Abstraction Refinement for Games with Incomplete Information

by

Rayna Dimitrova    Bernd Finkbeiner
Abstraction Refinement for Games with Incomplete Information

Rayna Dimitrova and Bernd Finkbeiner

Universität des Saarlandes

Abstract. Counterexample-guided abstraction refinement (CEGAR) is used in automated software analysis to find suitable finite-state abstractions of infinite-state systems. In this paper, we extend CEGAR to games with incomplete information, as they commonly occur in controller synthesis and modular verification. The challenge is that, under incomplete information, one must carefully account for the knowledge available to the player: the strategy must not depend on information the player cannot see. We propose an abstraction mechanism for games under incomplete information that incorporates the approximation of the players’ moves into a knowledge-based subset construction on the abstract state space. This abstraction results in a perfect-information game over a finite graph. The concretizability of abstract strategies can be encoded as the satisfiability of strategy-tree formulas. Based on this encoding, we present an interpolation-based approach for selecting new predicates and provide sufficient conditions for the termination of the resulting refinement loop.

1 Introduction

Infinite games are a natural model of reactive systems as they capture the ongoing interaction between a system and its environment. Many problems in automated software analysis, including controller synthesis [8] and modular verification [11], can be reduced to finding (or deciding the existence of) a winning strategy in an infinite game. The design of algorithms for solving such games is complicated by the following two challenges: First, games derived from software systems usually have an infinite (or finite, but very large) state space. Second, the games are usually played under incomplete information: it is unrealistic to assume that a system has full access to the global state, e.g., that a process can observe the private variables of the other processes.

The most successful approach to treat infinite state spaces in software verification is predicate abstraction with counterexample-guided abstraction refinement (CEGAR) [3, 1]. For games with complete information [6, 4], CEGAR builds abstractions that overapproximate the environment’s moves and underapproximate the system’s moves. If the system wins the abstract game it is guaranteed to also win the concrete game. If the environment wins the abstract game, one checks if the strategy is spurious in the sense that it contains an abstract state from which the strategy cannot be concretized. If such a state exists, the state is split to ensure that the strategy is eliminated from further consideration.

For games with incomplete information, the situation is more complicated, because the strategic capabilities of a player depend not only on the available moves, but also on the knowledge about the state of the game. If the abstract game provides less information to the system than the concrete game, then the environment may spuriously win the abstract game, because the abstract system may be unable to distinguish a certain pair of states and may therefore be forced to apply the same move in the two states where the concrete system can select different moves.

An abstraction refinement approach for games with incomplete information must therefore carefully account for the information collected by the system. A first requirement is that the refinement should avoid predicates that mix variables that are observable to the system with those that are hidden. Such mixed predicates lead to the situation that the concrete system has partial information (the values of the observable variables), while the abstract system does not know the value of the predicate at all. Since the system may collect information over multiple steps of a play, however, just separating the variables alone is not enough. Consider, for example, a situation where, in order to win, the system has to react with output \( x_o \approx 0 \) if some hidden
variable $x_h$ has value $x_h \approx 0$ and with output $x_o \approx 1$ if $x_h \approx 1$. Now, suppose the system is able to deduce the value of $x_h$ from the prefix that leads to the state, because an observable rational-valued input variable $x_i$ is either always positive or always negative if $x_h \approx 0$ and flips its sign otherwise. To rule out the spuriously winning strategy for the environment, it is necessary to refine the abstraction with the new predicate $x_i > 0$, even though the system wins for any value of $x_i$.

**Contributions.** In this paper, we propose the first CEGAR approach for games with incomplete information. We extend the abstraction of the game with a subset construction on the abstract state space that ensures that the system only uses information it can see. The result is a perfect-information game over a finite game graph that soundly abstracts the original game under incomplete information.

The refinement of the abstraction accounts for two cases: we refine the abstract transition relations by adding new predicates if the environment spuriously wins because it uses moves that are impossible in the concrete game or because moves of the system are impossible in the abstract game but possible in the concrete game; we refine the observation equivalence by adding new predicates if the environment spuriously wins because the abstract system has too little information. To ensure that the new predicates do not mix observable and unobservable variables, we develop a novel constraint-based interpolation technique which provides interpolants that meet arbitrary variable partitioning requirements.

The resulting refinement loop terminates for games for which a finite region algebra (that satisfies certain conditions related to the observation-equivalence) exists. This includes important infinite-state models such as timed games or games defined by bounded rectangular automata, given that the observation-equivalence meets the requirement.

**Related work.** The classic solution to games with incomplete information is the translation to perfect-information games with a knowledge-based subset construction due to Reif [9]. For games over infinite graphs, however, this construction is in general not effective. Our approach is symbolic and is therefore suited to the analysis of games over infinite state spaces. For incomplete-information games with finite state spaces, an alternative would be to first use the knowledge-based subset construction to obtain a perfect-information concrete game and then apply the CEGAR technique of [6] in the usual way. However, since the subset construction leads to an exponential blow-up of the state space of the game, which for realistic systems will make the problem practically infeasible, it is imperative to first use predicate abstraction and obtain a much smaller state space and only then construct the subsets of observation-equivalent prefixes.

**Symbolic fixed-point algorithms** based on antichains were proposed in [5, 2]. In the case of infinite game graphs, these algorithms are applied on a given finite region algebra for the infinite-state game. Our approach, on the other hand, automatically constructs a sufficiently precise finite abstraction.

**Interpolation** was applied successfully in verification for the generation of refinement predicates. There one infers from an unconcretizable abstract counterexample -trace predicates, each of which refers only to variables that describe a single state on that trace. In our case we need to consider sets of traces each of which is concretizable and that are represented symbolically using sets of variables whose intersection contains observable variables only. The straight-forward application of existing interpolation methods ([7, 10]) would produce refinement predicates that are either guaranteed to be observable or guaranteed not to relate two or more states. These approaches are incapable of meeting both guarantee requirements. To this end, we present our extension of the algorithm from [10] which provides interpolants that meet arbitrary variable partitioning requirements.

## 2 Preliminaries

### 2.1 Variables, Predicates and Formulas

We model the communication between a system and its environment by a finite set $X$ of variables, which is partitioned into four pairwise disjoint sets: $X_h, X_i, X_o$ and $\{t\}$. The environment updates
(and can observe) the variables in $X_h$ and $X_i$ and the system updates (and can observe) the variables in $X_o$. The variables in $X_i$ are the input variables for the system, i.e., it can read their value but not update them. The variables in $X_h$ are private variables for the environment, i.e., the system cannot even observe them. The set $X_o$ consists of the output variables of the system which can be only read by the environment. The value of the auxiliary variable $t$ determines whether it is the system's or the environment's turn to make a transition, i.e., the two players take turns in making a transition. The set $X'$ consists of the primed versions of the variables in $X$.

Sets of concrete and abstract states transitions are represented as formulas over some possibly infinite set $AP$ of predicates (atomic formulas) over the variables in $X \cup X'$. For a formula $\varphi$, we denote with $\text{Vars}(\varphi)$ and $\text{Preds}(\varphi)$ the sets of variables and predicates, respectively, that occur in $\varphi$. The set $\text{Obs}(X \cup X')$ is the set of observable variables in $X \cup X'$, formally, $\text{Obs}(X \cup X') = (X \cup X') \setminus (X_h \cup X'_h)$. For a set $\mathcal{P}$ of predicates, the set $\text{Obs}(\mathcal{P})$ consists of the predicates in $\mathcal{P}$ that contain only observable variables.

### 2.2 Game Structures

A game structure with perfect information $C = (S_s, S_c, s_0, R_s, R_e)$ consists of a set of states $S = S_s \cup S_c$, which is partitioned into a set $S_s$ of system states and a set $S_c$ of environment states, a distinguished initial state $s_0 \in S$, and a transition relation $R = R_s \cup R_e$. The transition relation $R_s \subseteq S_s \times S_s$ is the transition relation for the system player. Note that the successor states of a system state are always environment states. The transition relation $R_e \subseteq S_e \times S_s$ is the transition relation for the environment player and an environment state can have both system and environment states as successors. A state $v$ for which there is no state $w \in S$ with $(v, w) \in R$ is called a dead-end.

A game structure with incomplete information $(S_s, S_c, s_0, R_s, R_e)$ additionally defines an observation equivalence $\equiv$ on $S$. The system has partial knowledge about the current state, i.e., it knows the equivalence class of the current state, but not the particular state in this class.

We require that the relation $\equiv$ meets the following requirements. The relation $\equiv$ respects the partitioning of $S$ into $S_s$ and $S_c$: If $v_1 \in S_s$ and $v_2 \in S_c$ then $v_1 \not\equiv v_2$. The system can distinguish between the different successors of a system state: For every $v \in S_s$ and $w_1, w_2 \in S_c$, if $(v, w_1) \in R_s$ and $(v, w_2) \in R_s$ and $w_1 \neq w_2$, then $v_1 \equiv w_1$. The set of available transitions in a system state is the same for all observation-equivalent states: For every states $v_1, v_2 \in S_s$ and $w_1 \in S_c$ such that $v_1 \equiv v_2$ and $(v_1, w_1) \in R_s$, there exists a state $w_2 \in S_c$ such that $w_1 \equiv w_2$ and $(v_2, w_2) \in R_e$.

#### Symbolic representation

We represent game structures symbolically. A symbolic game structure with incomplete information $C = (X, \text{init}, T_s, T_e)$ consists of a set of variables $X$ (partitioned into $X_h, X_i, X_o$ and $\{t\}$), a formula $\text{init}$ over $X$ and formulas $T_s$ and $T_e$ over $X \cup X'$. For simplicity of the presentation, we assume w.l.o.g. that we have singleton sets $X_h = \{x_h\}$, $X_i = \{x_i\}$ and $X_o = \{x_o\}$. The extension to the general case is straightforward.

A symbolic game structure needs to satisfy the following conditions, in order to define a valid game structure according to the above definition.

1. The following formula is valid
   $$\text{init}\{x_h^1/x_h, x_i^1/x_i, x_o^1/x_o, t^1/t\} \land \text{init}\{x_h^2/x_h, x_i^2/x_i, x_o^2/x_o, t^2/t\} \rightarrow x_h^1 = x_h^2 \land x_i^1 = x_i^2 \land x_o^2 \land t^1 \equiv t^2.$$

2. The formula $T_{s}\{x_h^1/x_h\}$ implies $t \equiv 0$, $t^1 \approx x_h^1 \approx x_h$ and $x_i^1 \approx x_i$.

3. The following formula is valid
   $$T_s\{x_i^1/x_i\} \rightarrow T_s\{x_h^2/x_h\}.$$
The set \( \text{Val}(X) \) consists of all total functions that map each variable in \( X \) to its domain. For a formula \( \varphi \) over \( X \), and \( v \in \text{Val}(X) \) we denote with \( \varphi[v] \) the truth value of the formula \( \varphi \) for the valuation \( v \) of the variables. We write \( v \models \varphi \) iff \( \varphi[v] \) is true. For a formula \( \varphi \) over \( X \cup X' \) and valuations \( v \in \text{Val}(X) \) and \( w \in \text{Val}(X') \), the truth value \( \varphi[v, w] \) is defined analogously.

A symbolic game structure \( C = (X, \text{init}, T, O) \) together with corresponding variable domains \( H, I \) and \( O \), defines a game structure with incomplete information \( C = (S_s, S_e, s_0, \equiv, R_s, R_e) \) in the following way. The set \( S_s \) of system states consists of all valuations in \( \text{Val}(X) \) where \( t \) is mapped to 1 and the set \( S_e \) of environment states consists of the valuations in \( \text{Val}(X) \) where \( t \) is mapped to 0. The initial state \( s_0 \) is the single state that satisfies the formula \( \text{init} \) (the uniqueness is implied by condition (0) above).

The formulas \( T_s \) and \( T_e \) define the transition relations, where \( (v, w) \in R_s \) iff \( T_s[v, w] \) is true, and \( (v, w) \in R_e \) iff \( T_e[v, w] \) is true. Conditions (1) and (2) from the definition of symbolic game structure ensure that the transition relations defined in this way meet the corresponding requirements.

Since the variable \( x_h \) cannot be observed by the system, two states are observation-equivalent if they agree on the valuation of the variables \( x_i, x_o \) and \( t \). Clearly, the equivalence relation defined in this way satisfies the first two requirements we imposed. Condition (3) from the definition of symbolic game structure guarantees that the third requirement is also met. Note, that if \( X_h = \emptyset \), then the symbolic game structure defines a game structure with perfect information.

Example. Consider the game structure \( C = (S_s, S_e, s_0, \equiv, R_s, R_e) \) described below, which we are going to use as a running example throughout this paper.

The variables of the symbolic game structure \( C = (X, \text{init}, T_s, T_e) \) are the following. The private environment variables are \( x_h \) and \( y_h \). The single input and output variables are \( x_i \) and \( x_o \), respectively. The formula \( \text{init} \) is

\[
\text{init} = t \approx 0 \land x_h \approx 0 \land y_h \approx 0 \land x_i \approx 0 \land x_o \approx 0
\]

and the formulas \( T_s \) and \( T_e \) are given in Table 1 as disjunctions of formulas each of which is a conjunction of a \textit{guard} (a formula that refers only to variables in \( X \)) and an \textit{update} (a formula that refers to variables in \( X \) and \( X' \)).

<table>
<thead>
<tr>
<th>( T_s ) guard</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \approx 1 )</td>
<td>( t' \approx 0 \land x'_o \approx 0 \land x'_h \approx x_h \land y'_h \approx y_h \land x'_i \approx x_i )</td>
</tr>
<tr>
<td>( t \approx 1 )</td>
<td>( t' \approx 0 \land x'_o \approx 1 \land x'_h \approx x_h \land y'_h \approx y_h \land x'_i \approx x_i )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( T_e ) guard</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t \approx 0 \land x_h \approx 0 )</td>
<td>( t' \approx 1 \land y'_h \approx y_h \land x'_h \neq 0 \land x'_i \approx x'_h \land x'_o \approx x_o )</td>
</tr>
<tr>
<td>( t \approx 0 \land x_h \neq 0 )</td>
<td>( t' \approx 1 \land y'_h \approx 1 \land x'_h \neq 0 \land x'_i \approx x'_h \land x'_o \approx x_o )</td>
</tr>
<tr>
<td>( t \approx 0 \land x_h \neq 0 )</td>
<td>( t' \approx 1 \land y'_h \approx 1 \land x'_h \neq 0 \land x'_i \neq x'_h \land x'_o \approx -x_i \land x'_o \approx x_o )</td>
</tr>
</tbody>
</table>

| Table 1. Transition relation formulas |
The corresponding game structure $\mathcal{C}$ with incomplete information is defined by $\mathcal{C}$ and the domains of the variables which are: $\mathbb{Q}$ for the variables $x_h$, $y_h$ and $x_i$, and $\{0,1\}$ for $x_o$ and $t$. Initially, all variables are set to 0. The system is allowed to update $x_o$ to 0 or 1 from every system state. The environment can initially pick arbitrary values for $x_h$ and $x_i$ as long as they are equal and different from 0. Later, it can either update $x_h$ and $x_i$ to equal values different from 0 and preserve the sign of $x_i$ or, alternatively, set $x_h$ and $x_i$ to different values (both different from 0) and flip the sign of $x_i$. In both cases, it sets $y_h$ to 1.

**Predicate transformers.** In order to be able to perform symbolic analysis of game structures, we require that the following predicate transformers can be defined and effectively computed. For a formula $\varphi$ and $c_0 \in C_0$, $\text{Pre}_s(c_0, \varphi)$ is a formula such that $v \models \text{Pre}_s(c_0, \varphi)$ iff there exists a state $w \models \varphi \land x_o \approx c_0$ such that $(v, w) \in R_s$ and $\text{Pre}_s(\varphi)$ is a formula such that $v \models \text{Pre}_s(\varphi)$ iff there exists a state $w \models \varphi$ such that $(v, w) \in R_e$. The formula $\text{Pre}_s(\varphi)$ is defined as $\forall c_0 \in C_0 \text{Pre}_s(c_0, \varphi)$ and the formula $\text{Enabled}(c_0)$ is defined as $\text{Pre}_s(c_0, \text{true})$.

### 2.3 Safety games

We consider safety games defined by a set of error states, which we assume w.l.o.g. to be a subset of the set $S_e$ of environment states. The objective for the system is to avoid the error states. Clearly, w.l.o.g. we can assume that the set $S_e$ of environment states does not contain dead-ends. A safety game with perfect information (with incomplete information) $G = (\mathcal{C}, E)$ consists of a game structure $\mathcal{C}$ with complete information (with incomplete information) and a set of error states $E$. A symbolic safety game $G = (\mathcal{C}, \text{err})$ consists of a symbolic game structure $\mathcal{C}$ and a formula $\text{err}$ denoting the set of error states.

**Example.** We consider the symbolic safety game $G = (\mathcal{C}, \text{err})$, where $\mathcal{C}$ is the symbolic game structure from our running example and

$$\text{err} = (t \approx 0 \land y_h \not\approx 0 \land x_h \approx x_i \land x_o \approx 0) \lor (t \approx 0 \land y_h \not\approx 0 \land x_h \not\approx x_i \land x_o \approx 1).$$

In order to avoid the error states in the resulting safety game, the system player has to act as follows. Once the environment sets the variable $y_h$ to 1, the system has to assign value 0 to the variable $x_o$ if the variables $x_h$ and $x_i$ have different values and assign value 1 to $x_o$ if $x_h$ and $x_i$ have the same value.

### 2.4 Strategies

**Paths, prefixes and plays.** Let $G = ((S_s, S_e, s_0, \equiv, R_s, R_e), E)$ be a safety game with incomplete information. A path in $G$ is a finite sequence $\pi = v_0v_1\ldots v_n$ of states such that for all $0 \leq j < n$, we have $(v_j, v_{j+1}) \in R$. The length $|\pi|$ of $\pi$ is $n + 1$. For $0 \leq j < |\pi|$, $\pi[j]$ is the $j$-th element of $\pi$ and $\pi[0, j] = v_0 \ldots v_j$. We define $\text{last}(\pi) = \pi[|\pi| - 1]$. A prefix in $G$ is a path $\pi = v_0v_1\ldots v_n$ such that $v_0 = s_0$. We call $\pi$ a system prefix if $\text{last}(\pi) \in S_s$, and an environment prefix otherwise. We denote with $\text{Pre}_s(G)$ the set of prefixes in $\mathcal{C}$, and with $\text{Pre}_e(G)$ and $\text{Pre}_s(\mathcal{C})$ the sets of system and environment prefixes in $G$, respectively.

A play in $G$ is either an infinite sequence $\omega = v_0v_1\ldots v_j\ldots$ such that $v_0 = s_0$ and for all $j \geq 0$, $(v_j, v_{j+1}) \in R$ or a prefix $\pi$ such that $\text{last}(\pi)$ is an error state or a dead-end. For an infinite play $\omega$, $|\omega| = \infty$.

The observation-equivalence $\equiv$ can be extended in a natural way to prefixes and plays. Let $Q_\omega(\text{Pre}_s(G))$ be the set of equivalence classes of prefixes in $G$ w.r.t. $\equiv$ and $Q_\omega(S_s)$ be the set of equivalence classes of environment states.

**Strategies.** A partial function $f_s : \text{Pre}_s(\mathcal{C}) \to S_e$ is non-blocking if $f_s(\pi)$ is defined for every prefix $\pi \in \text{Pre}_s(G)$ for which $\text{last}(\pi)$ is not a dead-end and which is compatible with $f_s$, i.e., for all $0 \leq j < |\pi| - 1$ with $\pi[j] \in S_s$ it holds that $\pi[j + 1] = f_s(\pi[0, j])$. A non-blocking partial function $f_e : \text{Pre}_e(\mathcal{C}) \to S$ is defined analogously.
A strategy for the system is a non-blocking partial function $f_s : \text{Prefs}_s(G) \to S_e$ such that if $f_s(\pi) = v$, then \((\text{last}(\pi), v) \in R_s\). Strategies for the environment are defined analogously.

For a system strategy $f_s$, the set of system prefixes $\text{Prefs}_s(f_s)$ consists of all prefixes $\pi \in \text{Prefs}_s(G)$ such that for all $0 \leq j < |\pi| - 1$, if $\pi[j] \in S_s$, then it holds that $\pi[j + 1] = f_s(\pi[0:j])$. The set $\text{Prefs}(f_s)$ is defined analogously.

A strategy $f_s$ for the system in an incomplete-information game is called consistent iff for all system prefixes $\pi_1, \pi_2 \in \text{Prefs}_s(f_s)$ with $\pi_1 \equiv \pi_2$, it holds that $f_s(\pi_1) \equiv f_s(\pi_2)$.

The outcome of two strategies $f_s$ and $f_e$ for the system and the environment respectively is a play $\omega = \text{Outcome}(f_s, f_e)$ such that for all $0 \leq j < |\omega| - 1$ if $\omega[j] \in S_s$ then $\omega[j + 1] = f_s(\omega[0:j])$ and if $\omega[j] \in S_e$ then $\omega[j + 1] = f_e(\omega[0:j])$. A strategy $f_s$ for the system is winning iff for every strategy $f_e$ for the environment, if $\omega = \text{Outcome}(f_s, f_e)$ then $\omega$ is an infinite play and for every $j \geq 0$, $\omega[j]$ is not an error state. A strategy $f_s$ for the environment is winning iff for every strategy $f_e$ for the system, if $\omega = \text{Outcome}(f_s, f_e)$ then for some $0 \leq j < |\omega|$, $\omega[j]$ is an error state or a dead-end.

**Strategy trees.** A winning strategy $f_e$ for the environment in a safety game $G$ can be naturally represented as a finite tree $T(f_e)$, called strategy tree. Each node in $T(f_e)$ is labeled by a state in $S$, such that the following are satisfied:

1. The root of $T(f_e)$ is labeled by the initial state $s_0$ of $G$.
2. If an internal node is labeled by a state $v$ and a child of that node is labeled by a state $w$, then $w$ is a successor of $v$, i.e., $(v, w) \in R$.
3. If an internal node $\pi$ is labeled by $v \in S_s$, then for every $w \in S$ with $(v, w) \in R_s$, there exists exactly one child of $\pi$ which is labeled by $w$. We denote with $\text{Children}(\pi, T(f_e))$ the set of all children of $\pi$ in $T(f_e)$.
4. If an internal node $\pi$ is labeled by $v \in S_e$, then $\pi$ has exactly one child, denoted $\text{Child}(\pi, T(f_e))$ and labeled by some $w \in S$ with $(v, w) \in R_e$.
5. A node is a leaf if and only if it is labeled by an error state or a dead-end.

Thus, each node in the strategy tree $T(f_e)$ corresponds to a prefix in $G$, and a prefix in $\text{Prefs}(G)$ is represented by at most one node. We identify each node with the corresponding prefix and define $\text{Prefs}(f_e)$ as the set of prefixes in $T(f_e)$.

### 2.5 Knowledge-based subset construction

An incomplete-information game can be translated to a perfect-information game in which the system has a winning strategy if and only if the system has a consistent winning strategy in the original game with incomplete information. The knowledge-based subset construction of an incomplete-information game $G = ((S_s, S_e, \{s_0\}, \equiv, R_s, R_e), E)$ is a perfect-information game $G^k$, where $G^k = ((S_s^k, S_e^k, s_0^k, R_s^k, R_e^k), E^k)$ is defined as follows.

The sets of system and environment states of $G^k$ are $S_s^k = \{V \in 2^{S_s} : \forall v_1, v_2 \in V, v_1 \equiv v_2\}$ and $S_e^k = \{V \in 2^{S_e} : \forall v_1, v_2 \in V, v_1 \equiv v_2\}$, respectively. The initial state of $G^k$ is $s_0^k = \{s_0\}$.

The transition relation for the system player in $G^k$ is defined as: for states $V \in S_s^k$ and $W \in S_e^k$, we have that $(V, W) \in R_s^k$ iff the following conditions hold:

1. for every $v \in V$ there exists a $w \in W$ such that $(v, w) \in R_s$ in the game $G$,
2. for every $w \in W$ there exists a $v \in V$ such that $(v, w) \in R$ in the game $G$,
3. if $w_1 \equiv w_2$, $w_1 \in W$ and there exists a $v \in V$ with $(v, w_2) \in R$, then $w_2 \in W$.

Similarly for the environment transition relation, for $V \in S_e^k$ and $W \in S_e^k \cup S_s^k$ we have $(V, W) \in R_e^k$ iff conditions (1'), (2) and (3) are satisfied, where (1') there exist $v \in V$ and $w \in W$ such that $(v, w) \in R_e$.

The set of error states in $G^k$ is $E^k = \{V \in S_e^k : |V \cap E| \neq \emptyset\}$. 

2.6 The game solving problem

The game solving problem is, given a safety game with incomplete information, to determine whether there exists a consistent winning strategy for the system player. The strategy synthesis problem is to find such a strategy if one exists.

For incomplete-information games with finite state space, the two problems can be solved using the knowledge based subset construction which can be effectively computed in this case. For incomplete-information games with infinite state space, however, this is not the case. Therefore, in the next sections we propose a predicate-abstraction-based approach to these problems.

3 Abstraction

We use two subset constructions to abstract infinite-state games with incomplete information into finite-state games with perfect information: first, we overapproximate the moves of the environment and underapproximate the moves of the system in the abstract domain defined by the predicate valuations. Then, we overapproximate the observation-equivalence based on the observable predicates to obtain a sound abstraction with respect to incomplete information.

3.1 Abstraction Predicates

Let \( \mathcal{G} = (S, err) \) be a symbolic safety game. For a finite set \( \mathcal{P} \) of predicates over \( X \), \( \text{Vals}(\mathcal{P}) \) is the set of all truth valuations of the elements of \( \mathcal{P} \). For \( a \in \text{Vals}(\mathcal{P}) \), \( p \in \mathcal{P} \), \([a]\) is the corresponding formula over \( \mathcal{P} \) and we write \( a \models p \) iff the value of \( p \) in \( a \) is true. Similarly for a formula \( \varphi \) over \( \mathcal{P} \).

The concretization \( \gamma_\mathcal{P}(a) \) of \( a \in \text{Vals}(\mathcal{P}) \) is the set of concrete states \( \{s \in S \mid \forall p \in \mathcal{P} : s \models p \text{ if } a \models p\} \). For \( A \subseteq \text{Vals}(\mathcal{P}) \), we define \( \gamma_\mathcal{P}(A) = \bigcup_{a \in A} \gamma_\mathcal{P}(a) \).

The equivalence \( \equiv_\mathcal{P} \) on the set of valuations \( \text{Vals}(\mathcal{P}) \) is defined as: \( a_1 \equiv_\mathcal{P} a_2 \) iff for every observable variable \( p \in \text{Obs}(\mathcal{P}) \) it holds that \( a_1 \models p \) iff \( a_2 \models p \).

We abstract the concrete game \( \mathcal{G} \) w.r.t. a pair \( \mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s) \) of finite sets of predicates such that \( \text{Preds(init)} \cup \text{Preds(err)} \cup \{t \approx 0\} \subseteq \mathcal{P}_{se} \). The states in \( S_e \) are abstracted w.r.t. \( \mathcal{P}_{se} \) and the states in \( S_s \) are abstracted w.r.t. the full set \( \mathcal{P} = \mathcal{P}_{se} \cup \mathcal{P}_s \). By enhancing at the refinement step the set \( \mathcal{P}_s \) with predicates that are used to split only abstract system states, we ensure that the transition relation for the system in the refined abstract game is not less precise than the transition relation for the system in the current abstract game.

In the following, \( \gamma(a) \) means \( \gamma_{\mathcal{P}_{se}}(a) \) if \( a \in \text{Vals}(\mathcal{P}_{se}) \) and \( \gamma(a) \) if \( a \in \text{Vals}(\mathcal{P}) \). Similarly for the observation-equivalence \( \equiv_\mathcal{P} \).

For two pairs of sets of predicates \( \mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s) \) and \( \mathcal{Q} = (\mathcal{Q}_{se}, \mathcal{Q}_s) \), we write \( \mathcal{P} \subseteq \mathcal{Q} \) iff \( \mathcal{P}_{se} \subseteq \mathcal{Q}_{se} \) and \( \mathcal{P}_s \subseteq \mathcal{Q}_s \), and we define \( \mathcal{P} \cup \mathcal{Q} = (\mathcal{P}_{se} \cup \mathcal{Q}_{se}, \mathcal{P}_s \cup \mathcal{Q}_s) \).

3.2 Abstract Game

The abstraction \( \alpha(\mathcal{G}, \mathcal{P}) \) of a symbolic safety game \( \mathcal{G} = (S, err) \) w.r.t. a pair \( \mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_s) \) of finite sets of predicates is the finite-state perfect-information safety game \( G^\alpha = ((S^\alpha_s, S^\alpha_e, s^\alpha_0, R^\alpha_e, R^\alpha_s), E^\alpha) \) defined below.

**States.** The set \( S^\alpha \) of abstract states is the union of \( S^\alpha_e \subseteq 2^{\text{Vals}(\mathcal{P})} \setminus \{\emptyset\} \) and \( S^\alpha_s \subseteq 2^{\text{Vals}(\mathcal{P}_{se})} \setminus \{\emptyset\} \) which are defined as follows.

The set of abstract system states \( S^\alpha_s \) consists of all nonempty subsets \( A \) of the set of valuations \( \text{Vals}(\mathcal{P}) \) such that for every \( a \in A \), it holds that \( a \not\models t \approx 0 \) and \( \gamma(a) \neq \emptyset \), and for every two valuations \( a_1, a_2 \in A \), it holds that \( a_1 \equiv_\mathcal{P} a_2 \).

The set of abstract environment states \( S^\alpha_e \) consists of all nonempty subsets \( A \) of the set of valuations \( \text{Vals}(\mathcal{P}_{se}) \) such that for every \( a \in A \), it holds that \( a \models t \approx 0 \) and \( \gamma(a) \neq \emptyset \), and for every \( a_1, a_2 \in A \), it holds that \( a_1 \equiv_\mathcal{P} a_2 \).
The initial abstract state \( s_0^a \) consists of the single valuation \( a_0 \) such that \( a_0 \models \text{init} \) and \( \gamma(a_0) \neq \emptyset \).

**Must transitions.** The abstract transition relation \( R_s^a \subseteq S_s^a \times S_s^a \) for the system is defined as: \((A, A') \in R_s^a\) iff the following conditions are satisfied:

1. **(must)** for every valuation \( a \in A \) and every concrete state \( v \in \gamma(a) \), there exist a valuation \( a' \in A' \) and a concrete state \( v' \in \gamma(a') \) with \((v, v') \in R_s\),
2. for every valuation \( a' \) in the target state \( A' \) there exist a valuation \( a \) in the source state \( A \) and concrete states \( v \in \gamma(a) \) and \( v' \in \gamma(a') \) such that \((v, v') \in R_s\),
3. for every \( a_1' \in \text{Vals}(P) \) and \( a_2' \in \text{Vals}(P) \), if \( a_1' \in A' \), \( a_1' \equiv a_2' \) and there exist \( a \in A \), \( v \in \gamma(a) \) and \( v' \in \gamma(a_2') \) with \((v, v') \in R_s\), then \( a_2' \in A' \).

**May transitions.** The abstract transition relation \( R_e^a \subseteq S_e^a \times S_e^a \) for the environment is defined as: \((A, A') \in R_e^a\) iff the following conditions are satisfied:

1. **(may)** there exist a valuation \( a \in A \), a concrete state \( v \in \gamma(a) \), a valuation \( a' \in A' \) and a concrete state \( v' \in \gamma(a') \) such that \((v, v') \in R_e\),
2. for every valuation \( a' \) in the target state \( A' \) there exist a valuation \( a \) in the source state \( A \) and concrete states \( v \in \gamma(a) \) and \( v' \in \gamma(a') \) such that \((v, v') \in R_e\),
3. for every \( a_1' \in \text{Vals}(P_{se}) \) and \( a_2' \in \text{Vals}(P_{se}) \), if \( a_1' \in A' \), \( a_1' \equiv a_2' \) and there exist \( a \in A \), \( v \in \gamma(a) \) and \( v' \in \gamma(a_2') \) with \((v, v') \in R_e\), then \( a_2' \in A' \).

**Error states.** An abstract state \( A \) is an element of the set of abstract error states \( E^a \) iff there exists a valuation \( a \in A \) with \( a \models \text{err} \).

**Example.** Consider our running example and the sets of predicates \( P = (P_{se}, \emptyset) \), where \( P_{se} = \{p_0 = t \equiv 0, p_1 = x_h \approx 0, p_2 = y_h \approx 0, p_3 = x_i \approx 0, p_4 = x_o \approx 0, p_5 = x_h \approx x_i, p_6 = x_o \approx 1, p_7 = x_i < 0\} \). The set of valuations reachable in the abstract game \( \alpha(G, P) \) is given in Table 2, where each valuation is represented by the set of predicates whose value is true in the corresponding valuation.

| \( 40 \) | \{p_0, p_1, p_2, p_3, p_4, p_5\} |
| \( 410 \) | \{p_2, p_4, p_5, p_7\} |
| \( 420 \) | \{p_0, p_2, p_4, p_5, p_7\} |
| \( 423 \) | \{p_0, p_2, p_4, p_5\} |
| \( 430 \) | \{p_4\} |
| \( 432 \) | \{p_5, p_6, p_7\} |
| \( 434 \) | \{p_4, p_7\} |
| \( 436 \) | \{p_5, p_6\} |
| \( 440 \) | \{p_0, p_6\} |
| \( 442 \) | \{p_0, p_4, p_5, p_7\} |
| \( 444 \) | \{p_0, p_4, p_7\} |
| \( 446 \) | \{p_0, p_4, p_5\} |

**Table 2.** Reachable valuations in the abstract game \( \alpha(G, P) \).

The game graph for the corresponding game structure is given on Fig. 1. The states with a dashed line are the system states and the states with a solid line are the environment states. The gray states are the abstract error states.

Note that abstract states \( A \) of the form \( \{a_1, a_2\} \) with \( a_1 \models x_h \approx x_i \) and \( a_2 \models \neg x_h \approx x_i \) are not reachable because the predicate \( x_i < 0 \) is in \( P \).

Thus, there exists a winning strategy \( f_* \) for the system in the abstract perfect information game \( \alpha(G, P) \). The strategy \( f_* \) maps each prefix \( \pi \) with last(\( \pi \)) = \{a\} for some \( a \) to the successor in which \( x_o \approx 0 \) is false if \( a \models x_h \approx x_i \) and to the successor in which \( x_o \approx 0 \) is true if \( a \models \neg (x_h \approx x_i) \).
3.3 Concretization

The concretization $\gamma^k(f_c)$ of a winning strategy $f_c$ for the environment in $G^a$ is a set of winning environment strategies in the knowledge-based game $G^k$.

For an abstract prefix $\pi \in \text{Prefs}(G^k)$, we define

$$\gamma^k(\pi) = \{\pi' \in \text{Prefs}(G^k) \mid |\pi'| = |\pi|, \forall j : 0 \leq j < |\pi| \Rightarrow \pi'[j] \subseteq \gamma(\pi[j])\},$$

$$\gamma(\pi) = \{\pi^c \in \text{Prefs}(G) \mid |\pi^c| = |\pi|, \forall j : 0 \leq j < |\pi| \Rightarrow \pi^c[j] \in \gamma(\pi[j])\}.$$  

The concretization functions for abstract paths are defined analogously.

The concretization $\gamma^k(f_c)$ of the winning abstract environment strategy $f_c$ is the set of all winning environment strategies $f^c_e$ in $G^k$ such that for every $\pi^k \in \text{Prefs}(f^k_c)$ there exists $\pi \in \text{Prefs}(f_c)$ with $\pi^k \in \gamma(\pi)$.

Let $\mathcal{P}$ and $\mathcal{Q}$ be pairs of sets of predicates with $\mathcal{P} \subseteq \mathcal{Q}$. If $\pi$ and $\pi'$ are prefixes in $\alpha(G, \mathcal{P})$ and $\alpha(G, \mathcal{Q})$, respectively, we write $\pi' \leq \pi$ iff $|\pi| = |\pi'|$ and for every $0 \leq j < |\pi|$, $\gamma(\pi[j]) \subseteq \gamma(\pi'[j])$. If $f_c$ and $f'_c$ are winning strategies for the environment in $\alpha(G, \mathcal{P})$ and $\alpha(G, \mathcal{Q})$ respectively, then $f'_c \leq f_c$ iff for every $\pi' \in T(f'_c)$ there exists $\pi \in T(f_c)$ such that $\pi' \leq \pi$.

**Theorem 1 (Soundness of the abstraction).** If $f_s$ is a winning strategy for the system in the abstract perfect-information game $\alpha(G, \mathcal{P})$, then there exists a consistent winning strategy $f^c_s$ for the system in the concrete symbolic game $\mathcal{G}$ with incomplete information.

**Proof.** By definition, the defined abstraction has the following properties.

**Property 1.** If $\pi$ is in $\text{Prefs}_c(G)$ and $\pi^a$ is in $\text{Prefs}_c(\alpha(G, \mathcal{P}))$ such that $\pi \in \gamma(\pi^a)$, and for some concrete state $w \in S$ it holds that $\pi w \in \text{Prefs}(G)$, then there exists an abstract state $A$ such that $\pi^a A \in \text{Prefs}(\alpha(G, \mathcal{P}))$ and $\pi w \in \gamma(\pi^a A)$.

**Property 2.** If $\pi^a$ is an element of $\text{Prefs}(\alpha(G, \mathcal{P}))$ and $\pi_1 \in \gamma(\pi^a)$, then for every $\pi_2 \in \text{Prefs}(G)$ such that $\pi_1 \equiv \pi_2$, it holds that $\pi_2 \in \gamma(\pi^a)$.

**Property 3.** If $f_s$ is a winning strategy for the system in $\alpha(G, \mathcal{P})$, and the equivalence class $\Pi \in Q_\equiv(\text{Prefs}_s(G))$ is such that there exists a prefix $\pi^a \in \text{Prefs}_s(f_s)$ with $\Pi \subseteq \gamma(\pi^a)$, then there exists an equivalence class $V$ in $Q_\equiv(S_s)$ such that for every $\pi \in \Pi$, there exists a $v \in V \cap \gamma(f_s(\pi^a))$ such that $\text{last}(\pi, v) \in R_s$.

Let $f_s$ be a winning strategy for the system in $\alpha(G, \mathcal{P})$. We define a strategy $f^c_s$ for the system in the concrete game as follows. Let $\Pi \in Q_\equiv(\text{Prefs}(G))$ be an equivalence class of concrete system prefixes. If there exists an abstract prefix $\pi^a$ such that $\Pi \cap \gamma(\pi^a) \neq \emptyset$, then according to Property 2, it holds that $\Pi \subseteq \gamma(\pi^a)$ and moreover, for every abstract prefix $\rho^a$ different from $\pi^a$ it holds that $\Pi \cap \rho^a = \emptyset$. If $\pi^a \in \text{Prefs}_s(f_s)$ and $f_s(\pi^a) = A$, then according to Property 3, it holds
that there exists an equivalence class \( V \) in \( Q_s(S_0) \) such that for every \( \pi \in \Pi \), there exists a \( v \in V \cap \gamma(A) \) such that \((\text{last}(\pi), v) \in R_s\). Thus, in this case we fix for \( \Pi \) one such equivalence class \( V \) in \( Q_s(S_0) \) and for every \( \pi \in \Pi \), we define \( f^*_\pi(v) = v \) where \( v \) is the single element of \( V \) such that \((\text{last}(\pi), v) \in R_s\). Otherwise, for each element of \( \Pi \), \( f^*_\pi \) is undefined.

Clearly, \( f^*_\pi \) is a valid strategy, since \( s_0 \in \gamma(s_0^0) \) and \( s_0^0 \in \text{Prefs}(f_{\pi}) \), and therefore by the definition of \( f^*_\pi \) and the above properties, for every prefix \( \pi \in \text{Prefs}(f_{\pi}) \), there exists a prefix \( \pi^0 \in \text{Prefs}(f_{\pi}) \) such that \( \pi \in \gamma(\pi^0) \).

According to the definition of the strategy \( f^*_\pi \), it is consistent. For every environment strategy \( f_{\pi} \) in \( G \), the play \( \text{Outcome}(\pi, f_{\pi}) \) is infinite, according to the three properties and the fact that \( f_{\pi} \) is a winning strategy in \( \alpha(G, \mathcal{P}) \). For every prefix \( \pi \in \text{Prefs}(f^*_\pi) \), there exists a prefix \( \pi^0 \in \text{Prefs}(f_{\pi}) \) such that \( \pi \in \gamma(\pi^0) \). Therefore, \( f^*_\pi \) is winning, since \( f_{\pi} \) is winning. \( \Box \)

4 Abstract Counterexample Analysis

A winning strategy \( f_{\pi} \) for the environment in the game \( \alpha(G, \mathcal{P}) \) is a genuine counterexample if it has a winning concretization in \( G^S \). Otherwise it is called spurious. The analysis of the strategy tree \( T(f_{\pi}) \) constructs a strategy-tree formula \( F(f_{\pi}) \) that is satisfiable iff \( f_{\pi} \) is genuine. The key idea is to symbolically simulate a perfect-information game over the equivalence classes of the prefixes of the concrete game structure \( G \) with incomplete information.

4.1 Trace formulas

Let \( f_{\pi} \) be a winning strategy for the environment in the abstract game \( \alpha(G, \mathcal{P}) \).

**Traces and error paths.** With each node \( \pi \) in \( T(f_{\pi}) \), we associate a set \( \text{Traces}(\pi) \) of traces, where a trace is a finite sequence \( \tau \in C^*_o \) of system outputs. We define \( \text{Traces}(f_{\pi}) = \text{Traces}(s^0_0) \).

Each trace induces a set of concrete error paths in \( G \). If the strategy \( f_{\pi} \) is genuine, then for each \( \tau \in \text{Traces}(f_{\pi}) \), the concrete strategy in \( G^S \) should provide an error path \( \xi_\tau \) in \( G \).

For each node \( \pi \) in \( T(f_{\pi}) \), the set \( \text{Traces}(\pi) \) is defined recursively as follows:

- If \( \pi \) is a leaf environment node (i.e., an error node), then \( \text{Traces}(\pi) = \{\epsilon\} \);
- If \( \pi \) is a leaf system node (i.e., a dead end), then \( \text{Traces}(\pi) = C_o \);
- If \( \pi \) is an internal environment node, then

\[
\text{Traces}(\pi) = \text{Traces}(\text{Child}(\pi, T(f_{\pi}))); \\
\]

- If \( \pi \) is an internal system node, then

\[
\text{Traces}(\pi) = \{c_o\tau \mid c_o \in C_o, \rho \in \text{Children}(\pi, T(f_{\pi})), \tau \in \text{Traces}(\rho)\}. \\
\]

Let \( \tau \) be a trace. A path \( \xi_\tau \) in the concrete game \( G \) is an error path for the trace \( \tau \) if and only if some of the following three conditions is satisfied:

(i) \( \xi[0] = o\tau \) (i.e. the first state is an error state);
(ii) \( \xi[0] \in S_\epsilon \) and \( \xi[1], |\xi| - 1 \) is an error path for \( \tau \);
(iii) \( \xi[0] \in S_\varepsilon, \tau = c_o\sigma, \) and either \( c_o \) is disabled in the state \( \xi[0] \) or \( \xi[1] \) is a \( c_o \)-successor of \( \xi[0] \) and \( \xi[1], |\xi| - 1 \) is an error path for the remaining trace \( \sigma \).

**Indexing of variables.** For each trace \( \tau \in \text{Traces}(f_{\pi}) \) we define a trace formula \( F(f_{\pi}, \tau) \) which is satisfiable iff there is a node \( \rho \in T(f_{\pi}) \) such that there is an error path for \( \tau \) in \( \gamma(\rho) \). Here, unlike in the perfect-information case, the error paths \( \xi_\tau \) and \( \xi_\tau' \) in the concrete strategy for two different traces \( \tau_1, \tau_2 \in \text{Traces}(f_{\pi}) \) may differ even before the position that corresponds to the first position in which \( \tau_1 \) and \( \tau_2 \) are different, as long as their prefixes up to that position are equivalent. We encode this constraint by indexing the variables in the trace formulas.

Consider a trace \( \tau \in \text{Traces}(f_{\pi}) \). Let \( \pi \) be a node in \( T(f_{\pi}) \) and the traces \( \sigma_1 \) and \( \sigma_2 \) be such that \( \tau = \sigma_1\sigma_2 \), the length of \( \sigma_1 \) is equal to the number of system states on the prefix \( \pi \) and \( \sigma_2 \in \text{Traces}(\pi) \). The variables in \( F(f_{\pi}, \tau) \) that represent a concrete state in \( \gamma(\text{last}(\pi)) \) are indexed...
as follows. Each of the variables is indexed by the node π, so that there are different variables in the formula for different nodes. We index each of these variables also with the prefix σ1 of π, so that there are different variables in the formulas for different traces after the first position in which the traces differ. The unobservable variables are additionally indexed with the remaining part σ2 of π, in order to have different unobservable variables for corresponding states in the formulas for different traces even before the first position in which the traces differ.

To this end, with each node π ∈ T(fe) and traces σ1, σ2 ∈ C∗ we associate a set X(π, σ1, σ2) = {xh(π, σ1, σ2), xi(π, σ1), xo(π, σ1), t(π, σ1) | t} of variables and define substitutions which map variables from the original set X ∪ X′ to variables in the sets X(π, σ1, σ2) and vice versa. The substitution σs(π, σ1, σ2) indexes the variables in X w.r.t. π, σ1 and σ2. The substitution subst1(π, π′, σ1, σ2, σ′2) indexes the variables in X w.r.t. π, σ1 and σ2, and the variables in X′ w.r.t. π′, σ1 and σ′2. The substitution substo(c0, π) re-indexes variables by appending c0 to the front of the σ1-component of the indexes of all variables indexed with a node different from π. Finally, substX maps indexed observable variables to variables in X.

For π, π′ ∈ T(fe), c0 ∈ C0 and σ1, σ2, σ′2 ∈ C∗, we define:

\[
\text{subst}_s(\pi, \sigma_1, \sigma_2) = \{ x_h(\pi, \sigma_1, \sigma_2)/x_h, x_i(\pi, \sigma_1)/x_i, x_0(\pi, \sigma_1)/x_0, t(\pi, \sigma_1)/t \},
\]

\[
\text{subst}_1(\pi, \pi', \sigma_1, \sigma_2, \sigma_2') = \{ x_h(\pi, \sigma_1, \sigma_2)/x_h, x_i(\pi, \sigma_1)/x_i, x_0(\pi, \sigma_1)/x_0, t(\pi, \sigma_1)/t, \]
\[
\quad x_h(\pi', \sigma_1, \sigma_2')/x_i', x_i(\pi', \sigma_1)/x_i, x_0(\pi', \sigma_1)/x_0, t(\pi', \sigma_1)/t' \},
\]

\[
\text{subst}_o(c_0, \pi) = \{ x_i(\pi, \sigma_1, \sigma_2)/x_i, x_0(\pi, \sigma_1, \sigma_2)/x_0, t(\pi, \sigma_1, \sigma_2)/t \} \}
\]

\[
\text{subst}_X = \{ x_i/x_i, x_0/x_0, t/t(\pi, \sigma_1, \sigma_2) | \pi \in T(fe), \sigma \in C^0 \}.
\]

**Trace formulas.** We define recursively a trace formula F(π, τ) for every node π ∈ T(fe) and trace τ. If τ ∈ Traces(π), then we consider three cases that correspond to the three cases in the definition of error paths. The auxiliary formulas ErrorState, EnvTrans, Disabled and SystTrans account for cases (i), (ii) and the two parts of case (iii) in the definition of error paths, respectively.

- If π is a leaf environment node, then τ = ε and we define
  \[ F(\pi, \epsilon) = \text{ErrorState}(\pi). \]

- If π is an internal environment node and π′ = Child(π, T(fe)), then we define
  \[ F(\pi, \tau) = \text{EnvTrans}(\pi, \pi', \tau). \]

- If π is a system node, then τ = c0σ for some c0 ∈ C0 and σ ∈ C∗. Then,
  - if π is a leaf node, we define
    \[ F(\pi, c_0\sigma) = \text{Disabled}(\pi, c_0\sigma); \]
  - if π is an internal node, we define
    \[ F(\pi, c_0\sigma) = \text{Disabled}(\pi, c_0\sigma) \lor \bigvee_{\pi' \in \text{Children}(\pi, T(fe))} \text{SystTrans}(\pi, \pi', c_0\sigma). \]

We need to define a formula F(π, τ) also for the case when τ ∉ Traces(π). In this case, F(π, τ) is obtained from F(π, σ) by adjusting the indices accordingly w.r.t. τ, where σ is the maximal prefix of τ such that σ ∈ Traces(π), if such a prefix exists, and F(π, τ) = false otherwise.

In the formula ErrorState which accounts for case (i) from the definition of error path, π stands for a leaf environment node in T(fe). In the formula EnvTrans, which corresponds to case (ii), π
stands for an environment node in \( T(f_e) \) and \( \pi' \) stands for \( \text{Child}(\pi, T(f_e)) \). Finally, in the formulas \( \text{Disabled} \) and \( \text{SystTrans} \), which account for case (iii), \( \pi \) stands for a system node and \( \pi' \) stands for an environment node in \( \text{Children}(\pi, T(f_e)) \). In all the formulas, \( \tau \) (or \( c_o, \sigma \)) stands for an element of \( \text{Traces}(\pi) \). We define:

\[
\text{ErrorState}(\pi) = \bigvee_{a \in \text{last}(\pi), a=err}[a] \text{subst}_s(\pi, \varepsilon, \varepsilon)
\]

\[
\text{EnvTrans}(\pi, \pi', \tau) = [\text{last}(\pi)] \text{subst}_s(\pi, \varepsilon, \tau) \lor [\text{last}(\pi')] \text{subst}_s(\pi', \varepsilon, \tau) \land T_s \text{subst}_s(\pi, \pi', \varepsilon, \tau, \tau) \land F(\pi', \tau)
\]

\[
\text{Disabled}(\pi, c_o, \sigma) = ([\text{last}(\pi)] \land \neg \text{Enabled}(c_o)) \text{subst}_s(\pi, \varepsilon, c_o, \sigma)
\]

\[
\text{SystTrans}(\pi, \pi', c_o, \sigma) = [\text{last}(\pi)] \text{subst}_s(\pi, \varepsilon, c_o, \sigma) \land ([\text{last}(\pi')] \text{subst}_s(\pi', \varepsilon, \sigma) \land x_o^{(\pi', \varepsilon)} \approx c_o \land T_s \text{subst}_s(\pi, \pi', \varepsilon, c_o, \sigma, \sigma) \land F(\pi', \sigma)) \text{subst}_o(c_o, \pi).
\]

The definition of a trace formula implies that the trace formula \( F(\pi, \tau) \) is satisfied by a sequence \( \xi \) of concrete states if \( \xi \) is an error path for \( \tau \) and there exists a node \( \rho \) in the subtree of \( T(f_e) \) below the node \( \pi \) such that \( \xi \in \gamma(\rho) \).

\[
\begin{align*}
\text{Algorithm:} & \text{annotates a strategy tree with sets of traces and formulas} \\
\text{Input:} & \text{symbolic game } G = (G, \text{err}), \text{ finite sets of predicates } P = (P_{se}, P_s), \\
& \text{strategy tree } T(f_e) \text{ of a winning environment strategy in } \alpha(G, P) \\
\text{Output:} & \text{annotated tree } (T(f_e), \text{Traces}, F) \\
\text{forall nodes } \pi & \text{ in } T(f_e) \text{ in a bottom-up manner do} \\
& \text{if } \pi \text{ is a leaf environment node then} \\
& \text{Traces}(\pi) := \{\varepsilon\}; \\
& F(\pi, e) := \text{ErrorState}(\pi); \\
& \text{else} \\
& \text{if } \pi \text{ is a leaf system node then} \\
& \text{Traces}(\pi) := C_o; \\
& \text{forall } \tau \in \text{Traces}(\pi) \text{ do} \\
& \quad F(\pi, \tau) := \text{Disabled}(\pi, \tau); \\
& \text{else} \\
& \text{if } \pi \text{ is an environment node then} \\
& \text{Traces}(\pi) := \text{Traces}(\text{Child}(\pi)); \\
& \text{forall } \tau \in \text{Traces}(\pi) \text{ do} \\
& \quad F(\pi, \tau) := \text{EnvTrans}(\pi, \pi', \text{Child}(\pi, T(f_e)), \tau) \\
& \text{else} \\
& \text{Traces}(\pi) := \{c_o \tau \mid c_o \in C_o, \pi' \in \text{Children}(\pi, T(f_e)), \\
& \quad \tau \in \text{Traces}(\pi')\}; \\
& \text{forall } \tau \in \text{Traces}(\pi) \text{ do} \\
& \quad F(\pi, \tau) := \text{Disabled}(\pi, \tau) \lor \bigvee_{\pi' \in \text{Children}(\pi, T(f_e))} \text{SystTrans}(\pi, \pi', \tau)
\end{align*}
\]

\text{return } (T(f_e), \text{Traces}, F)

4.2 Strategy-tree formula

As a strategy that concretizes \( f_e \) needs to provide an error path for every trace \( \tau \in \text{Traces}(f_e) \), the strategy-tree formula is defined to be the conjunction of the trace formulas for the root node
of \(T(f_e)\). Formally, we define the strategy-tree formula to be \(F(f_e) = \bigwedge_{\tau \in \text{Traces}(f_e)} F(f_e, \tau)\) where \(F(f_e, \tau) = F(s_0^\tau, \tau)\) is the trace formula for \(\tau \in \text{Traces}(f_e)\). The strategy tree formula is constructed by the algorithm outlined above, which annotates the nodes of the strategy tree in a bottom-up manner with the corresponding sets of traces and trace formulas.

The following theorem is a straightforward consequence of the definition of the strategy-tree formula \(F(f_e)\) for a winning environment strategy \(f_e\) in \(\alpha(\mathcal{G}, \mathcal{P})\).

**Theorem 2.** Let \(f_e\) be a winning strategy for the environment in the abstract game \(\alpha(\mathcal{G}, \mathcal{P})\). The strategy-tree formula \(F(f_e)\) is satisfiable iff the strategy \(f_e\) is genuine, i.e. it has a non-empty concretization \(\gamma^f(f_e)\).

![Strategy tree for the spurious abstract environment strategy \(f_e\).](image)

**Example.** Consider the symbolic safety game \((\mathcal{C}, \text{err})\) from our running example and the initial pair of sets of predicates \(\mathcal{P}_0 = (\mathcal{P}_0^e, \mathcal{P}_0^a)\), where \(\mathcal{P}_0^e\) consists of the predicate \(t \approx 0\) and the predicates which occur in init and err, i.e., \(\mathcal{P}_0^e = \{p_0 = t \approx 0, p_1 = x_h \approx 0, p_2 = x_i \approx 0, p_3 = x_o \approx 0, p_4 = y_h \approx 0, p_5 = x_i \approx x_h, p_6 = x_o \approx 1\}\) and \(\mathcal{P}_0^a = \emptyset\).

| \(a_0\) | \(\{p_0, p_1, p_2, p_3, p_4, p_5\}\) |
| \(a_20\) | \(\{p_0, p_2, p_4, p_5\}\) |
| \(a_30\) | \(\{p_4, p_5\}\) |
| \(a_31\) | \(\{p_4\}\) |
| \(a_{40}\) | \(\{p_0, p_4\}\) |
| \(a_{42}\) | \(\{p_0, p_4\}\) |

**Table 3.** Valuations in the strategy tree \(T(f_e)\).

In the abstract game \(\alpha(\mathcal{G}, \mathcal{P}^a)\), the environment has a winning strategy \(f_e\), whose strategy tree \(T(f_e)\) is given on Fig. 2 and the valuations that are elements of the corresponding states are given in Table 3.

We have the following sets of traces for the nodes in \(T(f_e)\):

\[
\begin{align*}
\text{Traces}(\pi_{40}) &= \text{Traces}(\pi_{41}) = \text{Traces}(\pi_{42}) = \text{Traces}(\pi_{43}) = \{\epsilon\}, \\
\text{Traces}(\pi_{20}) &= \text{Traces}(\pi_{21}) = \text{Traces}(\pi_{30}) = \text{Traces}(\pi_{31}) = \{0, 1\}, \\
\text{Traces}(\pi_0) &= \text{Traces}(\pi_1) = \{00, 01, 10, 11\}.
\end{align*}
\]

Each of the trace formulas \(F(f_e, 00), F(f_e, 01), F(f_e, 10)\) and \(F(f_e, 11)\) for the strategy tree is satisfiable. On Fig. 2 we have indicated the sequences of abstract valuations corresponding to the concrete error paths. However, the conjunction of the trace formulas, i.e., the strategy-tree formula \(F(f_e)\) is unsatisfiable. Therefore, the strategy \(f_e\) is a spurious abstract counterexample.
5 Counterexample-Guided Refinement

If the abstraction predicates in \( P = (P_{se}, P_s) \) are such that there exists a spurious winning strategy \( f_e \) for the environment in the abstract game \( \alpha(G, P) \), we enhance \( P_{se} \) and \( P_s \) with sets of predicates \( R_{se}(f_e) \) and \( R_s(f_e) \), respectively. The refinement predicates are computed w.r.t. the strategy \( f_e \) and such that in the refined game \( \alpha(G, (P_{se} \cup R_{se}(f_e), P_s \cup R_s(f_e))) \) the environment has no winning strategy subsumed by \( f_e \), i.e. a winning strategy \( f'_e \) with \( f'_e \leq f_e \).

We define two functions that map a set of traces and a set of predicates respectively to set of predicates. These functions are used to compute some of the refinement predicates. For a finite set \( T \) of traces, we define \( \text{OutPreds}(T) = \{ x_o \approx \tau[j] \mid \tau \in T, 0 \leq j < \tau \} \). For a finite set of predicates \( Q \), we define \( \text{PredsSyst}(Q) = \bigcup_{a \in \text{Vals}(	ext{Obs}(Q))} \text{Preds}(	ext{Pre}(s[a])) \).

In the following two sections we consider the two possible sources of spuriousness of a counterexample: the coarseness of the approximation of the transition relations and the coarseness of the approximation of the observation-equivalence.

5.1 Refining the Abstract Transition Relations

If for some \( \tau_0 \in \text{Traces}(f_e) \) the formula \( F(f_e, \tau_0) \) is unsatisfiable, then the occurrence of the spurious abstract strategy is due to the approximations of the transition relations. Therefore we compute refinement predicates for eliminating the approximations that cause the existence of \( f_e \). Such predicates are determined by a bottom-up analysis of the strategy tree \( T(f_e) \) that annotates each node \( \pi \) in the tree with a formula \( \tilde{F}(\pi, \tau) \) for each trace \( \tau \in \text{Traces}(\pi) \). The formula \( \tilde{F}(\pi, \tau) \) denotes the subset of \( \gamma(\pi) \) that consist of those concrete states from which there exists a concrete path that satisfies \( F(\pi, \tau) \). We denote with \( \text{RPGG}(f_e) \) (Refinement Predicates for the Game Graph) the pair \( (\text{RPGG}_{se}(f_e), \text{RPGG}_s(f_e)) \) of sets of predicates computed at this step and used to enhance \( P_{se} \) and \( P_s \), respectively.

**State formulas.** For a node \( \pi \in T(f_e) \) and a trace \( \tau \in \text{Traces}(\pi) \), we define the state formula \( \tilde{F}(\pi, \tau) \) as follows:

- if \( \pi \) is a leaf environment node, then \( \tau = \epsilon \) and we define
  \[
  \tilde{F}(\pi, \epsilon) = \bigvee_{a \in \text{last}(\pi), a = \text{err}} [a];
  \]
- otherwise, \( \tau = c_o \sigma \) for some output \( c_o \in C_o \) and trace \( \sigma \in C^*_o \) and
  - if \( \pi \) is an environment node we define
    \[
    \tilde{F}(\pi, \tau) = [\text{last}(\pi)] \wedge \text{Pre}_c(\tilde{F}(\text{Child}(\pi, T(f_e)), \tau));
    \]
  - if \( \pi \) is a system node, we define
    \[
    \tilde{F}(\pi, \tau) = [\text{last}(\pi)] \wedge (\neg\text{Enabled}(c_o) \vee \text{Pre}_s(c_o, \bigvee_{\pi' \in \text{Children}(\pi, T(f_e))} \tilde{F}(\pi', \sigma))).
    \]

We define \( \tilde{F}(\pi, \tau) \) also for \( \tau \notin \text{Traces}(\pi) \). In that case, \( \tilde{F}(\pi, \tau) \) is \( F(\pi, \sigma) \), where \( \sigma \) is the maximal prefix of \( \tau \) such that \( \sigma \in \text{Traces}(\pi) \) if such exists, and \( \text{false} \) otherwise.

**Refinement predicates.** The sets \( \text{RPGG}_{se}(f_e) \) and \( \text{RPGG}_s(f_e) \) of refinement predicates are computed by the algorithm \( \text{RPGG} \). It annotates the nodes in the strategy tree \( T(f_e) \) with the corresponding sets of state formulas and collects the predicates occurring in these formulas. These predicates are added to \( \text{RPGG}_{se}(f_e) \). To this set we add also the elements of \( \text{OutPreds}(\{\tau_0\}) \) for a trace \( \tau_0 \in \text{Traces}(f_e) \) for which \( F(f_e, \tau_0) \) is unsatisfiable. Thus, the refined abstraction is precise w.r.t. the outputs from \( \tau_0 \). The set \( \text{RPGG}_s(f_e) \) is equal to \( \text{PredsSyst}(P_{se} \cup \text{RPGG}_{se}(f_e)) \). By refining with these predicates we ensure that the transition relation for the system in the refined abstract game is not less precise than the transition relation for the system in the current abstract game.

*Example.* As already mentioned, in our running example, each of the trace formulas for the strategy \( f_e \), whose strategy tree is depicted in Fig. 2, is satisfiable. Thus, if we would use algorithm \( \text{RPGG} \) to compute the refinement predicates, the result would be a set of refinement predicates which are insufficient to exclude the strategy \( f_e \) from subsequent abstractions.
5.2 Refining the Abstract Observation Equivalence

Now we consider the case when each of the trace formulas for the spurious environment strategy $f_e$ is satisfiable. Then the predicates from $\text{RPGG}(f_e)$ do not suffice in general to eliminate the counterexample strategy, because the reason for its existence is the coarseness of the abstract observation-equivalence. In this subsection we propose an algorithm $\text{RPOE}$ (Refinement Predicates for the Observation Equivalence) for computing a set of observable refinement predicates that allow us to distinguish the concrete error paths for different traces.

For the rest of this paper we fix the underlying theory to be the theory of linear arithmetic. The set $\mathcal{AP}$ is the set of linear inequalities and equalities with rational coefficients.

The predicates are obtained from interpolants for unsatisfiable conjunctions of trace formulas. According to the construction of these formulas, they share only observable variables and hence the computed interpolants contain only observable predicates. The key challenge for the interpolant computation in our case is to ensure that these predicates are localized, i.e., that the variables which occur in an atom correspond to a single concrete state and not to a sequence of concrete states. We extend the algorithm LI from [10], which reduces the computation of interpolants for linear arithmetic to linear programming problems, in order to handle this additional condition on the variable occurrences. Our more general algorithm LILA (Linear Interpolation with Localized Atoms) receives in addition a partitioning of the variables which occur in the input systems of inequalities and as a result, each atom in the generated interpolant is guaranteed to contain variables from exactly one partition. We first present the algorithm $\text{RPOE}$ and then give the procedure LILA.

**Partitioning of the set of indexed variables.** In order to avoid refinement predicates that talk about different states on a path, we partition the set $\bigcup_{\tau \in \text{Traces}(f_e)} X^{(\tau, \sigma_1, \sigma_2)}$ of all indexed variables according to the length of the prefixes $\pi$ with which these variables are indexed. For $j \in \mathbb{N}$, $X^j$ is the union of all sets $X^{(\pi, \sigma_1, \sigma_2)}$ with $|\pi| = j$. Thus, each partition consists only of variables that correspond to one level of the strategy tree. In this way, we partition the set of variables $\text{Vars}(F(f_e, \tau))$ occurring in each trace formula $F(f_e, \tau)$. For a formula $\phi$, we denote with $\text{Vars}^j(\phi)$ the partition $\text{Vars}(\phi) \cap X^j$. The index $\text{MaxIx}(\phi)$ is the maximal $j$ with $\text{Vars}^j(\phi) \neq \emptyset$. 

```plaintext
Algorithm: RPGG

Input: symbolic safety game $(S, err)$, finite sets of predicates $\mathcal{P} = (\mathcal{P}_{se}, \mathcal{P}_o)$, strategy tree $T(f_e)$ of a spurious abstract winning environment strategy trace $\tau_0 \in \text{Traces}(f_e)$ such that the formula $F(f_e, \tau_0)$ is unsatisfiable
Output: pair $R = (R_{se}, R_o)$ of finite sets of refinement predicates

$R_{se} := \emptyset$; 
for all nodes $\pi$ in $T(f_e)$ in a bottom-up manner do 
  if $\pi$ is a leaf environment node then 
    $F(\pi, \epsilon) := \bigvee_{a \in \text{last}(\pi), a \neq \epsilon} [a]$;  
    $R_{se} := R_{se} \cup \text{Preds}(F(\pi, \epsilon))$;  
  else 
    forall $\tau \in \text{Traces}(\pi)$ do  
      if $\pi$ is an environment node then 
        $F(\pi, \tau) := [\text{last}(\pi)] \land \text{Pre}_e(\tilde{F}(\text{Child}(\pi, T(f_e)), \tau))$;  
      else 
        $\tau$ is $c_o\sigma$ for some $c_o \in C_o$ and $\sigma \in C_o^*$; 
        $F(\pi, c_o\sigma) := [\text{last}(\pi)] \land (\neg \text{Enabled}(c_o) \lor \text{Pre}_e(c_o, \bigvee_{\pi' \in \text{Children}(\pi, T(f_e))} F(\pi', \sigma)))$;  
        $R_{se} := R_{se} \cup \text{Preds}(F(\pi, \tau))$;  
    return $(R_{se}, \text{PredsSyst}(\mathcal{P}_{se} \cup R_{se}))$
```
Algorithm: RPOE

Input: symbolic game $G = (C, err)$, finite sets of predicates $P = (P_{se}, P_s)$, strategy tree $T(f_e)$ of a winning environment strategy in $\alpha(G, P)$.

Output: pair $R = (R_{se}, R_s)$ of finite sets of refinement predicates.

$$R_{se} := \emptyset; \quad \Phi := \{ F(f_e, \tau) \mid \tau \in \text{Traces}(f_e) \};$$

while all elements of $\Phi$ are satisfiable do

pick nonempty $\Psi \subseteq \Phi$, such that $\psi := \bigwedge_{\phi \in \Psi} \phi$ is satisfiable and $\varphi \in \Phi \setminus \Psi$ such that $\varphi \land \psi$ is unsatisfiable;

$n := \max(\text{MaxIx}(\varphi), \text{MaxIx}(\psi));$

$\theta := \text{LILA}(\varphi, \psi, (\text{Vars}^0(\varphi) \cup \text{Vars}^0(\psi), \ldots, \text{Vars}^n(\varphi) \cup \text{Vars}^n(\psi)));$

if $R_{se} = \emptyset$ then $R_{se} := \text{OutPreds}({\tau \mid F(f_e, \tau) \in \{\varphi \cup \Psi\}});$/$^*$

$R_{se} := R_{se} \cup \text{OutPreds}(\theta) \text{ subst}_X;$

$\Phi := \{ \theta \land \phi \mid \phi \in \Psi \};$

return $(R_{se}, \text{PredsSys}(P_{se} \cup R_{se}));$/$^*$

Distinguishing abstract prefixes. Let $\Phi$ be the set of all trace formulas for the strategy $f_e$, i.e., $\Phi = \{ F(f_e, \tau) \mid \tau \in \text{Traces}(f_e) \}$. As each trace formula $F(f_e, \tau)$ is satisfiable, but the strategy-tree formula $F(f_e)$, which is the conjunction of all trace formulas, is not, there exist a nonempty subset $\Psi \subseteq \Phi$ and a formula $\varphi \in \Phi$ such that the conjunction $\bar{\psi} = \bigwedge_{\phi \in \Psi} \phi$ of the elements of $\Psi$ is satisfiable and the conjunction $\varphi \land \bar{\psi}$ of $\varphi$ and $\bar{\psi}$ is unsatisfiable. This means that there does not exist a set of error paths that satisfies this conjunction. Therefore, each set of error paths which satisfy the formula $\psi$ and each error path that satisfies the formula $\varphi$ can be distinguished.

We transform $\varphi$ and $\psi$ into disjunctions of mixed systems of linear inequalities: into $\bigvee_k A_k x \leq a_k$ and $\bigvee_j B_j x \leq b_j$ respectively. We then apply algorithm LILA described in the next paragraph to compute an interpolant $\theta_{kl}$ for each pair of disjuncts $A_k$ and $B_l$ and obtain the interpolant $\theta = \bigvee_k \bigwedge_l \theta_{kl}$ for the pair of formulas $\varphi$ and $\psi$. Each atomic formula which occurs in $\theta$ is an inequality of the form $i x < \delta$ where $< \in \{ \leq, < \}$ and the only variables which occur in such an inequality are among the observable variables in some set $X$, i.e. the coefficients in front of all other variables are 0. By applying the substitution $\text{subst}_X$ to each of the atomic formulas in $\theta$, we obtain a set of predicates over observable variables from the original set of variables $X$.

Although the conjunction of the interpolant $\theta$ and the formula $\psi$ is unsatisfiable, it might be the case that each of the formulas $\phi \land \theta$ for $\phi \in \Psi$ is satisfiable. In this case it is possible that the refinement predicates obtained only from the interpolant $\theta$ are insufficient. In order to guarantee that the counterexample strategy $f_e$ is excluded from subsequent abstractions, we update $\Phi$ to be the set of conjunctions $\theta \land \phi$, where $\phi \in \Psi$. Then, the process is repeated with the new set $\Phi$ until at some iteration there is a formula $\phi$ in the current set $\Psi$ such that the conjunction of $\phi$ and the interpolant computed at that iteration is unsatisfiable (which is guaranteed to be the case if $\Psi$ is a singleton).

The set $\text{RPOE}_{se}(f_e)$ of refinement predicates consists of all atomic formulas that occur in the computed interpolants, plus the set of output predicates for the traces corresponding to the formulas in the initial set $\Psi \cup \{ \varphi \}$. As in algorithm RPGG, the predicates in $\text{RPOE}_{se}(f_e)$ ensure that the transition relation for the system in the refined abstract game is not less precise than the transition relation for the system in the current abstract game.

Computing interpolants with localized atoms. We now present the algorithm LILA for computing localized interpolants for linear arithmetic. A mixed system, denoted $Ax \leq a$, consists of strict and non-strict linear inequalities.

The input of algorithm LILA from [10] consists of two mixed systems of inequalities $Ax \leq a$ and $Bx \leq b$ such that the conjunction $Ax \leq a \land Bx \leq b$ is not satisfiable. The output is a linear interpolant $ix < \delta$ where $\delta \in \{ \leq, < \}$. 

16
The conjuncts of $\theta$ exists, then the algorithm will find one.

There exist row vectors $\delta$ a conjunction $Bx \leq \{0\}$ of the variables in $x$ and only variables from $V^j$ occur in $i_j x < j_\delta$.

\[ \chi_1 := \lambda \geq 0 \land \mu \geq 0 \land \lambda A + \mu = 0; \]
\[ \chi_2 := \lambda = \sum_{j \in \lambda} \lambda_j \land \lambda_j \geq 0 \land i_j = \lambda \land \delta_j = \lambda \land \Lambda_k \in \lambda; \lambda_j A \land k = 0; \]

If exist $\lambda, \mu, \lambda, i_j, \delta_j$, for $0 \leq j \leq n$ satisfying $\chi_1 \land \chi_2 \land \lambda + \mu b \leq -1$
then return $\Lambda^j_{i_j} \leq \delta_j$;

elif exist $\lambda, \mu, \lambda, i_j, \delta_j$, for $0 \leq j \leq n$ satisfying $\chi_1 \land \chi_2 \land \lambda + \mu b \leq 0 \land \lambda^b \neq 0$
then return $\Lambda^j_{i_j} \leq \delta_j$;

else return $\perp$.

Algorithm LILA receives in addition a partitioning $(V^0, V^1, \ldots, V^n)$ of the variables in the vector $x$. The output is an interpolant for $Ax \leq a$ and $Bx \leq b$ which is of the form $\bigwedge_{i=0}^n i_j x < j_\delta$, where $i_j \in \{\leq, <\}$ and for each $0 \leq j \leq n$, only variables from $V^j$ occur in $i_j x < j_\delta$. If the algorithm does not find such an interpolant, the element $\perp$ is returned.

Each of the inequalities $i_j x < j_\delta$ is a linear combination of inequalities in $Ax \leq a$. The variables $\lambda_0, \lambda_1, \ldots, \lambda_n$ denote vectors which define these linear combinations and the variable $\lambda$ stands for the sum of $\lambda_0, \lambda_1, \ldots, \lambda_n$. The subvectors $\lambda^k, \lambda^k, \lambda^k$ for $j = 0, 1, \ldots, n$ define linear combinations of strict and non-strict inequalities in $Ax \leq a$, respectively. Similarly for $\mu, \mu^b, \mu^b$.

For each $0 \leq j \leq n$, the set of variables $V^j$ defines a set $Ix(j)$ of indices that consists of the indices of the columns in the matrix $A$ that correspond to the variables in the set $V^j$. Formally, $Ix(j) = \{ k \mid k \in \{i, \ldots, m_A\}, x_k \in V^j \}$, where $m_A$ is the number of columns in $A$. The complement $\{i, \ldots, m_A\} \setminus Ix(j)$ of the set $Ix(j)$ is denoted with $Ix(j)$. Then, the constraint that the coefficients in $i_j x < j_\delta$ of all variables in $x$ that are not in $V^j$ are $0$ can be expressed as $\Lambda_{k \in Ix(j)} \Lambda_j A_k = 0$, where for $1 \leq k \leq m_A$, $A_k$ is the $k$-th column of $A$.

For disjunctions of mixed systems, i.e., for formulas in the form $\bigvee_{k} A_k x \leq a_k$ and $\bigvee_{l} B_l x \leq b_l$, we proceed as in [10]: we compute an interpolant $\theta_{kl}$ for each pair of disjuncts and then take $\bigvee_{k} A_k x \leq a_k$ and $\bigvee_{l} B_l x \leq b_l$.

Example. In our running example, each of the trace formulas $F(f_e, 00)$, $F(f_e, 01)$, $F(f_e, 10)$ and $F(f_e, 11)$ for the strategy $f_e$ in the abstract game $\alpha(G, P_0)$ is satisfiable. The conjunction of the two trace formulas $F(f_e, 00)$ and $F(f_e, 01)$, however, is not. We convert each of the two formulas into a disjunction of mixed systems of inequalities and apply algorithm LILA to compute for each pair of systems the corresponding interpolant. Thus, the procedure RPOE derives the predicate $x_i < 0$. This predicate yields sufficient precision as we obtain an abstract game with the same game graph as the one depicted on Fig. 1. Thus, in the refined abstraction, the system has a winning strategy.

Theorem 3 (Soundness and completeness of LILA). Algorithm LILA is sound: If it returns a conjunction $\theta = \bigwedge_{i=0}^n \theta_j$, then $\theta$ is an interpolant for the pair of mixed systems $Ax \leq a$ and $Bx \leq b$ with the following properties: (1) for each $j$, $\theta_j$ is of the form $i_j x < j_\delta$ where $i_j \in \{\leq, <\}$; (2) there exist row vectors $\lambda_0, \ldots, \lambda_n$ such that for every $0 \leq j \leq n$, $\lambda_j \geq 0$, $i_j = \lambda_j A$ and $\delta_j = \lambda_j a$; (3) for each $0 \leq j \leq n$, only variables from $V^j$ occur in $\theta_j$.

Algorithm LILA is complete: if an interpolant $\theta = \bigwedge_{i=0}^n \theta_j$ with the properties (1), (2) and (3) exists, then the algorithm will find one.

Proof. Soundness. Assume that algorithm LILA returns the conjunction $\theta = \bigwedge_{i=0}^n \theta_j$. Clearly, the conjuncts of $\theta$ satisfy condition (1). According to the constraints on $i_j$ and $\delta_j$ for $j = 0, \ldots, n$,
there exist row vectors $\lambda_0, \ldots, \lambda_n$ with the properties form condition (2). Since the row vectors $\lambda_0, \ldots, \lambda_n$ satisfy the constraint $\chi_2$, condition (3) is also satisfied. It remains to show that $\theta$ is indeed an interpolant for $Ax \leq a$ and $Bx \leq b$. As each $\theta_j$ is of the form $\lambda_j Ax + j \lambda_i a$, we have that $Ax \leq a$ implies $\theta_j$ for every $j$. Thus, $Ax \leq a$ implies $\theta$. We have $\lambda = \sum_{j=0}^n \lambda_j$ and by [10], $\lambda Ax < \lambda a$ is an interpolant for $Ax \leq a$ and $Bx \leq b$. Since $\theta$ implies $\lambda Ax < \lambda a$, the conjunction of $\theta$ and $Bx \leq b$ is unsatisfiable.

Completeness. Let $\theta = \bigwedge_{j=0}^n \theta_j$ be an interpolant for $Ax \leq a$ and $Bx \leq b$ that satisfies the conditions (1), (2) and (3). Consider $\theta$ as a mixed system $Ix \leq d$. Since the conjunction of $\theta$ and $Bx \leq b$ is unsatisfiable, there exist row vectors $\lambda' = (\lambda'_0, \ldots, \lambda'_n) \geq 0$ and $\mu' \geq 0$, such that $XI + \mu' B = 0$ and one of the following is satisfied: $\lambda'd + \mu'b \leq -1$; or $\lambda'd + \mu'b \leq 0$ and $\lambda'^t \neq 0$; or $\lambda'd + \mu'b \leq 0$ and $\lambda'^t \neq 0$. Let $\tilde{\lambda}_j = \lambda'_j \lambda_j$ for each $0 \leq j \leq n$, $\lambda = \sum_{j=0}^n \tilde{\lambda}_j$ and $\mu = \mu'$. We have $XI = \lambda A$ and $\lambda'd = \lambda a$. Also, if $\lambda'^t \neq 0$, then for some $j$, $\theta_j$ is strict and $\lambda'_j \neq 0$ and hence $\lambda'^t \neq 0$ and thus, $\tilde{\lambda}_j$ is strict. So, we have that $\lambda A + \mu B = 0$ and one of the following is satisfied: $\lambda a + \mu b \leq -1$; or $\lambda a + \mu b \leq 0$ and $\lambda^t \neq 0$; or $\lambda a + \mu b \leq 0$ and $\lambda^t \neq 0$. As $\bigwedge_{k \in \mathbb{I} \cup \mathbb{J}} \lambda_j A_{ik} = 0$, we have that $\bigwedge_{k \in \mathbb{I} \cup \mathbb{J}} \tilde{\lambda}_j A_{ik} = 0$. Thus, $\lambda, \mu, \tilde{\lambda}_0, \ldots, \tilde{\lambda}_n$ satisfy the constraints and therefore, algorithm LILA will find an interpolant with the desired properties.

5.3 Refinement Loop

In each iteration of the refinement loop, an abstract perfect-information game is solved. If the winner in this game is the system player, then the algorithm terminates returning an abstract winning strategy for the system. Otherwise the constructed abstract winning strategy $f_e$ for the environment is checked for concretizability. If $f_e$ is genuine, then the algorithm terminates returning this abstract strategy. If $f_e$ is spurious, then the abstraction is refined with the predicates $R(f_e)$, computed as we explain below.

<table>
<thead>
<tr>
<th>Algorithm: ARGII</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> symbolic safety game $G = (S, err)$</td>
</tr>
<tr>
<td><strong>Output:</strong> pair (winner, abstract strategy)</td>
</tr>
<tr>
<td>$\mathcal{P} := (\text{Preds(init)} \cup \text{Preds(err)} \cup {t \approx 0}, \emptyset)$;</td>
</tr>
<tr>
<td>solve $\alpha(G, \mathcal{P})$ and determine: winner and strategy;</td>
</tr>
<tr>
<td>while winner = env do</td>
</tr>
<tr>
<td>if $F($strategy$)$ is satisfiable then return (winner, strategy);</td>
</tr>
<tr>
<td>$f_e :=$ strategy;</td>
</tr>
<tr>
<td>$R := (\emptyset, \emptyset)$;</td>
</tr>
<tr>
<td>if there is a $\tau \in \text{Traces}(f_e)$ such that $F(f_e, \tau)$ is unsatisfiable then</td>
</tr>
<tr>
<td>$R := \text{RPGG}(T(f_e), \tau)$;</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>$R := \text{RPO}(T(f_e))$;</td>
</tr>
<tr>
<td>compute $S := \text{Refine}(f_e, R)$;</td>
</tr>
<tr>
<td>forall $f'_e \in S$ do</td>
</tr>
<tr>
<td>find $\tau' \in \text{Traces}(f'_e)$ such that $F(f'_e, \tau')$ is unsatisfiable;</td>
</tr>
<tr>
<td>$R := R \cup \text{RPGG}(T(f'_e), \tau')$;</td>
</tr>
<tr>
<td>$\mathcal{P} := \mathcal{P} \cup R$;</td>
</tr>
<tr>
<td>solve $\alpha(G, \mathcal{P})$ and determine winner and strategy;</td>
</tr>
<tr>
<td>return (winner, strategy)</td>
</tr>
</tbody>
</table>

At the refinement step we distinguish between the following two cases. If there exists an unsatisfiable trace formula for $f_e$, then it suffices to refine the abstract transition relations in order
to exclude the spurious strategy $f_e$. Therefore in this case we apply algorithm RPGG to compute the refinement predicates. If all trace formulas are satisfiable, we apply algorithm RPOE. However, in this case it is possible that in the game $\alpha(G, P \cup RPOE(f_e))$, the environment has a winning strategy $f'_e$ with $f'_e \leq f_e$. If this is the case, then we also refine with the predicates in RPGG($f'_e$) for every such $f'_e$. The set Refine($f_e$, RPOE($f_e$)) consists of all winning strategies for the environment in $\alpha(G, P \cup RPOE(f_e))$ that are subsumed by $f_e$. It is computed from the strategy $f_e$ and the predicates in $P \cup RPOE(f_e)$.

**Theorem 4 (Soundness of algorithm ARGII).** The algorithm ARGII is sound: if it returns $(\text{sys}, f^a_e)$, then the winner of the concrete symbolic game $(S, err)$ is the system and $f^a_e$ is a concretizable abstract winning strategy for the system; if it returns $(\text{env}, f^a_e)$ then the winner in $(S, err)$ is the environment and $f^a_e$ is a concretizable abstract winning strategy for the environment.

**Proof.** The theorem follows from the soundness of the abstraction and the construction of strategy-tree formula. \qed

### 5.4 Progress of the Refinement

In the following lemma we formalize and prove the progress property of algorithm RPGG, namely that if some of the trace formulas for a spurious winning environment strategy is unsatisfiable, then RPGG computes predicates that suffice to exclude the spurious counterexample.

**Lemma 1.** Let $f_e$ be a spurious winning strategy for the environment in the game $\alpha(G, P)$. If there exists a trace $\tau \in \text{Traces}(f_e)$ such that $F(f_e, \tau)$ is unsatisfiable, then in the abstract game $\alpha(G, P')$, where $P' \supseteq \text{RPGG}(f_e)$, there does not exist a winning strategy $f'_e$ for the environment with $f'_e \leq f_e$.

**Proof.** Assume that $f'_e$ is a winning strategy for the environment in the game $\alpha(G, P')$ and $f'_e \leq f_e$. Consider the trace $\tau_0 \in \text{Traces}(f_e)$ for which $F(f_e, \tau_0)$ is unsatisfiable and $\text{OutPreds}(\tau_0) \subseteq \text{RPGG}(f_e)$.

Since $P' \supseteq \text{RPGG}(f_e)$ and $f'_e \leq f_e$ is a winning strategy for the environment, there exists a node $\pi' \in T(f'_e)$ such that $\pi'[j_0]\pi'[j_1] \ldots \pi'[j_m]$ is the subsequence of $\pi'$ that consists of all system states in $\pi'$, $m + 1 \leq |\tau_0|$, for every $0 \leq k \leq m$, if $j_k + 1 < |\pi'|$ then $\pi'[j_k + 1] \models x_o \approx \tau_0[k]$ and either $last(\pi')$ is an error state or $last(\pi') = \pi'[j_m]$ and $\pi'[j_m]$ does not have a successor satisfying $x_o \approx \tau_0[m]$.

Since $P' \supseteq \text{RPGG}(f_e)$, there exists a path $\pi^e \in \gamma(\pi')$ such that $last(\pi^e)$ is an error state or $last(\pi^e) \models \neg \text{Enabled}(\tau_0[m])$. Therefore, $\pi^e \models F(f'_e, \tau_0)$. As $F(f'_e, \tau_0)$ implies $F(f_e, \tau_0)$, we have that $\pi^e \models F(f_e, \tau_0)$, which contradicts the choice of $\tau_0$. \qed

In order to show the progress of the refinement for the case when all trace formulas are satisfiable, we first show that after refining the abstract game with the predicates computed by the algorithm RPOE, we obtain an intermediate abstraction in which every winning environment strategy subsumed by the strategy that guides the refinement has an unsatisfiable trace formula.

The results in the rest of this section hold in the case when the calls to procedure LILA return result different form $\perp$.

Consider now an execution of algorithm RPOE applied to some spurious strategy $f_e$ for which all trace formulas are satisfiable. At the first iteration the set $\Phi$ is the set of all trace formulas and the set $\Psi \cup \{\varphi\}$ is chosen to be a subset of $\Phi$. For such an execution of RPOE, RefTraces($f_e$) is a set of traces that depends on $\Psi$ and $\varphi$ and is defined as follows: $\text{RefTraces}(f_e) = \{\tau | F(f_e, \tau) \in \Psi \cup \{\varphi\}\}$.

Let $\tau$ be a trace in $\text{Traces}(f_e)$ for some abstract winning environment strategy $f_e$ and $\pi$ be a prefix in the concrete game $G$ such that $\pi \models F(f_e, \tau)$. If the number of system states in $\pi$ is greater than the length of $\tau$, then there exists a prefix $\pi'$ of $\pi$ such that $\pi' \models F(f_e, \tau)$ and the number of system states in $\pi'$ is less than or equal to $|\tau|$. Therefore, whenever $\pi \models F(f_e, \tau)$ we assume w.l.o.g. that the number of system states in $\pi$ is less than or equal to $|\tau|$.

Let $\pi$ be a prefix in the concrete game $G$, $\rho = \pi[j_0] \ldots \pi[j_m]$ be the subsequence of $\pi$ that consists of all system states in $\pi$ and $\tau$ be a trace with $|\tau| \geq m + 1$, such that for every $0 \leq k \leq m$,
if $j_k + 1 < |\pi|$, then $\pi[j_k + 1]$ is a $\tau[k]$-successor of $\pi[j_k]$. Then, for every $0 \leq k < |\pi|$ we define $\text{pos}(\pi, \tau, k)$ to be the index $j_k$ if $k \leq m$ and to be $m$ otherwise. Thus, $\text{pos}(\pi, \tau, k)$ is the position on $\pi$ right before the output $\tau[k]$ if such position exists and the position of the last state system in $\pi$ otherwise.

For two different traces $\tau_1$ and $\tau_2$, $\text{Diff}(\tau_1, \tau_2)$ is the smallest $j$ such that $\tau_1[j] \neq \tau_2[j]$ if $|\tau_1| = |\tau_2|$, and is $\min(|\tau_1| - 1, |\tau_2| - 1)$ otherwise.

**Lemma 2.** Let $\mathcal{P}$ be a pair of finite sets of predicates and $f_e$ be a spurious winning strategy for the environment in the game $\alpha(\mathcal{G}, \mathcal{P})$ such that for every $\tau \in \text{Traces}(f_e)$, $F(f_e, \tau)$ is satisfiable. Let $\text{RefTraces}(f_e)$ be the set of traces that corresponds to the application of algorithm $\text{RPOE}$ to $f_e$ where all calls to the procedure LILA returned result different from $\bot$. Then, if $\Xi \subseteq \text{Pref}(\mathcal{G})$ is a set of concrete prefixes such that for every $\tau \in \text{RefTraces}(f_e)$, there exists $\pi^c_\tau \in \Xi$ with $\pi^c_\tau \models F(f_e, \tau)$, then there exist:

- $\tau_1, \tau_2 \in \text{RefTraces}(f_e)$ such that $\tau_1 \neq \tau_2$ and $\text{Diff}(\tau_1, \tau_2) = n$ for some $n$,
- a predicate $p \in \text{Obs}(\text{RPOE}_e(f_e))$ and
- a position $j \leq \min(\text{pos}(\pi^c_\tau_1, \tau_1, n), \text{pos}(\pi^c_\tau_2, \tau_2, n))$,

such that $\pi^c_\tau[j] \models p$ iff $\pi^c_\tau_2[j] \models \neg p$.

**Proof.** Consider the execution of algorithm $\text{RPOE}$ applied to the strategy $f_e$. At each iteration of the loop, the algorithm computes an interpolant for a pair of formulas $\varphi$ and $\psi$, where $\psi$ is the conjunction of the elements of a set $\Psi$ of formulas. We are going to prove the following statement. If at the current iteration $\varphi$ and $\Psi$ are such that for every $\varphi \in \Psi \cup \{\varphi\}$ there exists a trace $\tau \in \text{RefTraces}(f_e)$ such that $\pi^c_\tau \models \varphi$ and for every pair of different formulas, the corresponding traces are different, then there exist traces $\tau_1, \tau_2 \in \text{RefTraces}(f_e)$, a predicate $p \in \text{RPOE}_e(f_e)$ and a position $j$ with the desired properties. This will complete the proof, since the precondition of the statement holds for the initial set $\Psi$ and formula $\varphi$ as $\Psi \subseteq \Phi$ and $\varphi \in \Phi$.

The proof is by induction on the size of $\Psi$. In the base case $\Psi = \{\psi\}$, i.e., $\Psi$ is a singleton set. Let $\tau_1 \in \text{RefTraces}(f_e)$ be a trace such that $\pi^c_\tau \models \varphi$ and $\tau_2 \in \text{RefTraces}(f_e)$ be a trace such that $\pi^c_\tau \models \psi$ and $\tau_1 \neq \tau_2$. Let $\theta$ be the interpolant computed for $\varphi$ and $\psi$ by algorithm $\text{RPOE}$. Then, $\pi^c_\tau \models \theta$ and $\pi^c_\tau \models \neg \theta$. We have that the renamed versions of all atomic formulas that occur in $\theta$ are in $\text{Obs}(\text{RPOE}_e(f_e))$. Therefore, there exists a predicate $p \in \text{Obs}(\text{RPOE}_e(f_e))$ and a position $j \leq \min(\text{pos}(\pi^c_\tau_1, \tau_1, \text{Diff}(\tau_1, \tau_2)), \text{pos}(\pi^c_\tau_2, \tau_2, \text{Diff}(\tau_1, \tau_2)))$ such that $\pi^c_\tau[j] \models p$ iff $\pi^c_\tau_2[j] \models \neg p$.

Now, consider the case when $|\Psi| > 1$. Let $\psi = \bigwedge_{\varphi \in \Psi} \varphi$ and $\theta$ be the interpolant for $\varphi$ and $\psi$. Let $\tau_1 \in \text{RefTraces}(f_e)$ be a trace such that $\pi^c_\tau \models \varphi$. There are two possibilities. If there exists a trace $\tau_2 \in \text{RefTraces}(f_e)$ such that $\pi^c_\tau \models \neg \theta$ then $\tau_2 \neq \tau_1$ and the proof is analogous to the one in the base case. The other possibility is that for every $\tau \in \text{RefTraces}(f_e)$, $\pi^c_\tau \models \theta$. Therefore, for each $\phi \in \Psi$, the formula $\phi \land \theta$ is satisfiable and there exists a trace $\tau \in \text{RefTraces}(f_e)$ with $\pi^c_\tau \models \phi \land \theta$. As $\theta$ is an interpolant for $\varphi$ and $\psi$, the conjunction $\bigwedge_{\varphi \in \Psi} \phi \land \theta$ is unsatisfiable. Let $\Phi' = \{\phi \land \theta \mid \phi \in \Psi\}$. The set $\Phi'$ is nonempty and each of its elements is a satisfiable formula. Therefore, for some nonempty subset $\Psi' \subseteq \Phi'$ and some $\varphi' \in \Phi'$ such that the conjunction of the elements of $\Psi'$ is satisfiable and the conjunction of that conjunction and the formula $\varphi'$ is unsatisfiable, the algorithm has computed an interpolant. Since $1 \leq |\Phi'| < |\Psi|$, we can apply the induction hypothesis which yields the desired property. Thus, the proof is completed. □

**Lemma 3.** Let $\mathcal{P}$ and $\mathcal{P}'$ be pairs of finite sets of predicates and $f_e$ and $f'_e$ be spurious winning strategies for the environment in the games $\alpha(\mathcal{G}, \mathcal{P})$ and $\alpha(\mathcal{G}, \mathcal{P}')$, respectively, such that $f'_e \leq f_e$ and $\mathcal{P}' \supseteq \mathcal{P} \cup \text{RPOE}(f_e)$. If all calls to the procedure LILA in $\text{RPOE}$ have returned result different from $\bot$, then there exists a trace $\tau \in \text{Traces}(f'_e)$ such that the trace-formula $F(f'_e, \tau)$ is unsatisfiable.

**Proof.** Consider the application of algorithm $\text{RPOE}$ to the strategy $f_e$ and the corresponding set of traces $\text{RefTraces}(f_e) \subseteq \text{Traces}(f_e)$. Since $f'_e$ is a winning strategy in $\alpha(\mathcal{G}, \mathcal{P}')$ such that $f'_e \leq f_e$, it holds that $\text{Traces}(f_e) \subseteq \text{Traces}(f'_e)$. Therefore $\text{RefTraces}(f_e) \subseteq \text{Traces}(f'_e)$. Thus, if for some $\tau \in \text{RefTraces}(f_e)$ there is no node $\pi_\tau$ in $T(f'_e)$ such that there is a $\pi^c_\tau \in \gamma(\pi_\tau)$ with $\pi^c_\tau \models F(f'_e, \tau)$, the property is satisfied.
Assume that for every trace \( \tau \in \text{RefTraces}(f_c) \) there exist a node \( \pi_\tau \in T(f'_c) \) and a path \( \pi_\tau^c \in \gamma(\pi_\tau) \) such that \( \pi_\tau^c \models F(f'_c, \tau) \). Since \( f'_c \leq f_c \), the formula \( F(f'_c, \tau) \) implies \( F(f_c, \tau) \) for every \( \tau \in \text{Traces}(f_c) \). Therefore, according to Lemma 2, there exist traces \( \tau_1 \) and \( \tau_2 \) in \( \text{RefTraces}(f_c) \) with \( \tau_1 \neq \tau_2 \), a predicate \( p \in \text{Obs}(P') \) and a position \( j \leq \min(\text{pos}(\pi_\tau^c, \tau_1, n), \text{pos}(\pi_\tau^c, \tau_2, n)) \), where \( n = \text{Diff}(\tau_1, \tau_2) \), such that \( \pi_\tau^c[j] \models p \) iff \( \pi_\tau^c[j] \models \lnot p \).

Since \( \pi_\tau_1 \) and \( \pi_\tau_2 \) are both nodes in \( T(f'_c) \), \( \text{OutPreds}(\{\tau_1, \tau_2\}) \subseteq P' \). \( \pi_\tau_1 \in \gamma(\pi_\tau_1) \) and \( \pi_\tau_2 \in \gamma(\pi_\tau_2) \), for every \( k \leq \min(\text{pos}(\pi_\tau_1, \tau_1, n), \text{pos}(\pi_\tau_2, \tau_2, n)) \), it holds that \( \pi_\tau_1[k] = \pi_\tau_2[k] \). This contradicts to the fact that we showed above, namely, that there is a predicate \( p \in \text{Obs}(P') \) and a position \( j \leq \min(\text{pos}(\pi_\tau_1, \tau_1, n), \text{pos}(\pi_\tau_2, \tau_2, n)) \) such that \( \pi_\tau_1[j] \models p \) iff \( \pi_\tau_2[j] \models \lnot p \). Thus, this case is not possible and the proof is completed.

**Theorem 5** (Progress property of the refinement). Let \( f_c \) be a spurious winning strategy for the environment in the game \( \alpha((C, err), P) \). If all calls to the procedure LILA have returned result different from \( \perp \), then in \( \alpha((C, err), P \cup R(f_c)) \), the environment does not have a winning strategy \( f'_c \) with \( f'_c \leq f_c \).

**Proof.** The theorem is a direct consequence of Lemma 3 and Lemma 1.

**6 Termination of the Abstraction Refinement Loop**

In this section we provide sufficient conditions for termination of the refinement loop. In order to guarantee that only finitely many different abstract states are generated during the execution of the algorithm, we make standard assumptions about the concrete game graph, which we extend with conditions related to the presence of incomplete information.

As we have for the refinement predicates obtained from interpolants, we apply the standard technique (e.g., [7]) of restricting the interpolants computed at each step to some finite language \( L_b \) and maintaining completeness by gradually enlarging the restriction language when this is needed. We make use of the fact that our algorithm reduces interpolant computation to constraint solving and achieve the restriction of the language by imposing additional constraints on the generated inequalities.

**Computing restricted linear interpolants.** We restrict the language of the computed interpolants to the set of rectangular predicates over the variables in \( \text{Obs}(X) \). A rectangular predicate over \( \text{Obs}(X) \) is a conjunction of rectangular inequalities of the form \( ax < c \), where \( x \in \text{Obs}(X) \), \( a \in \{-1, 1\} \), \( c \in \{<, \leq\} \) and \( c \) is an integer constant. For \( m \in \mathbb{N} \), a rectangular predicate \( \varphi \) is called \( m \)-bounded if for each conjunct \( ax < c \) of \( \varphi \), \( |c| \leq m \). Let \( L_m \) be the set of all \( m \)-bounded rectangular predicates over \( \text{Obs}(X) \).

The algorithm LILA, a modification of LILA, gets as input also a bound \( b \in \mathbb{N} \) and ensures that every conjunct in the computed interpolant is in \( L_b \). If such an interpolant does not exist, then the algorithm LILA is called with increased bound \( b + 1 \). The modified algorithm partitions the variables into singleton sets and uses in conjunction with \( \chi_1 \) and \( \chi_2 \) the additional constraints:

1. \( \chi_3 \) defined as \( \chi_3 = \bigwedge_{i=0}^n (i_j \leq 1 \land i_j \geq -1 \land \delta_j \leq b \land \delta_j \geq -b) \)
2. \( \delta_j \) and the variables \( i_j \) in the vector \( i_j \) assume integer values.

**Region algebra for an incomplete-information game.** A region algebra for a symbolic safety game \( G = (C, err) \) is a pair \( (R, \text{Obs}) \), where \( R \subseteq 2^S \) is a possibly infinite set of regions such that \( \bigcup R = S \) and \( \text{Obs} \subseteq R \) is a possibly infinite subset of observable refions, with the following properties:

1. The sets \( R \) and \( \text{Obs} \) are closed under boolean operations:
   - for every \( \tau_1, \tau_2 \in R \), the sets \( \tau_1 \cup \tau_2, \tau_1 \cap \tau_2 \), and \( S \setminus \tau_1 \) are in \( R \),
   - for every \( \tau_1, \tau_2 \in \text{Obs} \), the sets \( \tau_1 \cup \tau_2, \tau_1 \cap \tau_2 \) and \( S \setminus \tau_1 \) are in \( \text{Obs} \);
2. The sets \( \{v \in S \mid v \models t \approx 0\} \) and \( \{v \in S \mid v \models t \approx 1\} \) are in \( \text{Obs} \);
3. For every \( c_o \in C_o \), the set \( \{v \in S \mid v \models x_o \approx c_o\} \) is in \( \text{Obs} \);
(4) Every region is expressible over $\mathcal{AP}$ and the predicate transformers $Pre_s$ and $Pre_e$ are defined in such a way that for every $r \in R$, $c_0 \in C_0$, and predicate $p$ with $p \in Pre_s(Pre_e(r))$ or $p \in Pre_e(Pre_s(c_0, r))$, it holds that for every $r' \in R$, the sets $r' \cap \{v \in S \mid v \models p\}$ and $r' \cap \{v \in S \mid v \models \neg p\}$ are in $R$;

(5) For every $\pi_1, \pi_2 \in Prefs(G)$, if each of last($\pi_1$) and last($\pi_2$) is a dead-end or error state and there exists an index $j$ such that $\pi_1[j]$ and $\pi_2[j]$ are system states and $\pi_1[j] \not\equiv \pi_2[j]$, then there exist an index $0 \leq k \leq j$ and an observable region $r \in \text{Obs}$ such that $\pi_1[k] \in r$ and $\pi_2[k] \not\in r$.

**Theorem 6 (Termination).** Let $(\mathcal{C}, \text{err})$ be a symbolic safety game with incomplete information for which there exists a finite region algebra $(R, \text{Obs})$ with $\text{Obs} = L_m$ for some $m \in \mathbb{N}$. If algorithm ARGII using the modified algorithm LILA is called with argument $(\mathcal{C}, \text{err})$, then it terminates.

**Proof.** Since each $L_b$ is finite, after finitely many steps, if the algorithm has not terminated, we have that the current bound $b$ is equal to $m$. From that iteration on, when algorithm LILA has to compute an interpolant for some $\varphi$ and $\Psi$, then an interpolant over $L_b$ exists and will be found by the algorithm. Therefore, there exists an iteration, from which on, the bound $b \leq m$ is not increased. Hence, all abstract states produced by algorithm ARGII during its execution are elements of $R$. According to the progress property, if the algorithm does not terminate, infinitely many spurious counterexamples are ruled out, which means that infinitely many times an abstract state is split. This is not possible, since each abstract state is an element of $R$ and $R$ is finite. Therefore, in this case algorithm ARGII terminates.

7 Conclusions

We have proposed a counterexample-guided abstraction-refinement approach for solving incomplete-information games between a system and its environment. We believe that counter-strategy guided refinement is the right technique to use for abstraction refinement in the incomplete-information case. That is because the reason for spuriousness cannot in general be localized at a single abstract state, i.e., one has to consider the prefix of a play that leads to that state in order to sufficiently refine the abstract observation-equivalence. Our main contribution is the novel refinement method that accounts for the overapproximation of the observation-equivalence and uses interpolation to derive suitable predicates to refine it. Our approach is applicable to important classes of infinite-state models such as timed games or games defined by bounded rectangular automata.

References


