![v^\alpha]\!]$. Let $C(n) = \{l \in \Lambda \mid n \text{ has an } l\text{-child}\}$ the set of moves that label the outgoing edges of n. $Avl(l) = \{v \in V \mid l \in L(v)\}$ is the set of states where the move l is available and $Epre(X, l) = \{v \in V \mid \exists w : (v, l, w) \in \delta \wedge w \in X\}$ the set of states which do have a move l to a node $w \in X$. $r_{l,j}$ means the good set of the l-children of n. The annotation will be done bottom up by the operator $Focus(n : v^\alpha : r)$.

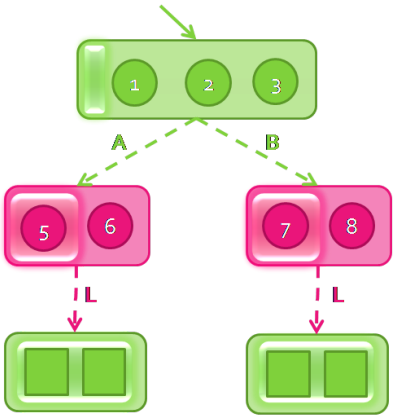

Figure 4: Abstract counterexample tree after applying the focus operator

$$Focus(n : v^\alpha : r) = \begin{cases} r & \text{if n leaf and } L^\alpha(v^\alpha) \neq \emptyset \\ r \cap \left(\bigcap_{l \in C(n)} Epre(\bigcup_j r_{l,j}, l)\right) & \\ \quad \cap \left(\bigcap_{l \notin C(n)} \overline{Avl(l)}\right) & \text{if n other player 1 node} \\ r \cap \left(\bigcup_{l \in C(n)} Epre(\bigcup_j r_{l,j}, l)\right) & \text{if n player 2 node} \end{cases}$$

The operator $Focus(n : v^\alpha : r)$ use the following rules:

- Node n is a leaf and $L^\alpha(v^\alpha) \neq \emptyset$: $r = [\![v^\alpha]\!]$
- Node is a player 1 state: $v \in r$ if for all $l \in L(v)$: $l \in C(n)$ and for every $l \in L(v)$ there is an l-child from where player 2 can spoil
- Node is a player 2 state: $v \in r$ if there is a successor of v from where player 2 can spoil

An abstract counterexample is genuine iff the root of the corresponding ACT has a non-empty good set after applying focus.

Abstraction refinement If we find an ACT to be spurious, we have to refine our abstraction to rule out the spurious abstract counterexample. A refinement will split an abstract state into several states. This is done by the *Shatter* operator which takes a node of the annotated ACT. It takes a node $n : v^\alpha : r$ and returns $R = \{r_1, r_2, ..., r_m\}$ under the following rules:

- Player 1 nodes will be split in the good set, in sets of nodes which have moves which are not edges of n and sets which have a move such that the successor is not in a good set

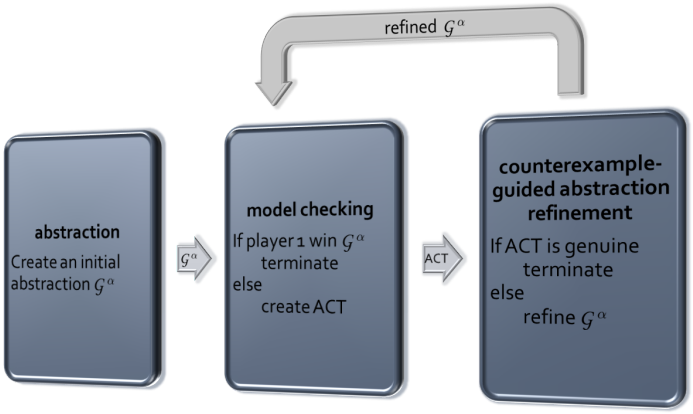

Figure 5: Algorithm of the abstract-verify-refine loop

- Player 2 nodes will be split in the good set and the bad set

Formalized:

$$Shatter(n:q:r) = \begin{cases} \{r\} \cup \{(q \setminus r) \cap Avl(l) \mid l \notin C(n)\} \\ \quad \cup \left\{ (q \setminus r) \cap \overline{Epre\left(\bigcup_j r_{l,j}, l\right)} \mid l \in C(n) \right\} & \text{n player 1 node} \\ \{r, (q \setminus r)\} & \text{n player 2 node} \end{cases}$$

The *Shatter* operator will be applied to each node of the annotated ACT. Let $R \subseteq 2^V$ denote the collection of all sets r which were produced by the *Shatter* operator. $\equiv_R \subseteq V \times V$ is the equivalence relation which is defined by $v_1 \equiv_R v_2$ if for all sets $r \in R$ $v_1 \in r \Leftrightarrow v_2 \in r$. $Closure(R)$ denotes the equivalence classes of $\equiv_R$. The refined Abstraction is uniquely specified by the set $Abstraction(R)$ which contains for each set $r \in Closure(R)$ an abstract state w_r^α with $r = [\![w_r^\alpha]\!]$ and the concretization function $[\![\cdot]\!]$.

4 Counterexample-Guided Controller Synthesis

Algorithm Given a game structure $\mathcal{G}$ and a safety objective $\Box\overline{err}$ we wish to determine if player 1 wins. If she wins we also want to construct a winning strategy for her. Figure 5 shows the algorithm which use the abstraction refinement loop.

Termination The refinement loop may not terminate in general for infinite state games. But termination is guaranteed for finite state games and games for which certain state equivalences has finite index.

5 Conclusion

It was shown that abstraction could reduce the state space so that the abstraction is less expensive to solve. Further infinite state games could be reduced to finite abstractions. Soundness ensures that if player 1 wins the abstract game she will also win the concrete game. The refinement is fully automatic guided by spurious spoiling strategies. It can be extended to ω-regular objectives. The main difference is that spoiling strategies can be not longer represented as finite trees but as finite graphs. For more details see [1].

References

[1] T. A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *ICALP*, pages 886–902, 2003.