Counterexample Guided Control

Thomas A. Henzinger Ranjit Jhala Rupak Majumdar
Department of EECS, UC Berkeley
\{tah,jhala,rupak\}@eecs.berkeley.edu

Abstract. A major hurdle in the algorithmic verification and control of systems is the need to find suitable abstract models, which omit enough details to overcome the state-explosion problem, but retain enough details to exhibit satisfaction or controllability with respect to the specification. The paradigm of counterexample-guided abstraction refinement suggests a fully automatic way of finding suitable abstract models: one starts with a coarse abstraction, attempts to verify or control the abstract model, and if this attempt fails and the abstract counterexample does not correspond to a concrete counterexample, then one uses the spurious counterexample to guide the refinement of the abstract model. We present a scheme for counterexample-guided refinement with the following properties. First, our scheme is the first such method for control. The main difficulty here is that in control, unlike in verification, counterexamples are strategies in a game between system and controller. Second, our scheme can be implemented symbolically and is therefore applicable to infinite-state systems. Third, in the case that the controller has no choices, our scheme subsumes the known algorithms for counterexample-guided verification. In particular, we present a symbolic algorithm that employs counterexample-guided abstraction refinement in a uniform way to check satisfaction as well as controllability for all linear-time specifications (LTL or Büchi automata). Our algorithm is game-based and can be applied in all situations where games provide an adequate model, such as supervisory control, program synthesis, and modular verification.

1 Introduction

The key to the success of algorithmic methods for the verification (analysis) and control (synthesis) of complex systems is abstraction. Useful abstractions have two desirable properties. First, the abstraction should be sound, meaning that if a property (e.g., safety, controllability) is proved for the abstract model of a system, then the property holds also for the concrete system. Second, the abstraction should be effective, meaning that the abstract model is not too fine and can be handled by the tools at hand; for example, in order to use conventional model checkers, the abstraction must be both finite-state and of manageable size. Recent research has focused on a third desirable property of abstractions. A sound and effective abstraction (provided it exists) should be found automatically; otherwise, the labor-intensive process of constructing suitable abstract models often negates the benefits of automatic methods for verification and control. The most successful paradigm in automatic abstraction is the method of counterexample-guided abstraction refinement [4, 8, 5]. According to that paradigm, one starts with a very coarse abstract model, which is effective but may not be informative, meaning that it may not exhibit the desired property even if the concrete system does. Then the abstract model is refined iteratively as follows: first, if the abstract model does not exhibit the desired property, then an abstract counterexample is constructed automatically; second, it can be checked automatically if the abstract counterexample corresponds to a concrete counterexample; if this is not the case, then, third, the abstract model is refined in order to eliminate the spurious counterexample.

The method of counterexample-guided abstraction refinement has been developed for the verification of linear-time properties [8], and some (universal) branching-time properties [9]. It has been applied successfully in both hardware [8] and software verification [5, 16]. We develop the
method of counterexample-guided abstraction refinement for the control of linear-time objectives. In verification, a counterexample to the satisfaction of a linear-time property is a trace that violates the property: for safety properties, a finite trace; for general LTL properties, an infinite, periodic (lasso-shaped) trace. In control, counterexamples are considerably more complicated: a counterexample to the controllability of a system with respect to a linear-time objective is a tree that represents a strategy of the system for violating the property no matter what the controller does. For safety objectives, finite trees are sufficient as counterexamples; for general LTL objectives on finite abstract models, infinite trees are necessary, but they can be finitely represented as graphs with cycles, because finite-state strategies are as powerful as infinite-state strategies [15].

In somewhat more detail, our method proceeds as follows. Given a two-Player game structure (player 1 “controller” vs. player 2 “system”), we wish to check if player 1 has a strategy to achieve a given LTL (or ω-regular) winning condition. Solutions to this problem have applications in supervisory control [19], sequential hardware synthesis and program synthesis [7,6,18], modular verification [2], receptiveness checking [13], interface compatibility checking [11], and schedulability analysis [1]. We automatically construct an abstraction of the given game structure that is as coarse as possible and as fine as necessary in order for player 1 to have a winning strategy. We start with a very coarse abstract game structure and refine it iteratively. First, we check if player 1 has a winning strategy in the abstract game; if so, then the concrete system can be controlled; otherwise, we construct an abstract player 2 strategy that spoils all abstract player 1 strategies. Second, we check if the abstract player 2 strategy corresponds to a spoiling strategy for player 2 in the concrete game; if so, then the concrete system cannot be controlled; otherwise, we refine the abstract game in order to eliminate the abstract player 2 strategy. In this way, we automatically synthesize “maximally abstract” controllers, which distinguish two states of the controlled system only if they need to be distinguished in order to achieve the control objective. In particular, we find finite-state controllers for infinite-state systems, such as hybrid systems, whenever such controllers exist. It should be noted that LTL verification problems are but special cases of LTL control problems, where player 1 (the controller) has no choice of moves. Our method, therefore, includes as a special case counterexample-guided abstraction for linear-time verification.

Furthermore, our method is fully symbolic: while traditional symbolic verification computes fixpoints on the iteration of a transition-precondition operator on regions (symbolic state sets), and traditional symbolic control computes fixpoints on the iteration of a more general, game-precondition operator CPre (Controllable Pre) [3,17], our counterexample-guided abstraction refinement also computes fixpoints on the iteration of two region operators, called Focus and Shatter. The Focus operator is used to check if an abstract counterexample is genuine or spurious. The Shatter operator is used to refine an abstract model guided by a spurious counterexample, splits an abstract state into several states. Our top-level algorithm calls only these three system-specific operators: CPre, Focus, and Shatter. It is therefore applicable not only to finite-state systems but also to infinite-state systems, such as hybrid systems, on which these three operators are computable (termination can be studied as an orthogonal issue along the lines of [12]; clearly, our abstraction-based algorithms terminate in all cases in which the standard, Pre-based algorithms terminate, as e.g., in the control of timed automata [17] and they may terminate in more cases).

2 Games and Abstraction

Two-player Games Let $P$ be a set of propositions. A (two-player) game structure $G = (V_1 \cup V_2, \Sigma, \Gamma, \delta, \mathcal{P})$ consists of two (possibly infinite) disjoint sets $V_1$ and $V_2$ of states (let $V = V_1 \cup V_2$ denote the set of all states), a finite set $\Sigma$ of moves, a function $\Gamma : V \to 2^\Sigma$ mapping states to subsets of moves enabled at the state, a transition relation $\delta \subseteq (V \times \Sigma \times V)$, and a labeling function $\mathcal{P} : V \to 2^\mathcal{P}$ mapping states to sets of propositions. In the sequel $i$ ranges over \{1, 2\}. The set $V_i$
is the set of player i states. The transition relation relates states \( v_i \in V_i \) and moves in \( I(v_i) \) to states in \( V_{\beta_i} \). RJ: do we really need this? Can’t the successor again just be a player i state? RJ. Intuitively, at state \( v \in V_i \), player i chooses a move \( l \in I(v) \), and the game proceeds to some state \( v' \) satisfying \( \delta(v, l, v') \). We require that each state \( v \) has an enabled move \( \Gamma(v) \neq \emptyset \). For a move \( l \in \Sigma \), let \( R_l = \{ v \in V \mid l \in I(v) \} \) denote the set of states in which move \( l \) is enabled. We extend the transition relation to sets via the operators \( \text{Apre} \), \( \text{Epre} : 2^V \times \Sigma \to 2^V \) by defining \( \text{Apre}(X, l) = \{ v \in V \mid \forall v', \delta(v, l, v') \Rightarrow v' \in X \} \) and its dual \( \text{Epre}(X, l) = \{ v \in V \mid \exists v', \delta(v, l, v') \land v' \in X \} \). RJ: shorten to a single definition? RJ. For \( p \in P \) let \( \langle p \rangle = \{ v \mid p \in P(v) \} \) and \( \langle \overline{p} \rangle = V \setminus \langle p \rangle \). We say \( v \) satisfies \( p \) if \( v \in \langle p \rangle \). We assume \( P \) contains a special proposition \( \text{init} \) such that the set \( \langle \text{init} \rangle \) is the set of initial states.

A source-\( v_0 \) run of the game \( G \) is an infinite sequence \( v_0v_1 \ldots \) of states in \( V \) such that for all \( j \geq 0 \), there is \( l_j \in I(v_j) \) such that \( \delta(v_j, l_j, v_{j+1}) \). RJ: changes in definition of strategy, outcomes, please read carefully! A strategy of player i is a function \( f_i : V^* : V_i \to \Sigma \) such that \( f_i(w, v) \in \Gamma(v) \) for every state sequence \( w \in V^* \) and every state \( v \in V_i \). Intuitively, a player i strategy is a function that suggests a move for player i given a sequence of states ending in a player i state.

The transition relation can be nondeterministic\(^1\) i.e., for some \( v \in V, l \in \Sigma \) we could have \( |\{ \delta(v, l, v') \mid v' \in V \}| > 1 \), and we allow the adversary to resolve the nondeterminism at every stage, thus making the games asymmetric. RJ: Do we need to make more explicit HOW this is happening given the def of outcome? RJ

Given strategies \( f_1, f_2 \) of players 1 and 2 \( (f_2, v_2) \). The outcome \( \Omega_{f_1, f_2}(v_0) \) from \( v_0 \in V \) of strategies \( f_1, f_2 \) is a subset of the source-\( v_0 \) runs of \( G \); a run \( v_0v_1v_2 \ldots \) belongs to \( \Omega_{f_1, f_2}(v_0) \) iff for all \( j \geq 0 \), we have \( \delta(v_j, f_1(v_0 \ldots v_j), v_{j+1}) \) where \( v_j \in V_i \).

### Winning Conditions

Due to space restrictions, we will dwell only on safety games and discuss games with linear temporal logic (LTL) or Büchi automaton objectives to the full paper. A safety game has the objective \( \Box \overline{p} \). Intuitively, the goal of player 1 is to keep any play starting from any initial state always inside the set of states satisfying \( p \). Formally, a run \( \pi = v_0v_1 \ldots \) is winning for player 1 if for all \( j \geq 0 \), we have \( v_j \in \langle \overline{p} \rangle \). Let \( \Pi_p \) denote the set of all runs winning for player 1. The strategy \( f_1 \) is winning for player 1 if for all strategies \( f_2 \) of player 2 and for all \( v \in \langle \text{init} \rangle \), the outcome \( \Omega_{f_1, f_2}(v) \subseteq \Pi_p \). Conversely, a strategy \( f_2 \) is spoiling for player 2 if for all strategies \( f_1 \) of player 1, there is a \( v \in \langle \text{init} \rangle \) such that the outcome \( \Omega_{f_1, f_2}(v) \not\subseteq \Pi_p \). In the following, we refer to the game \( (G, \langle \text{init} \rangle, \Box \overline{p}) \) as \( \overline{p} \).

\(^1\) Even if the transition relation is deterministic, we shall see that the abstraction may be nondeterministic.
Example 1 [A Safety Game: EX-SAFETY] Figure 1(a) shows an example of a safety game. The white states are player 1 states and the black ones are player 2 states. The labels on the edges denote moves. The objective is $\square \overline{p}$, i.e., player 1 seeks to avoid the error states $\langle \overline{p} \rangle$. The player 1 states 1, 2, 3 are the initial states, i.e., we wish player 1 to win from all those states. Notice that in fact player 1 wins from the states 1, 2, and 3: at state 1, he plays the move $C$, at state 2, he plays $A$, and at state 3, he plays $B$. In each case, the only move $L$ of player 2 brings the game back to the original state. This ensures the game never reaches a state in $\langle \overline{p} \rangle$.

We now turn our attention to solving games, i.e., determining the states from which player 1 can win. From the predecessor operator $Apre$, we can define the one step controllable predecessor operator $Cpre_1 : 2^V \rightarrow 2^V$, denoting, for a set $X \subseteq V$, the set of states from which player 1 can force the game into $X$ in one step. Player 1 can force the game into $X$ in one step from a state $v_1 \in V_1$ iff in that state he has some enabled move $l$ such that all the $l$-successors of $v_1$ are in $X$, and player 1 can force the game into $X$ from an state $v_2 \in V_2$ iff for all moves $l$ enabled at $v_2$, all $l$-successors of $v_2$ are in $X$. Hence we have:

$$Cpre_1(X) = \bigcup_{l \in \Sigma} (Apre(X, l) \cap V_1) \cup \left( \bigcap_{l \in \Sigma} Apre(X, l) \cap V_2 \right)$$

(1)

Safety games can be solved by iterating the one step controllable predecessor $Cpre_1$ until fixpoint. The set of states from which player 1 can control for $\square \overline{p}$ is exactly the greatest fixpoint $\nu X.\langle \overline{p} \rangle \land Cpre_1(X)$, so player 1 wins the safety game $(G, \langle \text{init} \rangle, \square \overline{p})$ if $\langle \text{init} \rangle \subseteq (\nu X.\langle \overline{p} \rangle \land Cpre_1(X))$.

Abstractions of Games Since solving games on the original structures may be expensive, we wish to find a sound abstraction with a small state space. Soundness means if player 1 wins the LTL game on an abstract game, then he wins the corresponding game on the concrete game. To ensure soundness, we restrict the power of player 1 and increase the power of player 2. Therefore we abstract the player 1 abstract states so that fewer moves are enabled, and the player 2 abstract states so that more moves are enabled.

Definition 1 [Abstract Game Structures] Given a game structure $G = (V_1, V_2, \Sigma, \Gamma, \delta, \mathcal{P})$ with state space $V = V_1 \cup V_2$, an abstract game structure $G^\alpha$ for $G$ is a tuple $(V_1^\alpha, V_2^\alpha, \Sigma, \Gamma^\alpha, \delta^\alpha, \mathcal{P}^\alpha)$ and a concretization function $[\cdot] : V^\alpha \rightarrow 2^V$ (where $V^\alpha = V_1^\alpha \cup V_2^\alpha$ is the abstract state space) such that: (1) The abstraction preserves the player structure and proposition labels: for $i \in \{1, 2\}$ we have $\forall v^\alpha \in V_1^\alpha, [v^\alpha] \subseteq V_i$; and for each $v^\alpha \in V^\alpha$, if $v, v' \in [v^\alpha]$ then $\mathcal{P}(v) = \mathcal{P}(v')$. Moreover, the abstract labeling function $\mathcal{P}^\alpha : V^\alpha \rightarrow 2^\Sigma$ maps an abstract state $v^\alpha$ to $\mathcal{P}(v)$ where $v \in [v^\alpha]$.

(2) The abstract states “cover” the concrete state space: $\bigcup_{v^\alpha \in V^\alpha} [v^\alpha] = V$, (3) For a player 1 abstract state $v_1^\alpha \in V_1^\alpha$, the set of enabled moves $\Gamma^\alpha(v_1^\alpha) = \bigcap_{v \in [v_1]} \Gamma(v)$. For a player 2 abstract state $v_2^\alpha \in V_2^\alpha$, $\Gamma^\alpha(v_2^\alpha) = \bigcup_{v \in [v_2]} \Gamma(v)$. (4) The abstract transition relation is $\delta^\alpha \subseteq V^\alpha \times \Sigma \times V^\alpha$ such that $\delta^\alpha(v^\alpha, l, w^\alpha)$ iff $l \in \Gamma^\alpha(v^\alpha)$ and $\exists v \in [v^\alpha], \forall v' \in [v^\alpha], \delta(v, l, v')^2$.

Notice that a game $G$, an abstract state space $V^\alpha$, and the concretization $[\cdot]$ uniquely determine (i.e., generate) the abstraction of the game $G^\alpha$. Intuitively, each abstract state represents a set of concrete states of the original game $G$; we sometimes identify (with abuse of notation) the concretization of an abstract state with the abstract state. The abstract transition relation can be extended to sets of abstract states via the operators $Apre^\alpha, Epre^\alpha$ as for concrete games.

Proposition 1 [Soundness] Let $G^\alpha$ be an abstract game structure for a concrete game structure $G$. For any LTL objective $\Psi$, if player 1 wins the LTL game $(G^\alpha, \{v^\alpha\}, \Psi)$ from a state $v^\alpha$ on the abstract game structure, then he wins the LTL game $(G, [v^\alpha], \Psi)$ on the concrete game $G$.  

2 So abstract game can be nondeterministic even if the concrete game is not
Example 2 [An Abstraction for EX-SAFETY] Figure 1(b) shows one particular abstraction for EX-SAFETY. The boxes denote abstract states with the states they represent drawn inside them. The dashed arrows are the abstract transitions. Notice that from the starting player 1 box, the move C is not enabled as not all the states in the box can do it. ■

3 Counterexample-Guided Refinement

In this section we define counterexamples and give an algorithm to decide whether an abstract counterexample corresponds to a player 2 spoiling strategy for the concrete safety game (i.e., is “genuine”), in which case no controller can be synthesized, or if it arises due to the coarseness of the abstraction (i.e., is “spurious”), in which case we must refine the abstraction to rule it out. In the sequel, we show the different steps on a fixed safety game (G, init, p).

Abstract Counterexamples A counterexample for a safety game \( \square p \) is a strategy of player 2 that ensures that for every strategy of player 1, some state of \( \langle p \rangle \) is reached eventually. Since memoryless strategies suffice for player 2, finite trees form a natural representation of such counterexamples. We work with rooted, directed, finite trees with labels on both nodes and edges. Each edge is labeled with a move from \( \Sigma \). If \( n \rightarrow n' \) is an edge in the tree we say \( n' \) is an l-child of \( n \). Nodes are marked with either an abstract state (denoted by \( v^a \)), a concrete state (denoted by \( v \)), or a set of concrete states (denoted by \( r \)).

We write \( n : \# \) for node \( n \) marked with \( \# \). A leaf node has no children.

Definition 2 [Abstract counterexample trees] An abstract counterexample tree \( T^a \) for \( G^a \) is a finite tree where each node is labeled by an abstract state such that \( n_1 : v^a_1 \) is an l-child of \( n : v^a \) only if \( \delta^a(v^a, l, v^a_1) \) and (1) The root is labeled by an abstract initial state, (2) If node \( n : v^a \) is a leaf, then either \( v^a \in V^a_1 \) and there is no enabled move \( (\Gamma^a(v^a) = \emptyset) \), or \( [v^a] \subseteq \langle p \rangle \). (3) If node \( n : v^a \) is an internal player 1 node, then for each \( l \in \Gamma^a(v^a) \), \( n \) has at least one l-child. (4) If node \( n : v^a \) is an internal player 2 node, then for some \( l \in \Gamma^a(v^a) \), \( n \) has at least one l-child. ■

We define a partial order on counterexample trees as follows: \( T_1 \subseteq T_2 \) iff the rooted tree \( T_1 \) is equal to some subgraph of \( T_2 \) (with the same root). An abstract counterexample tree \( T^a \) is maximal (respectively, minimal) for \( G^a \) if there is no counterexample tree \( T^a_1 \) for \( G^a \) such that \( T^a \subseteq T^a_1 \) (respectively, \( T^a \supseteq T^a_1 \)). The type of a counterexample tree \( T^a \) is a tree \( type(T^a) \) that is identical to \( T^a \), except that only the edges are labeled, not the nodes. The set of types of a counterexample tree \( T^a \) is denoted \( type(T^a) = \{type(T^a_1) \mid T^a_1 \text{ is a counterexample tree and } T^a_1 \subseteq T^a \} \).

An abstract counterexample tree \( T^a \) corresponds to a set of spoiling strategies for player 2 in the abstract game. At each player 1 state \( n : v^a \), for each enabled move \( l \) that player 1 can choose, there is (i.e., player 2 can choose) at least one successor abstract state from which player 2 wins subsequently, namely any one of the l-children of \( n : v^a \). Depending on how the nondeterminism is resolved we get a set of spoiling strategies for player 2.

Example 3 [An Abstract Counterexample for EX-SAFETY] Figures 1(c), 1(d) respectively show an abstract counterexample \( T^a \) for the abstraction of EX-SAFETY and the type of the counterexample. After player 1 plays either move \( A \) or move \( B \), player 2 can play move \( L \) and take the game to the error set \( \langle p \rangle \). ■

---

3 Unlike counterexamples for safety properties \([8,5]\), which are just traces.

4 To ensure soundness, player 1 abstract states with no successors need to be considered losing states.
Concretizing Abstract Counterexamples  An abstract counterexample may not be realizable in the concrete game, i.e., even though player 2 has a strategy to spoil the abstract game, it does not correspond to any spoiling strategy for player 2 in the concrete game.

A concrete counterexample tree $T$ is a finite tree where each node is labeled by a concrete state such that: (1) the root is labeled with an initial state, (2) if $(n : v_n) \xrightarrow{m} (n' : v_{n'}) \in T$ then $\delta(v_n, l, v_{n'})$, and (3) for each player 1 internal node $n : v_n$ in $T$, if the children of $n$ are labeled by moves $C(n)$ then $\Gamma(v_n) = C(n)$, i.e., exactly those moves are enabled at $v_n$, and (4) for each leaf node $n : v_n$ in $T$, $v_n \in \langle p \rangle$. An abstract counterexample $T^\alpha$ is realized by the concrete counterexample $T$ if $\text{type}(T) = \text{type}(T^\alpha)$ and each node $n$ marked with $\nu^n$ in $T^\alpha$ is marked in $T$ with a single state $v \in [\nu^n]$. An abstract counterexample tree $T^\alpha$ is genuine if there is some $T^\alpha \subseteq T^\alpha$ such that $T^\alpha$ is realized by some concrete counterexample; it is spurious otherwise. We need a procedure to determine if a given abstract counterexample is genuine.

The basic step in analysis of a counterexample tree $T^\alpha$ is the Focus operation $\text{Focus}_{T^\alpha} : (T^\alpha \times 2^V) \rightarrow 2^V$, that takes a node $n$ in $T^\alpha$ and a set of concrete states, and returns a subset of the concrete states. Let $C(n)$ be the labels on edges leaving $n \in T^\alpha$, and let $n_i = r_{l,i}$ be the various $l$-children of $n$ (indexed by $i$), where each of the $r_{l,i}$ denotes a set of concrete states. Recall that $R_l$ is the set of states where the move $l$ is enabled. Define the operation $\text{Focus}_{T^\alpha}$ as: \begin{equation} \text{Focus}_{T^\alpha}(n, r) = \begin{cases} r & \text{if } n \text{ is a leaf node} \\ r \cap \left( \bigcap_{i \in C(n)} \text{Epre}(\bigcup_{i \in C(n)} \bigcup_{l \in R_l}) \right) \cap \left( \bigbigcap_{i \in C(n)} \bigcup_{l \in R_l} \bigcup_{i \in C(n)} \bigcup_{l \in R_l} \right) & \text{if } n \text{ is a plr 1 node}, \\ r \cap \left( \bigcap_{i \in C(n)} \text{Epre}(\bigcup_{i \in C(n)} \bigcup_{l \in R_l}) \right) & \text{if } n \text{ is a plr 2 node}. \end{cases} \end{equation}

Intuitively, the Focus operator does the following. At each point, each node of the tree will be marked by those (concrete) states that we know are an upper approximation of the the set of states that can actually be a part of a concrete counterexample of the type of the given one. One step of the Focus operator sharpens this set by finding exactly which of the states in the present overapproximation can actually have successors which can lead to counterexamples. Thus the fixpoint of this operator contains exactly those states that can be part of the counterexample. For any leaf node (since we assume every leaf node has at least one enabled move), all states are in $\langle p \rangle$, and therefore can actually be in a counterexample. For player 1 internal nodes, the only states that can be part of a counterexample are those where (i) the only enabled moves are the moves labeling the child-edges on the tree and moreover (ii) where for each enabled move $l$, there exists some $l$-successor from which player 2 has a spoiling strategy corresponding to the given tree, i.e., those states which for every $l$, lie in the Epre of the node’s $l$-children’s approximations. Similarly for player 2 nodes, the only states that may participate in the counterexample are those states which have some successor from which player 2 can subsequently spoil, i.e., exactly the union of the Epre of the overapproximations of the child nodes.

The counterexample analysis procedure $\text{AnalyzeCounterex}$ iterates Focus on nodes until there is no change, i.e., it computes the greatest fixpoint of the Focus operator on the abstract tree. This is exactly the set of states that can be a part of a concrete counterexample. In other words, we model check $(T^\alpha)$ to find if it is a genuine counterexample. Let $\text{Focus}^*(n)$ denote the fixpoint value for the node $n : \nu^n$ of the abstract counterexample tree. Thus $\text{Focus}^*(\text{root})$ is empty iff $T^\alpha$ is spurious. The nondeterminism of the while loop of $\text{AnalyzeCounterex}$ can be efficiently resolved by focusing each node after focusing all of its children. We omit the details for brevity.

**Proposition 2 [Correctness of $\text{AnalyzeCounterex}$]** The counterexample $T^\alpha$ is spurious iff the procedure $\text{AnalyzeCounterex}(T^\alpha)$ returns SPURIOUS. Checking if a counterexample tree is spurious can be done in time linear in the size of the counterexample tree.
Algorithm 1 AnalyzeCounterex($T^a$)

INPUT: An abstract counterexample tree $T^a$, with root $\text{root}$.
OUTPUT: Whether the abstract counterexample $T^a$ is SPURIOUS or GENUINE

for each $n : v^a \in T^a$ do
  relabel $n$ with $[v^a]$
  while there is some node $n : r$ with $r \notin \text{Focus}_{T^a}(n, r)$ do
    pick some node $n : r$ to focus
    replace $n : r$ with $n : \text{Focus}_{T^a}(n, r)$ in the tree
  if $\text{root} : r$ with $r = \emptyset$ then
    return SPURIOUS
  return GENUINE

![Diagram](image)

Fig. 2. Focusing the nodes of $T^a$.

Fig. 3. Refinement for EX-SAFETY

Example 4 [Counterexample analysis for EX-SAFETY] Figure 2 shows the result of running AnalyzeCounterex (or Focus*) on the counterexample $T^a$. The lightly shaded parts of the boxes denote the states which may still be a part of a counterexample, i.e., the states that we take the Epres of when computing the Focus. The dashed arrows indicate abstract transitions and the solid arrows the concrete transitions occurring between the respective concrete states. Figure 2(a) shows the player 2 nodes getting focused. Note that all the states in the leaf node are error states (satisfying $p$) hence they are all in the lightly shaded box. Only 6 (and respectively 8) can go to the error region from the two abstract states hence only they are in the focused region. Figure 2(b) shows the effect of doing a subsequent Focus on the root node. None of the states in the root node are such that they can play only the moves $A, B$ and subsequently go to states from which player 2 can subsequently spoil. Hence neither of those states can serve as the root of a concrete counterexample tree of the same type as the abstract counterexample. Thus the shaded boxes in Figure 2(b) also show the fixpoint of Focus. Since Focus* is empty we can conclude the counterexample is spurious. □

Counterexample-Guided Refinement An abstraction that is coarse enough to exhibit a spurious counterexample must be refined, as player 1 has no winning strategy in it. Refinement is done by splitting the abstract states labeling the nodes of the counterexample tree so as to rule out all possible counterexamples whose type is in Types($T^a$). The splitting uses a Shatter operation that is dual to the Focus used in for counterexample analysis. The Shatter operator takes a node $n : v^a$ of the abstract counterexample tree and returns a set of subsets of $[v^a]$, which replace $v^a$ in the refined abstraction. One of those subsets is the “good” part Focus*(n) from which player 2 does indeed have a spoiling strategy given by the current counterexample tree. The other subsets are the “bad” subsets of $[v^a] \setminus \text{Focus}^*(n)$ from which the present spoiling strategy fails. Each “bad”
subset is small enough that it is clear why the present strategy fails when that subset is represented by a single abstract state.

For a player 1 node, a part is bad because every concrete state in it either (1) has some other move enabled that is not in the abstract counterexample that the present player 2 spoiling strategy does not account for, or (2) has some move \( l \) such that none of its \( l \)-successors is in a block from which player 2 can subsequently spoil. For a player 2 node, the bad part is the set of states from which there is no \( l \)-successor in a "good" block.

For the node \( n : v^n \) in \( T^n \) denote by \( r^+ \) the set Focus\(^*(n)\) and by \( r^- \) the set \( \{v^n\} \setminus r^+ \). Recall that \( C(n) \) is the set of labels on edges leaving \( n \in T^n \), and \( n_{i,i} : r_{i,i} \) (for index \( i \)) are the various \( l \)-children of \( n \), where \( r_{i,i} = \{v_{i,i}^n\} \). Let \( r_{i,i}^+ = \text{Focus}^*(n_{i,i}) \) and \( r_{i,i}^- = r_{i,i} \setminus r_{i,i}^+ \). The Shatter operator is then defined as \( \bullet \) RJ: changed! Please read carefully \( \bullet \):

\[
\text{Shatter}_{T^n}(n : v^n) = \begin{cases} 
\{r^+ \cup \{r^- \cap R_l \mid l \notin C(n)\} \cup \{r^- \cap \text{Epre}(\cup r_{i,i}^+ \cup l) \mid l \in C(n)\} & \text{if } n \text{ is Plr 1} \\
\{r^+, r^-\} & \text{if } n \text{ is Plr 2}
\end{cases}
\]

(3)

Example 5 [Refinement using Shatter for Ex-Safety] Figure 3 shows the effect of the Shatter operator on the root node, and the final refined game in which the counterexample is removed. For the other nodes the shatter is trivial, namely into \( r^+ \), \( r^- \). We break up the states of the root node into (i) those that can execute some different move, which we then break into groups such that each state in the group can play some new move here only the state 1 which can play the new move \( C \) and (ii) those that can execute only \( A \) or \( B \) but can escape to a state from which player 2’s strategy fails (i.e., a state not in the gfp of Focus). This latter group is further split into those that can play \( A \) and escape namely the state 2 and those that can play \( B \) and escape, namely the state 3. Notice that any abstraction in which any two of 1, 2, 3 are together admits a counterexample of this type. \( \blacksquare \)

For any abstract counterexample tree \( T^\alpha \), and \( n : v^n \) in \( T^\alpha \), it can be shown that if \( R = \text{Shatter}_{T^n}(n : v^n) \) then we have \( [v^n] = \bigcup_{r \in R} r^+ \). The refinement step then is: given the old abstraction structure \( G^\alpha \), we replace each "shattered" state \( v^n \in V^\alpha \) with the smaller states in \( \text{Shatter}_{T^n}(n : [v^n]) \), to get a finer abstraction structure \( G^\alpha \), and recompute \( \delta^\alpha, T^\alpha, \text{Apre}^\alpha \) and \( \text{Cpre}^\alpha \) for the refined state space.\(^6\) Formally, for any abstraction \( G^\alpha \), and a spurious abstract counterexample \( T^\alpha \) for \( G^\alpha \), define the refined game \( \text{Refine}(G, G^\alpha, T^\alpha) \) as the abstract game generated by \( G \) and the abstract state space \( V^\alpha \cup V_{T^\alpha} \setminus V_{T^n} \), where \( V_{T^n} = \{v^n \mid (n : v^n) \in T^n\} \) and \( V_{T^\alpha} = \bigcup_{(n : v^n) \in T^\alpha} \text{Shatter}(n : v^n) \).

Proposition 3 For a game \( G \) and every spurious counterexample \( T^\alpha \) on \( G^\alpha \), the refined abstraction \( \text{Refine}(G, G^\alpha, T^\alpha) \) (and any subsequent refinement thereof) has no counterexample whose type is in \( \text{Types}(T^\alpha) \).

4 Counterexample-Guided Controller Synthesis

**Safety Control** Our algorithm for safety control generalizes the “abstract-check-refine” loop described by [4,8]. Given a game structure \( G \), a set of initial states \( r_0 = \{\text{init}\} \), and a proposition \( p \) with \( W = \langle p \rangle \), we wish to solve the safety game \( (G, r_0, \square \neg p) \), i.e., we wish to find if player 1 wins from all initial states. Informally, the algorithm is as follows.

\(^6\) Each subset in \( \text{Shatter}_{T^n}(n : v^n) \) corresponds to a new abstract state

\(^7\) Recall that the abstract game is generated by its state space and the concrete game.
Algorithm 2 AnalyzeRefineCounterex($G, T^a$)

INPUT: An abstract counterexample tree $T^a$, with root root.
OUTPUT: If the counterex. is spurious then a refined game not containing the counterex. else report that the counterex. is genuine.
if AnalyzeCounterex($T^a$) = Spurious then
  $G^{a'} :=$ Refine($G, G^a, T^a$); return (Spurious, $G^{a'}$)
else return Genuine endif

Algorithm 3 CGSafetyControl($G, r_0, \Psi$)

INPUT: A game structure $G$, an initial set of states $r_0$, a safety objective $\Psi = \bigotimes p$.
OUTPUT: A synthesized controller exhibited by control strategy $T^a$ or no controller possible exhibited by adversary strategy $T^a$.
$G^a :=$ InitialAbstraction($G, r_0, \Psi$)
repeat
  (winner, $T^a$) := ModelCheck($G^a, r_0, \Psi$)
  if winner = 2 then
    if AnalyzeRefineCounterex($G, T^a$) = (Spurious, $G^{a'}$) then
      $G^a := G^{a'}$
      winner := 1
    until winner $\neq 1$
  if winner = 1 then
    return Synthesized Controller, $T^a$
  else
    return No Controller Possible, $T^a$
end repeat

Step 1 (“abstraction”) We first construct an initial abstract game. One such abstraction could be the trivial abstraction where all player 1 states are partitioned according to the labeling of propositions, i.e., into those satisfying both $p$ and init, those satisfying only $p$, those satisfying only init, and those satisfying neither. Recall that the abstract transition relation is generated by the abstract state space and the concrete game.

Step 2 (“model checking”) Next, we model check the abstract game to find if player 1 can keep the game inside the desired region $V^a \setminus \{p\}^a$ is in the abstract game, starting at all of the abstract states $r_0$. If player 1 can win the abstract safety game from the states $r_0^a$, then the model checker gives us a winning strategy on the abstract game, from which a winning strategy in the concrete game can be easily constructed [12]. If not, i.e., if player 2 has a strategy to reach $W^a$ no matter what player 1 does, then the model checker produces an abstract counterexample symbolically [10]. Note that as the abstract state space is finite, the model checking is guaranteed to terminate.

Step 3 (“counterexample-driven refinement”) If model checking returns an abstract counterexample, we analyze this counterexample strategy to see if it is genuine. If it is genuine, no controller can be synthesized. If instead it is spurious, then we refine the abstraction so that this counterexample (and similar ones) do not arise on subsequent model checking runs.

Goto Step 2, ("loop") We then the process with the refined abstraction, until either we get a player 1 winning strategy for player 1 (and hence a controller), or we get a genuine counterexample (i.e., no controller is possible).

We summarize the counterexample driven control procedure in Algorithm 3. The algorithm ModelCheck returns a pair $(1, \text{strategy})$ if player 1 can win the game where strategy is his winning strategy, or

---

*For any $X \subseteq V$, $X^a$ denotes the set $\{v^a \mid [v^a] \cap X \neq \emptyset\}$*
it returns \((2, T^\alpha)\) when player 2 has a spoiling strategy, where \(T^\alpha\) is an abstract counterexample for the safety game for \(G^\alpha\). The function \texttt{InitialAbstraction} just returns the trivial abstraction for the game that respects the propositional labeling. From the soundness of abstraction, we get the soundness of the algorithm.

**Theorem 1.** For any initial region \(r_0\), safety objective \(\Box \phi\), and for any terminating execution of Algorithm \texttt{CGSafetyControl}(\(G, r_0, \Box \phi\)), we have:

(i) If \texttt{CGSafetyControl}(\(G, r_0, \Box \phi\)) returns an error tree, then there is a state \(v_0 \in \lbrack r_0 \rbrack\) such that player 2 has a spoiling strategy for the safety game \((G, \{v_0\}, \Box \phi)\).

(ii) Otherwise, \texttt{CGSafetyControl}(\(G, r_0, \Box \phi\)) returns a set of states \(r\) that satisfies both \(r_0 \subseteq r\) and player 1 wins the safety game \((G, r, \Box \phi)\).

In general, Algorithm \texttt{CGSafetyControl} will not terminate for infinite-state games (it does terminate for finite-state games). However, one can prove sufficient conditions for termination provided certain state equivalences on the game structure have finite index [12]. Notice, for example, that in the course of Algorithm \texttt{CGSafetyControl}, the abstract state space always consists of blocks of the alternating bisimilarity relation [12]. Hence, termination is guaranteed for games with an alternating bisimilarity relation of finite index. The next example shows how the algorithm works for verification.

**Example 6 [A Safety Verification Problem]** The standard safety verification (or invariant checking) problem on transition systems is a special case of the safety game where there are only player 2 states. An example is given in Figure 4(a). The starting states are 1, 2 and we wish to check that \(\Box \phi\), i.e., that the states 5, 6 are never visited. It is easy to see that the system satisfies this criterion. In Figure 4(b), we show an abstraction for Ex-VERIF. This is exactly the standard existential abstraction for transition systems. Figure 4(c) shows a trace\(^9\) \(\tau^\alpha\) which is an abstract counterexample for the abstraction of Ex-VERIF. In the Figure 5 we see the result of running \texttt{AnalyzeCounterex} (or Focus\(^*\)) on \(\tau^\alpha\). In Figure 5(a) we see the effect of doing a Focus on the second abstract state in \(\tau^\alpha\). All the concrete states corresponding to the last abstract state are error states (i.e., satisfy \(\phi\)) hence they are all shaded. Only state 4 can go to one of the error states (i.e., lies in the Pre of the error states) hence it is the only state in the focused region of that node. In Figure 5(b) we see the second application of Focus, this time to the root of the trace — none of the states 1, 2 in the root go to 4 (which is the only candidate state for the second abstract state) hence the focused region is empty, and the counterexample spurious. The fixpoint of Focus is shown by the shaded portions in Figure 5(b). Figure 6(a) is shows the effect of Shatter on the nodes of \(\tau^\alpha\). Since the fixpoint of Focus for second state is \(\{4\}\), the state gets shattered into \(\{3\}, \{4\}\), which are respectively its \(r^-, r^+\). No other state is shattered. Figure 6(b) shows the refined transition system which free of counterexamples.

**LTL Control** We now generalize the counterexample guided safety control procedure to LTL games. First, the procedure for solving games must implement a symbolic model checker for LTL games: given a game with an LTL winning condition, we can construct a \(\mu\)-calculus formula with the \(\text{Cpre}_i\) operator that characterizes the set of states from which player 1 wins the game [12]. Moreover, from the fixpoint computation, one can symbolically construct a winning strategy for player 1 [17,12] or a counterexample strategy for player 2. Counterexamples for LTL control (i.e., strategies for player 2) are directed graphs (rather than trees). So we must generalize abstract counterexample trees to abstract counterexample graphs, which are rooted, directed graphs, with

\(^9\) The counterexample for invariant verification is just a trace.
Algorithm 4 AnalyzeRefineLTLCounterex($G, G^a$)

INPUT: An abstract counterexample graph $G^a$, with root root.

OUTPUT: If the counterex. is spurious then a refined game not containing the counterex. else reports that the counterex is genuine.

define $F(n) = v^a$ if $n : v^a$ in $G^a$

for each $n : v^a \in G^a$ do
    Re-label $n$ with $[v^a]$
    $W^a := v^a$
    \{ $W^a$ will be the abstract state space of the refinement \}
    while there is some node $n : r \in Focus_{G^a}(n, r)$ do
        pick some node $n : r$ to focus
        replace $n : r$ with $n : Focus_{G^a}(n, r)$ in the graph
        $W^a := W^a \setminus \{r\} \cup \text{Shatter}(n : r)$ \footnote{Note that now AnalyzeRefineLTLCounterex is not guaranteed to terminate. However, it does terminate for finite state games.}
        if root : $r$ with $r = \emptyset$ then
            return (SPURIOUS, Game generated by $(G, W^a)$)
    return GENUINE

nodes labeled by abstract states, satisfying conditions (i), (iii), (iv) of Definition 2. The operators $\text{type}$ and $\text{Types}$ and the partial order $\subseteq$ are trivially extended to abstract counterexample graphs.

Second, for analyzing counterexamples, we have to generalize the Focus and Shatter operators by considering successors of a node in the graph. This is done in algorithm AnalyzeRefineLTLCounterex which computes the fixpoint of the Focus operator on the counterexample graph. However, for general graphs we cannot apply a bottom-up marking strategy: in the presence of cycles, the fixpoint computation may require focusing a node several times before the fixpoint is reached\footnote{Note that now AnalyzeRefineLTLCounterex is not guaranteed to terminate. However, it does terminate for finite state games.}, and moreover, each focus must be accompanied by a shatter. We omit details due to lack of space. The Shatter procedure of Section 3 can be seen as a special case, where each node is focused (and shattered) exactly once.

From the procedure CGSafetyControl we obtain an algorithm CGLTLControl for counterexample-guided LTL control, by simply replacing the procedure AnalyzeRefineCounterex with the procedure AnalyzeRefineLTLCounterex and replacing the safety objective with any arbitrary LTL objective $\Psi$. 

Theorem 2. For any game $G$, initial region $r_0$, LTL objective $\Psi$, and for any terminating execution of Algorithm CGLTLControl($G, r_0, \Psi$), we have:

(i) If CGLTLControl($G, r_0, \Psi$) returns an error, then there is a state $v_0 \in [r_0]$ such that player 2 has a spoiling strategy in the game ($G, \{v_0\}, \Psi$).
(ii) Otherwise, \( \text{CGLTLControl}(G, r_0, \Psi) \) returns a region \( r \) that satisfies both \( r_0 \subseteq r \) and player 1 wins the game \( (G, r, \Psi) \).

**Example 7 [An LTL Game]** In Figure 7(a) we show an example of an LTL game. We wish to check whether player 1 can force the game into a \( p \)-state infinitely often, i.e., \( \Box \Diamond p \). Figure 7(b) shows an abstraction for such a game, and Figure 7(c) shows the result of solving the game on that abstraction: an abstract counterexample where player 2 forces a loop not containing a \( p \) state. We show how the counterexample is analyzed and discovered to be spurious in Figure 8. In the first figure we see the effect of doing a Focus on the second (lower) node of \( G^a \). As only the state \( \{4\} \) has a move that lands it in \( \{1, 2\} \) (the region of the \( B \)-child of the node) the focused region for the node is \( \{4\} \). Also, the abstract state is shattered into two states \( \{4\} \) and \( \{3\} \) (respectively \( r^+, r^- \)). Next, in Figure 8(b) we see the effect of doing a Focus on the upper node of \( G^a \). Only the state 2 has an \( A \) successor in the focused region of the lower node, hence the focused region becomes \( \{2\} \), and the upper node gets shattered into \( \{1\} \) and \( \{2\} \). In Figure 8(c) we again do a Focus on the lower node. Since no state (in particular 4) has a \( B \)-move to the focused region of the upper node, the focused region of the lower node becomes empty. In the next Figure 8(d) we see that when we do a last Focus on the upper node, it becomes empty as well. In Figure 7(d) we see the refined abstraction for the game; it is easy to see that player 2 has no spoiling strategy.

In [9], the authors consider counterexample based model checking for ACTL formulas. In that case (and in fact, for some more expressive logics considered in [9]), the counterexample graphs are tree-like, and our algorithm for analyzing counterexamples and refining the abstraction specializes to theirs; in fact since the counterexamples are models of ECTL formulas, the counterexample graph in that case contains only player 2 nodes. Similarly, in model checking the \( \mu \)-calculus, one can reduce the model checking question to solving a parity game [14]. Hence, the above method provides a counterexample driven model checking procedure for the \( \mu \)-calculus.

Finally, since the operators Focus and Shatter can be defined in terms of the symbolic Epre operators, our algorithms can be implemented symbolically.

**References**


