Refining Abstractions of Hybrid Systems Using Counterexample Fragments

Ansgar Fehnker, Edmund Clarke, Sumit Jha, Bruce Krogh

Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213 (USA)

Abstract. Counterexample guided abstraction refinement, a powerful technique for verifying properties of discrete-state systems [4, 9] has been extended recently to hybrid systems verification [1, 3]. Unlike in discrete systems, however, establishing the successor relation for hybrid systems can be a fairly expensive step since it requires evaluation and overapproximation of the continuous dynamics. In [3] it has been observed that it is often sufficient to consider fragments of counterexamples rather than complete counterexamples. In this paper we further develop the idea of fragments. We extend the notion of cut sets in network flows to cutting sets of fragments in abstractions. Cutting sets of fragments are then uses guide the abstraction refinement in order to prove safety properties for hybrid systems.

1 Introduction

Model checking for hybrid systems requires finite abstractions [1–3, 13]. Since abstractions are conservative representations of the hybrid system dynamics, only positive verification results are conclusive: a universal specification that is true for the abstraction is also true for the hybrid system. However, when the specification is not true for the abstraction it could still be true for the hybrid system because there may be no behaviors for the hybrid system that correspond to the behaviors of the abstraction that violate the specification. When model checking fails for the abstraction of a hybrid system, the abstraction can be *refined* to create a less conservative approximation to the hybrid system and model checking is repeated on the new abstraction. This paper presents a new method for constructing refinements of abstractions for hybrid systems. The method is developed for safety specifications, that is, for specifications that can be expressed in terms of reachability conditions.

Abstractions of hybrid systems are usually quotient transition systems for the infinite-state transition system that provides the semantics for the hybrid system. The two principal issues in constructing these quotient transition systems are: (i) identifying and representing the sets of hybrid system states that comprise the states for the abstraction; and (ii) computing the transition relation for the abstraction. Step (ii) is usually the most difficult and time-consuming step because it involves the computation of reachable sets for the continuous dynamics in the hybrid system. Moreover, the time involved in computing reachable sets for the continuous dynamics makes the time required to perform model checking on the abstraction negligible in the overall time required to perform the verify-refine iteration described above. The extensive time required to construct a

refinement of an abstraction makes it desirable to find ways to construct effective refinements.

Counterexample guided abstraction refinement (CEGAR) has been proposed to guide the refinement process. This refinement strategy, originally developed for discrete-state systems, uses counterexamples in the abstraction (runs that violate the specification) to determine how to refine the abstraction so that known counterexamples are eliminated [4, 9]. This approach was successfully extended to deal with hybrid systems [1, 3] as follows. For a given counterexample in the abstraction, which is a sequence of sets of hybrid states, the reachable states are computed for the hybrid system starting from the initial set of hybrid states in the counter example (which is a subset of the initial states for the hybrid system). This process is called *validating* the counterexample. If there is a set of states along the counterexample that cannot actually be reached in the hybrid system, the counterexample is *refuted* and is to be a *spurious* counterexample in the abstraction. This provides the guidance for refining the abstraction: the sets of reachable states computed along the counterexample are introduced as new states in a new abstraction for subsequent model checking. For the hybrid system case (in contrast to the discrete state case), not being able to refute a counterexample using reachability computations does not guarantee there exists a trajectory for the hybrid system that violates the specification. When the counterexample is not refuted, one might explore for possible validating counterexamples in the hybrid system using simulation or reachability computations with *inner approximations*. Alternatively, the information from the overapproximated reachability computations can be used to construct a refined abstraction.

We made two important observations in our work on CEGAR for hybrid systems. First, rather than refuting a complete counterexample, it is sufficient and often a lot cheaper to refute a *fragment* of the counterexample. Second, coarse overapproximation methods to compute reachable sets for hybrid systems are not only computationally faster, but can also lead to smaller refinements that lead to conclusive results more quickly than those obtained using more exact (but computationally expensive) methods. These observations are the basis for the new approach to abstraction refinement proposed in this paper. The overall goal is to obtain as much information as possible from an analysis of the structure of the graph representing *all* counterexamples for an abstraction, and to use this information to minimize the amount of time devoted to performing expensive reachability computations for the underlying hybrid system dynamics.

In this new procedure, rather than validating a single counterexample, the reachable set computations aim to validate a *cutting set* of fragments for the graph of counterexamples for a given abstraction. We introduce the concept of the cutting set of fragments as an extension of the standard notion of a cut set of links for a network graph. A weight assigned to each fragment identifies the expected cost for validating the fragment. An optimal cutting set of fragments is computed based on these weights to minimize the time devoted to hybrid system reachability computations. The new scheme combines ideas from computing cut sets for scenario graphs and network flows [10] with concepts taken from counterexample guided abstraction refinement [3].

The next section introduces transition systems, abstractions and the concept of cutting sets of fragments for abstractions. Section 3 illustrates the purpose of considering fragments of counterexamples and how fragments can be validated using methods for computing reachable sets for hybrid systems. The proposed procedure for hybrid system verification using fragments to guide abstraction refinement is then presented in Sect. 4. Section 5 explains how to compute minimal cutting sets of fragments based on an extension of standard cut set algorithms for network graphs. The procedure is illustrated with an example in Sect. 6, and the paper concludes a discussion of future work in Sect. 7.

2 Preliminaries

This section introduces the basic structures and concepts used in the proposed approach for refinement of abstractions for hybrid systems using fragments of counterexamples.

Hybrid systems are a class of infinite-state systems that include both continuous and discrete state variables. The standard model for hybrid systems is the hybrid automaton.

Definition 1. *A hybrid automaton is a tuple* $HA = (Z, z_0, z_f, X, inv, X_0, T, g, j, f)$ *where*

- Z *is a finite set of* locations *with* initial location $z_0 \in Z$, *and* final location z_f.
- $X \subseteq \mathbb{R}^n$ *is the continuous state space.*
- $inv : Z \to 2^X$ *assigns to each location* $z \in Z$ *an invariant* $inv(z) \subseteq X$.
- $X_0 \subseteq X$ *is the set of initial continuous states.*
- $T \subseteq Z \times Z$ *is the set of* discrete transitions *between locations.*
- $g : T \to 2^X$ *assigns a* guard set $g((z_1, z_2)) \subseteq X$ *to* $(z_1, z_2) \in T$.
- $j : T \to (X \to X)$ *assigns to each* $(z_1, z_2) \in T$ *and a reset or* jump *mapping from* X *to* X. *The notation* $j_{(z_1, z_2)}$ *is used for* $j((z_1, z_2))$
- $f : Z \to (X \to \mathbb{R}^n)$ *assigns to each location* $z \in Z$ *a continuous vector field* $f(z)$. *The notation* f_z *is used for* $f(z)$. *The evolution of the continuous behavior in location z is governed by the differential equation* $\dot{\chi}(t) = f_z(\chi(t))$. *The differential equation is assumed to have a unique solution for each initial value* $\chi(0) \in inv(z)$.

We use the standard transition-system semantics for the hybrid automaton.

Definition 2. *A transition system* TS *is a tuple* (S, S^0, S^f, R) *with a set of states S, a set of initial states $S^0 \subseteq S$, a set of accepting states $S^f \subseteq S$, and a transition relation* $R \subseteq S \times S$.

Definition 3. *The semantics of a hybrid automaton HA is a transition system* $TS(HA) = (\bar{S}, \bar{S}^0, \bar{S}^f, \bar{R})$ *with:*

- *the set of all* hybrid states $\bar{S} = \{(z, x) | z \in Z, x \in X, x \in inv(z)\}$,
- *the set of* initial hybrid states $\bar{S}^0 = \{z_0\} \times (X_0 \cap inv(z_0))$,
- *the set of* accepting hybrid states $\bar{S}^f = \{z_f\} \times inv(z_f)$
- *transitions $\bar{R}$ with* $((z_1, x_1), (z_2, x_2)) \in \bar{R}$, *iff* $(z_1, z_2) \in T$ *and there exist a trajectory* $\chi : [0, \tau] \to X$ *for some* $\tau \in \mathbb{R}^{>0}$ *such that:* $\chi(0) = x_1$, $\chi(\tau) \in g((z_1, z_2))$, $x_2 = j_{(z_1, z_2)}(\chi(\tau))$, *and* $\dot{\chi}(t) = f_{z_1}(\chi(t))$ *for* $t \in [0, \tau]$, $\chi(t) \in inv(z_1)$ *for* $t \in [0, \tau]$.

The first step in model checking hybrid systems is to find a suitable finite abstraction, where the notion of abstraction for transition systems is defined as follows.

Definition 4. *Given a transition system $C = (\bar{S}, \bar{S}^0, \bar{S}^f, \bar{R})$, a transition system $A = (S, S^0, S^f, R)$ is an abstraction of transition system C, denoted by $A \succeq C$, if there exist an abstraction function $\alpha : \bar{S} \to S$ such that $S^0 = \alpha(\bar{S}^0)$, $S^f = \alpha(\bar{S}^f)$ (where α is extended to subsets of $\bar{S}$ in the usual way),and*
$$R \supseteq \{(s_1, s_2) | \exists (\bar{s}_1, \bar{s}_2) \in E, \alpha(\bar{s}_1) = s_1, \alpha(\bar{s}_2) = s_2\}.$$

In this paper, we are interested in constructing finite abstractions for $C = TS(HA)$, where HA is a given hybrid automaton. This given infinite-state transition system is referred to as the *concrete system*. Note that the definition of abstraction above allows A to include transitions that have no counterpart in C. Such *spurious transitions* may arise in abstractions of hybrid systems because sets of reachable states for hybrid systems cannot, except for simple dynamics [8], be computed exactly, but have to be overapproximated. The computations of sets of reachable states required for our procedure are represented formally as follows.

For an abstraction function α, let $\overline{\mathcal{S}}_\alpha$ denote the partition of the set of hybrid states $\bar{S}$ defined by the inverse mapping α^{-1}. Our procedure requires a method for computing the set of states that can be reached from one element of $\overline{\mathcal{S}}_\alpha$ in another element of $\overline{\mathcal{S}}_\alpha$. That is, given two sets of hybrid states, $\bar{S}_1, \bar{S}_2$ in $\overline{\mathcal{S}}_\alpha$, we require a method for computing a subset of states in $\bar{S}_2$ that contains the set of hybrid states that can be reached from states in $\bar{S}_1$. We denote such a method by $\overline{succ}$. Given a set of hybrid states $\bar{S}_1 \subset \bar{S}$ the set of *successor states* is denoted by $succ(\bar{S}_1) = \{\bar{s}' | \exists \bar{s} \in \bar{S}_1 . (\bar{s}, \bar{s}') \in \bar{R}\}$. With this notation, an overapproximation method $\overline{succ}$ is defined as:

Definition 5. *Let HA be a hybrid automaton with $TS(HA) = (\bar{S}, \bar{S}^0, \bar{S}^f, \bar{R})$, and let $A = (S, S^0, S^f, R)$ and α as in Defn. 4. Let $\bar{S}_1 = \alpha^{-1}(s_1)$, and $\bar{S}_2 = \alpha^{-1}(s_2)$. Then $\overline{succ} : \overline{\mathcal{S}}_\alpha \to 2^{\bar{S}}$ is an overapproximation of the set of hybrid successors of $\bar{S}_1$ in $\bar{S}_2$ iff $\overline{succ}(\bar{S}_1, \bar{S}_2) \subseteq \bar{S}_2$ and $\overline{succ}(\bar{S}_1, \bar{S}_2) \supseteq succ(\bar{S}_1) \cap \bar{S}_2$.*

Our abstraction refinement procedure provides a framework to use the fact that different overapproximation techniques have different computational loads and accuracy. It was observed in [3] that combinations of coarse and precise methods can improve the efficiency of the verify-refine iterations significantly. In the following we assume a series of overapproximation methods $\overline{succ}_1, \ldots, \overline{succ}_n$ is given that provides a hierarchy of coarse to tight approximations. This hierarchy will be used to assign weights to fragments that reflect the computational effort required to apply the various overapproximation methods.

Our procedure is based on the analysis of sequences of states in abstractions called *fragments*.

Definition 6. *A fragment of a transition system $TS = (S, S^0, S^f, R)$ is a finite sequence $(s_0, \ldots, s_n)$ such that $(s_{i-1}, s_i) \in R$ for $i = 1, \ldots, n$. A run is a fragment with $s_0 \in S^0$. A state s is reachable if the there exists a run that ends in s. An accepting run is a run $(s_0, \ldots, s_n)$ with $s_n \in S^f$. The set of all accepting runs of TS will be denoted by $\mathcal{R}(TS)$. A run $(s_0, \ldots, s_n)$ is loop-free if for all $i, j \in \{0, \ldots, n\}$, $i \neq j$ implies $s_i \neq s_j$.*

This paper deals with *safety properties*. The set of states S^f should not be reachable, that is, the transition system should not have any accepting run. We refer to S^f as the *set of bad states* and to accepting runs as *counterexamples*. Our analysis of counterexamples for abstractions will focus on sets of fragments, using the following notions of cutting fragments and cutting sets of fragments.

Definition 7. *For $n_2 \geq n_1 \geq 0$, fragment $\varrho_1 = (s_0, \ldots, s_{n_1})$ cuts a fragment $\varrho_2 = (t_0, \ldots, t_{n_2})$, denoted by $\varrho_1 \sqsubseteq \varrho_2$, if there exists a $i \in 0, \ldots, n_2 - n_1$ such that $t_{i+j} = s_j$ for $j = 0, \ldots, n_1$.*

Definition 8. *A set $\mathcal{F}_1$ of fragments cuts a set of fragments $\mathcal{F}_2$, denoted by $\mathcal{F}_1 \sqsubseteq \mathcal{F}_2$, if for each fragment $\varrho_2 \in \mathcal{F}_2$ there exist $\varrho_1 \in \mathcal{F}_1$ such that $\varrho_1 \sqsubseteq \varrho_2$. Set $\mathcal{F}_1$ is minimal if $\mathcal{F}_1 \sqsubseteq \mathcal{F}_2$ and $\mathcal{F}_1 \setminus \varrho \not\sqsubseteq \mathcal{F}_2$ for all $\varrho \in \mathcal{F}_1$. Given a transition system TS, a set of fragments $\mathcal{F}$ cuts TS if $\mathcal{F} \sqsubseteq \mathcal{R}(A)$.*

In words, a fragment ϱ_1 cuts another fragment ϱ_2 if ϱ_1 is a subsequence ϱ_2. When a transition system is an abstraction of a hybrid system, a set of fragments $\mathcal{F}$ that cuts the abstraction covers all counterexamples for the abstraction, that is, any path from the initial state to the bad state (a counterexample) is cut by one of the fragments in $\mathcal{F}$. Any set of fragments that cuts $\mathcal{F}$ also cuts the abstraction. The remainder of the paper shows how the minimal cutting sets of fragments can be used to guide the refinement of abstractions for hybrid systems.

3 Validating Fragments

Abstractions can be represented as network graphs, with states as nodes and transitions as edges. The initial states can be considered as sources and the final states as sinks. A cut set is a set of edges such that all paths from source to sink contain at least one edge in the set. For example, for the graph in Fig. 1.(a) transitions (G, J) and (B, E) are a cut set. All paths from source to sink pass through one of those edges. All accepting runs are cut, if those edges are deleted from the graph.

This paper generalizes the idea of cut sets to sets of fragments that cut the abstraction. For example, fragments (D, H) and (C, G, J) in Fig. 1.(b) form a cut set since all runs from source to sink contain either (D, H) or (C, G, J). If both fragments were spurious, then there would exist no run in the concrete system that connects source to sink. Hence, the concrete system would satisfy the safety property.

The process of determining whether or not a fragment is spurious is called *validating a fragment*. For a given fragment $(s_0, \ldots, s_n)$ of an abstraction A with abstraction function α, the objective is to determine if there exists a fragment $(\bar{s}_0, \ldots, \bar{s}_n)$ of hybrid system C, such that $s_i = \alpha(\bar{s}_i)$, for all $i = 0, \ldots, n$. Computation of hybrid successors is the key step in the validation procedure. The validation procedure use methods $\overline{succ}_1, \ldots, \overline{succ}_m$ for the validation step. The procedure maintains a mapping $\mathcal{X} : (\mathcal{F} \times \mathbb{N}) \hookrightarrow \{1, \ldots, m\}$ that assigns method $\mathcal{X}((s_0, \ldots, s_n), i)$ to validate transition i of fragment $(s_0, \ldots, s_n)$. Initially $\mathcal{X}$ assigns to all transitions of in the initial set $\mathcal{F}$ the computationally cheapest method. $\mathcal{X}$ will be updated in the augmentation and refinement step, along with the set of fragments.
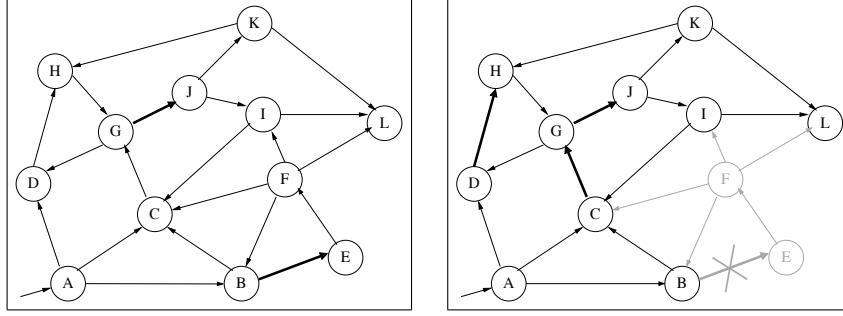
Fig. 1. The initial state of this transition system is A, the accepting state is L. Figure 1.(a) depicts a pair of transitions that cut the transition system. Cutting set can also contain fragments of length greater than 2 (Fig. 1.(b)).

The validation is performed as follows, given abstraction A, concrete system C, abstraction function α, and fragment $(s_0, \ldots, s_n)$:

$$valid := \text{true}, \bar{S}_0 := \alpha^{-1}(s_0)$$
$$\textbf{for } i = 1, \ldots, n-1$$
$$\quad \bar{S}_i := \alpha^{-1}(s_i)$$
$$\quad \bar{S}_i := \overline{succ}_{\mathcal{X}((s_0,\ldots,s_n),i)}(\bar{S}_{i-1}, \bar{S}_i)$$
$$\quad \textbf{if } \bar{S}_i = \emptyset$$
$$\quad\quad valid := \text{false}$$
$$\quad\quad \textbf{break}$$
$$\quad \textbf{end } \% \text{ if}$$
$$\textbf{end } \% \text{ for}$$

This procedure computes the hybrid successors along the fragment. There exist no corresponding run to $(s_0, \ldots, s_n)$ if a set of successors $\bar{S}_i$ becomes empty.

The need to consider fragments of length 2 or longer arises when all single-transition fragments have been validated and some are found to be non-spurious. Suppose for example, that (B, E) in Fig. 1.(a) has been shown to be spurious, while (G, J) has been shown to be non-spurious. The next iteration has to choose a cutting set from the abstraction in Fig. 1.(b). Fragment (G, J) however can not be part of the next cutting set, since it is known to be non-spurious. Suppose that (D, H) and (C, G, J) have not been validated yet. The set of fragments (D, H) and (C, G, J) can then be chosen as next as cutting set, and one must then checked if they are spurious.

4 Using Sets of Fragments for Abstraction Refinement

Figure 2 presents our procedure for model checking hybrid systems using sets of fragments to guide the abstraction refinement. The inputs to the procedure are: C, a given concrete (hybrid) system; A, an initial abstraction for C; $\mathcal{F}$, a set of loop-free fragments that cuts A; and $\mathcal{P} : \mathcal{F} \to \mathbb{N}$, an assignment of weights reflecting the computational effort required to validate each fragment. The concrete system is represented implicitly

through the equations defining the underlying hybrid automaton and the available over-approximation methods. The initial abstraction includes the abstraction function and a representation of the the associated partition of hybrid states. In this paper we assume the initial abstraction $A = (S, S^0, S^f, R)$ is defined as in [3]. This initial abstraction has one abstract state for each control location, with the exception of the initial location. For the initial location the abstraction includes two states, one to represent the set of hybrid states $z_0 \times (inv(z_0) \cap X_0)$, and one state to represent $z_0 \times (inv(z_0) \setminus X_0)$. Given the initial abstraction $A = (S, S^0, S^f, R)$, the initial set of fragments $\mathcal{F}$ is defined to be the set of transitions R. We assume that $\mathcal{P}$ assigns initially the weight associated with applying the computationally cheapest approximation method to a single transition.

In each iteration through the main loop, a new abstraction is constructed based on the results of validating sets of fragments. If there are no accepting runs for the abstraction coming into the main loop ($\mathcal{R}(A) = \emptyset$) the verification terminates with a positive result: the bad state is not reachable in the hybrid system.

The first step in each iteration is to compute a minimal cutting set of fragments $\mathcal{F}_{opt}$ for which the set sum of the weights is minimized (Fig. 2(i)). Section 5 describes the algorithm for finding $\mathcal{F}_{opt}$, which is a generalization of algorithms for finding minimal cut sets of links in a graph.

---

input: $C, A, \mathcal{F}, \mathcal{P}$

while $\mathcal{R}(A) \neq \emptyset$
    $\mathcal{F}_{opt} := \text{cutset}(A, \mathcal{F}, \mathcal{P})$ (i)
    while $\mathcal{F}_{opt} \neq \emptyset \wedge \mathcal{F}_{opt} \subseteq \mathcal{F}$
        $(s_0, \ldots, s_n) :\in \mathcal{F}_{opt}, \mathcal{F}_{opt} := \mathcal{F}_{opt} \setminus (s_0, \ldots, s_n)$
        $valid = \text{validate}((s_0, \ldots, s_n), A, C)$ (ii)
        if $valid \wedge s_0 \in S^0 \wedge s_n \in S^f$
          exit("Found valid accepting run of A")
        elseif $valid$
          $(\mathcal{F}, \mathcal{P}) = \text{augment}(\mathcal{F}, \mathcal{P}, (s_0, \ldots, s_n), A)$ (iii)
          break
        else
          $(A, \mathcal{F}, \mathcal{P}) = \text{refine}(A, \mathcal{F}, \mathcal{P}, (s_0, \ldots, s_n))$ (iv)
        end % if
    end % for
end % while

exit("z_f is not reachable for the HA")

Fig. 2. Abstraction refinement loop that uses cutting sets of fragments $\mathcal{F}_{opt}$ to guide the refinement.

Given the set of fragments $\mathcal{F}_{opt}$, the inner loop iterates through the elements of $\mathcal{F}_{opt}$ one at a time. Each fragment in $\mathcal{F}_{opt}$ will be validated (Fig. 2(ii)). This iteration continues until all fragments have been validated ($\mathcal{F}_{opt} = \emptyset$) or an abstraction has been constructed for which the remaining fragments no longer constitute a subset of the frag-

ments for the abstraction ($\mathcal{F}_{opt} \not\subseteq \mathcal{F}$). If the current fragment is a valid accepting run the procedure stops. Otherwise, if it is a valid fragment the procedure augments the set of fragments and weights assignment (iii), leaves the inner while loop, and recomputes $\mathcal{F}_{opt}$. If the fragment is not valid, the abstraction, fragments and weights are refined (Fig. 2(iv)). This refinement may change $\mathcal{F}$ such that $\mathcal{F}_{opt} \not\subseteq \mathcal{F}$. In this case the procedure exits the inner while loop and recomputes $\mathcal{F}_{opt}$.

To elaborate on the augment and refine functions in the procedure, the validation procedure has two possible outcomes: either the procedure finds an empty set of successors, i.e. there exists no corresponding fragment in C to $(s_0, \ldots, s_n)$, or the procedure could not find an empty set of hybrid successors. The latter may be caused by the overapproximation error of the selected methods. In this case there are two options on how to proceed: Either, the overapproximation can be improved by using a different approximation method, or the current fragment must be replaced by extensions of the current fragment.

Choosing a different overapproximation method. The result of the validation might be improved by a different approximation method in future iterations. Changing the validation methods for fragment $(s_0, \ldots, s_n)$, is done by changing the mapping $\mathcal{X}$ for at least one transition in $(s_0, \ldots, s_n)$. If the procedure changes the mix of methods used to validate $(s_0, \ldots, s_n)$ it has to update function $\mathcal{P}$ accordingly.

Extending the fragment If the overapproximation cannot improve, the current fragment $(s_0, \ldots, s_n)$ will be replaced by new, extended fragments. This becomes necessary if the validation step uses for each fragment the best available overapproximation method. The new fragments will extend $(s_0, \ldots, s_n)$ in both directions of the transition relation, i.e. sets $\{(s', s_0, \ldots, s_n) | (s', s_0) \in R\}$ and $\{(s_0, \ldots, s_n, s') | (s_n, s') \in R\}$ are added to $\mathcal{F}$. Recall the requirement that for all $\varrho_1, \varrho_2 \in \mathcal{F}, \varrho_1 \not\sqsubseteq \varrho_2$. The procedure enforces this requirement by removing all fragments from $\mathcal{F}$ that are cut by some other fragment. Finally, $\mathcal{X}$ and $\mathcal{P}$ are updated for all new fragments of $\mathcal{F}$.

To avoid fragments of unlimited length the augmentation might extend fragments only up to a certain length. First experiments show that an upper bound of 2 to 4 is reasonable. However, adding only a limited number of fragments may lead to a situation in which there are no fragments that cut a certain counterexample. In this case the procedure might add the complete counterexample to the cutting set, and validate it in the next iteration.

Refinement. If the current fragment is not valid, the refinement step (iv) in Fig. 2 uses the sets $\bar{S}_i$ that were computed in the validation step (ii), for $i = 1, \ldots, k$. For $i = 1, ..., k - 1$ the following steps are performed. If $\bar{S}_i$ is a proper subset of $\alpha^{-1}(s_i)$, split s_i into two abstract states, one, s_i^{reach}, to represent the states in $\bar{S}_i$, and one, s_i^{comp} to represent the states in $\alpha^{-1}(s_i) \setminus \bar{S}_i$ (Fig. 3). The new states s_i^{reach} and s_i^{comp} will have the same ingoing and outgoing transitions as S_i, with one exception. The transition from s_{i-1} to s_i^{comp} can be omitted, since there exists no hybrid transition from any state in $\bar{S}_{i-1}$ to some state in $\alpha^{-1}(s_i) \setminus \bar{S}_i$. All fragments from $\mathcal{F}$ that involve state s_i are removed, and the new transitions of the abstraction are added to $\mathcal{F}$. $\mathcal{X}$ assigns to the new
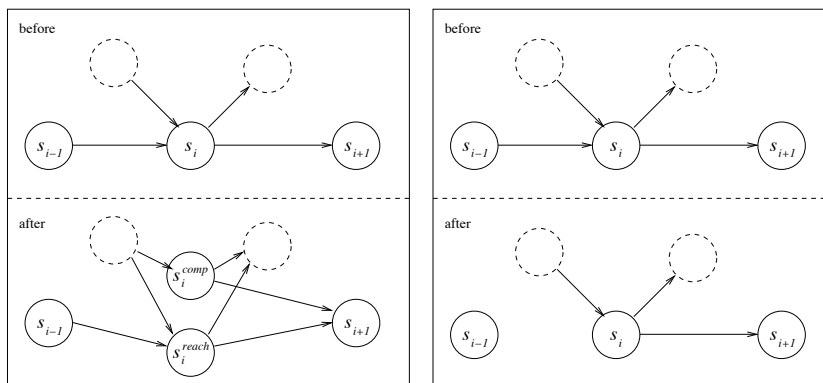
Fig. 3. Left: Refinement by splitting states. Right: Refinement by purging transitions. For a formal definition of the refinement operations see [3].

fragments the default method for single transitions, and $\mathcal{P}$ the weight that is associated with this method. If $\bar{S}_i$ is equal to $\alpha^{-1}(s_i)$, then there is no need to refine the abstraction.

For $i = k$ the transition (s_{k-1}, s_k) is omitted from the abstraction (Fig. 3), since there exists no hybrid transition from any state in $\bar{S}_{k-1}$ to some state in $\alpha^{-1}(s_i)$. Similarly, all fragments from $\mathcal{F}$ that contain transition (s_{k-1}, s_k) are removed.

5 Optimal Cutting Sets of Fragments

This section describes the cutset operation in step (i) of Fig. 2. Assume a finite transition system A (the current abstraction), a set of fragments $\mathcal{F}$ and a weight assignment $\mathcal{P} : \mathcal{F} \rightarrow \mathbb{N}$. The fragments in $\mathcal{F}$ have not been validated and are candidate elements of the optimal cutting set. By assumption for the initial abstraction, and by construction for all subsequent abstractions, all fragments in $\mathcal{F}$ are loop-free. $\mathcal{P}$ assigns to each $\varrho \in \mathcal{F}$ a weight; this weight reflects the expected cost of validating this fragment. The weight of a set $\mathcal{F}' \subseteq \mathcal{F}$ is the sum of the weights of the elements. Furthermore, it is assumed that $\varrho_1 \sqsubseteq \varrho_2$ implies $\mathcal{P}(\varrho_1) \leq \mathcal{P}(\varrho_2)$. As a consequence it is required for all $\varrho_1, \varrho_2 \in \mathcal{F}$ that $\varrho_1 \not\sqsubseteq \varrho_2$, i.e. no fragments in $\mathcal{F}$ cuts another.

Step (i) of the procedure in Fig. 2 computes a cutting set $\mathcal{F}_{opt} \subseteq \mathcal{F}$ of A that is minimal w.r.t. to $\mathcal{P}$, i.e. it satisfies

$$\sum_{f \in \mathcal{F}_{opt}} \mathcal{P}(f) = \min_{\substack{\mathcal{F}' \subseteq \mathcal{F} \\ \mathcal{F}' \sqsubseteq \mathcal{R}(A)}} \sum_{f \in \mathcal{F}'} \mathcal{P}(f) \tag{1}$$

Example Suppose that we are given transition system A in Fig. 4 as abstraction. Suppose furthermore that fragments $(0, 4, 5)$, $(1, 2, 4)$, $(0, 1)$ and $(4, 3)$ have not been validated, yet. Assume an associated weight of 2 with validating fragment $(0, 4, 5)$, a weight of 3 with $(1, 2, 4)$, and a weight of 1 with fragments $(0, 1)$ and $(4, 3)$. What subset of these fragments is the cutting set with the lowest sum of weights? Obviously, we have
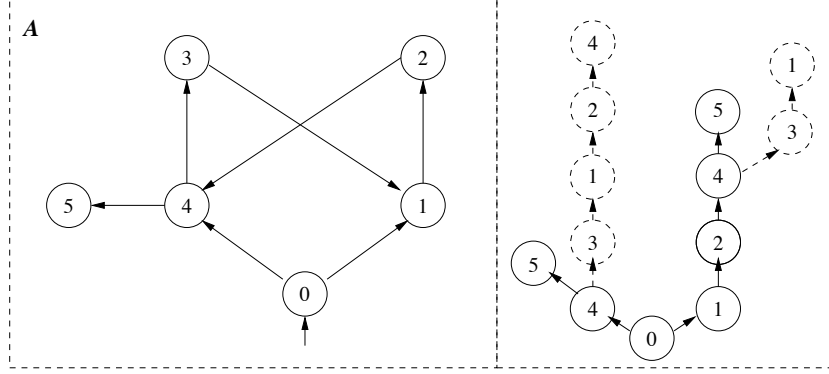
Fig. 4. Left: A finite transition system A. Right: The depth-first unrolling of A. The unrolling stops if either the final state 5 is reached (solid), or if a loop has been detected (dashed).

to include fragment $(0, 4, 5)$ in any cutting set. But is the set with fragments $(0, 4, 5)$ and $(0, 1)$ sufficient? After all, this set cuts all loop-free accepting runs.

Somewhat surprisingly, there exist an accepting run that is not covered by fragment $(0, 4, 5)$ or fragment $(0, 1)$. Neither cuts accepting run $(0, 4, 3, 1, 2, 4, 5)$, although fragment $(0, 4, 5)$ cuts it trivially once we remove loop $(4, 3, 1, 2, 4)$. This demonstrates that the problem of finding cutting sets of fragments is not a simple cut set problem in a network flow graph, for which it would be sufficient to concentrate only on loop-free runs. ∎

Standard cut set algorithm cannot be applied directly, since fragments in $\mathcal{F}$ are not represented by single transitions in A. To solve this problem this we define a collection of transition systems that *observe* A. This requires an extension of the notion of transition systems to *automata*.

Definition 9. *Given a set of labels Σ, an* automaton *A is a tuple $(\Sigma, S, S^0, S^f, R, L)$ where (S, S^0, S^f, R) is a transition system and $L : R \to 2^{\Sigma}$ a labelling function.*

Definition 10. *Let $A_i = (\Sigma_i, S_i, S_i^0, S_i^f, R_i)$ be a finite number of automata, $i = 1, \ldots, n$. The synchronous composition $A = A_1 || \ldots || A_n$ is an automaton (Σ, S, S^0, S^f, R) with*

- $\Sigma = \bigcup_{i \in \{1, \ldots, n\}} \Sigma_i$, *the union of all alphabets.*
- $S = S_1 \times \ldots \times S_n$, $S^0 = S_1^0 \times \ldots \times S_n^0$, *and* $S^f = S_1^f \times \ldots \times S_n^f$. *The projection $s|_{S_i}$ will be denoted as s_i.*
- $(s, s') \in R$ *if* $\bigcap_{i \in \{1, \ldots, n\}} L_i(s_i, s_i')$ *is nonempty.*
- $L(s, s') = \bigcap_{i \in \{1, \ldots, n\}} L_i(s_i, s_i')$.

Note that this is a very restricted notion of composition. The composition automaton can only take a transition if all automata can take a transition with the same label. However, the observing automata will be constructed such that they can synchronize always with any transition in the observed automaton. Given a transition system A and
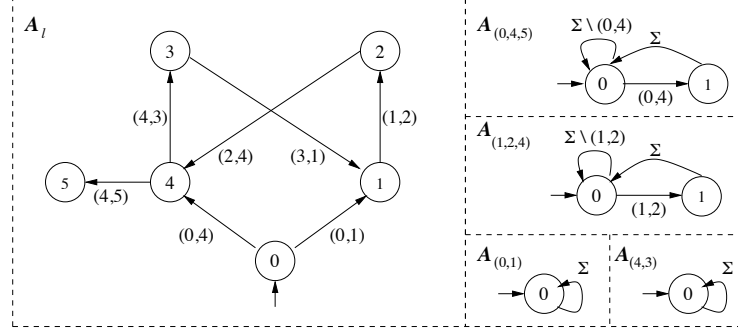
Fig. 5. The automata $A_{(0,4,5)}$, $A_{(1,2,4)}$, $A_{(0,1)}$, and $A_{(4,3)}$ observe the transitions in A_l. The only accepting state of A_l is the state 5.

a set of fragments $\mathcal{F}$ of A, the procedure first extends A with labels, and then introduces for each fragment in $\mathcal{F}$ a small automaton that observes the occurrence of a fragment. The steps to obtain the observing automata are the following:

1. Extend $A = (S, S^0, S^f, R)$ to a automaton $A_l = (\Sigma, S, S^0, S^f, R)$ with $\Sigma = R$ and L mapping $(s, s') \mapsto \{(s, s')\}$.
2. For each $\varrho \in \mathcal{F}$, $\varrho = (s_0, \ldots, s_{n-1})$, introduce an observer automaton $A_\varrho = (\Sigma_\varrho, S_\varrho, S_\varrho^0, S_\varrho^f, R_\varrho, L_\varrho)$ with
   - $\Sigma_\varrho = \Sigma$
   - $S_\varrho = \{t_0, \ldots, t_{n-1}\}$, where n is the length of fragment ϱ.
   - $S_\varrho^0 = \{t_0\}$ and $S_\varrho^f = S_\varrho$
   - R_ϱ is the set $\{(t_i, t_{i+1}) | i = 0, \ldots, n-2\} \cup \{(t_i, t_0) | i = 0, \ldots, n-1\}\}$
   - and L_ϱ is the following mapping

$$(t, t') \mapsto \begin{cases} (s_{i-1}, s_i) & \text{if } (t, t') = (t_i, t_{i+1}), \ i = 0, \ldots, n-2 \\ \Sigma \setminus (s_{i-1}, s_i) & \text{if } (t, t') \neq (t_i, t_{i+1}), \ i = 0, \ldots, n-2 \\ \Sigma & \text{if } (t, t') = (t_{n-1}, t_n) \end{cases}$$

The next step composes the labelled transition system A_l with the observer automata A_ϱ for all $\varrho \in \mathcal{F}$. This composition will be denoted as $A_\mathcal{F}$.

Example (Cont) Given the transition system A in Fig. 4, the first step is to obtain A_l by adding labels (Fig. 5). Recall that $\mathcal{F} = \{(0, 4, 5), (1, 2, 4), (0, 1), (4, 3)\}$, and $\mathcal{P}(0, 4, 5) = 2$, $\mathcal{P}(1, 2, 4) = 3$, $\mathcal{P}(0, 1) = 1$ and $\mathcal{P}(4, 3) = 1$. The next step includes a small observing automaton for each fragment (Fig. 5). The automaton for fragment $(0, 4, 5)$ has as many states as the fragment has transitions. In each state the automaton can synchronize with any transition in A_l.

If transition $(0, 4)$ occurs in A_l the observing automaton $A_{(0,4,5)}$ takes a transition from state 0 to state 1. If transition $(4, 5)$ occurs right after the first transition, the observing automaton will take a transition back to the initial state. This corresponds to the transition from $(4, 1, 0, 0, 0)^T$ to $(5, 0, 0, 0, 0)^T$ in composition $A_\mathcal{F}$ in Fig. 6. This transition marks an occurrence of the fragment $(0, 4, 5)$ in A_l.
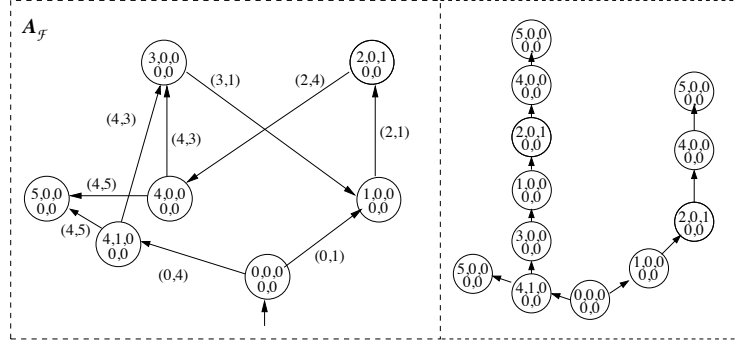
Fig. 6. Composition automaton $A_{\mathcal{F}}$, and the tree of all loop-free accepting runs.

Figure 6 depicts the composition automaton $A_{\mathcal{F}}$, and the tree of all loop-free counterexamples. Note that there are two transitions labelled $(4,3)$ in $A_{\mathcal{F}}$. Transition $(4,3)$ is an element of the set of fragments $\mathcal{F}$ that have not been validated yet. If we one could show that $(4,3)$ is spurious, it would eliminate both arcs in the graph in one go. Obviously, cut set algorithm for network flows cannot be used, since several arcs in $A_{\mathcal{F}}$ can represent the same fragment of A_l.

The set with $((4,1,0,0,0)^T,(5,0,0,0,0)^T)$ and $((2,1,0,0,0)^T,(4,0,0,0,0)^T)$ cuts composition $A_{\mathcal{F}}$[1]. These transitions in $A_{\mathcal{F}}$ mark the occurrence of fragments $(0,4,5)$ and $(1,2,4)$ in A. The overall weight of this set is 5. The tree also shows that the set containing fragments $(0,4,5)$ and $(0,1)$ does not cut A_l, since transitions $((4,1,0,0,0)^T,(5,0,0,0,0)^T)$ and $((0,0,0,0,0)^T,(1,0,0,0,0)^T)$ do not cut $A_{\mathcal{F}}$. It also shows that $(0,4,5),(1,0)$ and $(4,3)$ are a cutting set of A, with an associate weight of 4. This is the optimal cutting set for this example.

Note that the observers for the fragments $(0,1)$ and $(4,3)$ do not add anything and could be omitted. Likewise, one observer for fragments that are equal except for the last transition would be sufficient. However, maintaining those observers does not increase the size of the composition, and we chose to maintain them in this paper to treat the different fragments consistently. ∎

The construction of A_l ensures that each transition has a unique label. Consequently A_l is deterministic. All observers are deterministic, too, and can synchronize in each state with any transition of A_l. The behavior of A_l is not restricted by the observers. This yields a close relationship between $A_{\mathcal{F}}$ and A_l, and thus between $A_{\mathcal{F}}$ and A. As a matter of fact for each $\pi \in \mathcal{R}(A)$ there exist a $\pi_{\mathcal{F}} \in \mathcal{R}(A_{\mathcal{F}})$ such that $\pi_{\mathcal{F}}|_S = \pi$, and vice versa.

Lemma 1. *Given a transition system A, a set of fragment $\mathcal{F}$ of A and the composition automaton $A_{\mathcal{F}}$, the following holds.*

---

[1] Column vectors are used for elements of product state spaces to distinguish them from tuples of states that are fragments.

(i) A subset $\mathcal{F}' \subseteq \mathcal{F}$ cuts transitions system A, i.e. $\mathcal{F} \sqsubseteq \mathcal{R}(A)$ iff for all $\pi_{\mathcal{F}} \in \mathcal{R}(A_{\mathcal{F}})$ there exists $\varrho \in \mathcal{F}'$ such that $\varrho \sqsubseteq \pi_{\mathcal{F}}|_S$.

(ii) Given $\varrho = (s_0, \ldots, s_n)$ the following holds: $\varrho \sqsubseteq \pi_{\mathcal{F}}|_S$ iff the projection of the path $\pi_{\mathcal{F}}$ to $S \times S_{\varrho}$ contains a transition from $(s_{n-1}, t_{n-1})^T$ to $(s_n, t_0)^T$.

Proof: (i) This follows straightforward from the observation that $\pi_{\mathcal{F}}|_S$ is in $\mathcal{R}(A)$ for all $\pi_{\mathcal{F}} \in \mathcal{R}(A_{\mathcal{F}})$, and that for all path $\pi \in \mathcal{R}(A)$ there exists an $\pi_{\mathcal{F}}$, with $\pi_{\mathcal{F}}|_S = \pi$.
(ii)"$\Rightarrow$" Transition $(s_{n-1}, t_{n-1})^T$ to $(s_n, t_0)^T$ can only be taken if it was immediately preceded by transitions synchronizing on labels $(s_{n-i-1}, s_{n-i})^T$, for $i = 1, \ldots, n-1$.
"$\Leftarrow$". Let $\pi_{\mathcal{F}}|_S = (z_0, \ldots, z_m)$ and $\pi_{\mathcal{F}}|_{S_{\varrho}} = (z'_0, \ldots, z'_m)$. By definition, $\varrho \sqsubseteq \pi_{\mathcal{F}}|_S$ iff there exists a k such that $z_{k+i} = s_i$ for $i = 0, \ldots, n$.

First, we show that A_{ϱ} is in its initial state after the k-th transition of $\pi_{\mathcal{F}}$, that is, $z'_k = t_0$. If $k = 0$, $z'_k = t_0$ holds trivially. When $k > 0$ we have the following: (z_{k-1}, z_k) is a transition of A_l, but it is not a transition of fragment ϱ. The latter holds because $z_k = s_0$ and ϱ is loop-free. Since (z_{k-1}, z_k) is not in ϱ it can synchronize only with a transition of A_{ϱ} that leads to its initial state. This implies that $z'_k = t_0$. The transition (z_{k+i}, z_{k+i+1}) will then synchronize with (z'_{k+i}, z'_{k+i+1}) on label (s_i, s_{i+1}), for $i = 0, \ldots, n-2$. At this point A_{ϱ} will be in state t_{n-1}. In this state, transition (z_{n-1}, z_n) of A_l can only synchronize with transition (t_{n-1}, t_0) of A_{ϱ} on label (s_{n-1}, s_n), which concludes the proof. ∎

On first sight little has been gained. Rather than selecting subsets of $\mathcal{F}$ that cut A, the procedure can select subsets of $R_{\mathcal{F}}$, the transitions of $A_{\mathcal{F}}$, that cuts $A_{\mathcal{F}}$. But the advantage of looking at transitions rather than fragments is that it becomes sufficient to look only at loop-free accepting runs. A set $R'_{\mathcal{F}} \subseteq R_{\mathcal{F}}$ that cuts all loop-free accepting runs, also cuts all accepting runs. Let $\mathcal{R}_{lf}(A_{\mathcal{F}})$ be the set of all loop-free accepting runs.

Lemma 2. *Given a transition system A, a set of fragment $\mathcal{F}$ of A and the composition automaton $A_{\mathcal{F}}$. Let $R'_{\mathcal{F}} \subseteq R_{\mathcal{F}}$, then $R'_{\mathcal{F}} \sqsubseteq \mathcal{R}_{lf}(A_{\mathcal{F}})$ iff $R'_{\mathcal{F}} \sqsubseteq \mathcal{R}(A_{\mathcal{F}})$.*

Proof: "$\Rightarrow$" Suppose that we have an accepting run $\pi_{\mathcal{F}} \in \mathcal{R}(A_{\mathcal{F}})$. From this we can obtain a loop-free accepting run $\pi'_{\mathcal{F}}$ by eliminating all loops. According to the precondition there exists a $\varrho \in R'_{\mathcal{F}}$ such that $\varrho \sqsubseteq \pi'_{\mathcal{F}}$, which means that transition ϱ appears somewhere in $\pi'_{\mathcal{F}}$. Since the transitions that occur in $\pi_{\mathcal{F}}$ are a super-set of those that appear in $\pi'_{\mathcal{F}}$ we have $\varrho \sqsubseteq \pi'_{\mathcal{F}}$, too. "$\Leftarrow$" If a set of transitions cuts all accepting runs, it will cut all loop-free accepting runs. ∎

Lemma 2 allows the consideration of only loop-free accepting runs of $A_{\mathcal{F}}$. However, the example demonstrates that a cut set algorithm for network flows cannot be used to find a cut set of $A_{\mathcal{F}}$, since certain fragments of A may be represented by multiple transitions in $A_{\mathcal{F}}$. The cutting set problem can be solved by a translation to a set cover problem. A similar approach has been used in [10] to find cut sets for attack graphs. The cut set determines what actions of the attacker have to be disabled to prevent future attacks.

Given a finite (universal) set $\mathcal{U}$ and a set of sets $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ with $\mathcal{C}_i \subseteq \mathcal{U}$ as input, a set cover algorithm computes the smallest subset $\mathcal{C}_{opt} \subseteq \mathcal{C}$ such that $\bigcup_{\mathcal{C}_i \in \mathcal{C}_{opt}} \mathcal{C}_i = \mathcal{U}$. The set cover problem is NP-complete, but a greedy approach is

guaranteed to find an solution that is at most $\lg n$ as bad as the optimal solution in polynomial time (where n is the number of elements of $\mathcal{U}$) [11]. The greedy algorithm picks in each iteration the set from $\mathcal{S}$ that covers the greatest number of uncovered elements of $\mathcal{U}$, until the complete set $\mathcal{U}$ is covered.

The problem of finding a cutting set of fragments is a set cover problem, where the universal set is $\mathcal{R}_{lf}(A_{\mathcal{F}})$, and $\mathcal{C}$ contains for each fragment ϱ in $\mathcal{F}$ set $\mathcal{C}_{\varrho} = \{\pi_{\mathcal{F}} \in \mathcal{R}_{lf}(A_{\mathcal{F}}) | \varrho \sqsubseteq \pi_{\mathcal{F}}|_{S}\}$. The problem is to find an optimal subset of $\mathcal{C}$ that covers $\mathcal{R}_{lf}(A_{\mathcal{F}})$. We compute the sets $\mathcal{C}_{\varrho}$ by a depth-first exploration of $A_{\mathcal{F}}$, that starts backtracking if it either finds a loop, or reaches an accepting state. In the latter case it adds the accepting run to $\mathcal{C}_{\varrho}$ if $\varrho \sqsubseteq \pi$.

We modify the greedy algorithm to accommodate the fact that we are not looking for the smallest cover of $\mathcal{R}_{lf}(A_{\mathcal{F}})$, but for an optimal one. In each iteration this modified greedy algorithm adds the set $\mathcal{C}_{\varrho}$ to $\mathcal{C}_{opt}$ that has the smallest associated cost $\mathcal{P}(\varrho)$ per covered run. In the latter it considers only runs that have not been covered in earlier iterations. When all runs in $\mathcal{R}_{lf}(A_{\mathcal{F}})$ are covered the procedure tests if the set obtained by removing some $\mathcal{C}_{\varrho}$ from $\mathcal{C}_{opt}$ covers all runs. If this is the case, $\mathcal{C}_{\varrho}$ will be omitted from $\mathcal{C}_{opt}$.

The overall procedure to compute cutting sets of fragments can be summarized as follows. Given a finite transition system A and a set of loop-free fragments $\mathcal{F}$, construct composition automaton $A_{\mathcal{F}}$. Then compute all loop-free accepting runs $\mathcal{R}_{lf}(A_{\mathcal{F}})$. For each fragment ϱ compute the corresponding set $\mathcal{C}_{\varrho}$, which contains $\pi_{\mathcal{F}} \in \mathcal{R}_{lf}(A_{\mathcal{F}})$, if $\varrho \sqsubseteq \pi_{\mathcal{F}}|_{S}$. Finally, compute the optimal set cover $\mathcal{C}_{opt}$. The optimal cutting set of fragments $\mathcal{F}_{opt}$ contains all fragments ϱ with $\mathcal{C}_{\varrho} \in \mathcal{C}_{opt}$.

Example (cont) The composition automaton $A_{\mathcal{F}}$ has only three loop-free counterexamples (Fig. 6). The following tables shows which of these, projected to S, is cut by what fragment:

	$(0,4,5)$	$(1,2,4)$	$(0,1)$	$(4,3)$
$(0,4,5)$	x			
$(0,4,3,1,2,4,5)$		x		x
$(0,1,2,4,5)$		x	x	

Given the set $\mathcal{F} = \{(0,4,5),(1,2,4),(0,1),(4,3)\}$ there are two sets of fragments that cover all accepting runs: $\{(0,4,5),(1,2,4)\}$ and $\{(0,4,5),(0,1),(4,3)\}$. The latter is optimal with an overall weight of 4. ∎

6 Example

This section uses an adaptive cruise control system to illustrate the fragment-based approach. This example was used in [3] to illustrate counterexample-guided abstraction refinement. The results in [3] show that fragments can reduce the computation time by an order of magnitude. However, in [3] validating fragments is in addition to a counterexample guided abstraction loop. This paper generalized the concept of fragments to guide the abstraction. This section describes what sets of fragments will be selected, but omits in most cases to list the new fragment in $\mathcal{F}$ for brevity.

The adaptive cruise control system is part of a vehicle-to-vehicle coordination system [6]. This system comprises two modes: cruise control mode (cc-mode), and an adaptive cruise control mode (acc-mode). In acc-mode the controller tries to keep a safe distance to the vehicle ahead. The acc-controller switches into acc-mode whenever the distance between the car and a vehicle ahead falls below a certain threshold. This threshold depends linearly on car speed.

The hybrid automaton is a composition of an automatic transmission with 4 gears, an acc-controller with two modes (acc and cc), and an error state for collisions. Figure 7.A depicts the initial abstraction of the hybrid automaton.

Initially, the set of fragments contains all transitions. From this set we select the first cut set; it contains transition $(1, 2)$ and $(1, 5)$ (labelled i in Fig. 7.A). Two different methods methods are used for validation. The first method $\overline{succ}_{coarse}$ formulates the question if a trajectory between S_1 and S_2 exists as an optimization problem. The second method $\overline{succ}_{tight}$ computes polyhedra that enclose all trajectories that originate in S_1. This overapproximation with polyhedra is based on work presented in [2]. Both methods were also presented in [12]. The default method for single transitions is $\overline{succ}_{coarse}$. It has an associated weight of 1.

Validation of $(1, 2)$ shows that this fragment is valid. It will be removed $(1, 2)$ from $\mathcal{F}$, but we cannot add the extension $(1, 2, 10)$, $(1, 2, 6)$, $(1, 2, 3)$, and $(5, 1, 2)$ since $(2, 10)$, $(2, 6)$, $(2, 3)$, and $(5, 1)$ are still in $\mathcal{F}$. Otherwise it would violate the restriction that no fragment in $\mathcal{F}$ should be cut by another. The second fragment $(1, 5)$ is spurious. Transition $(1, 5)$ is removed from the abstraction A and from the set of fragments $\mathcal{F}$. The next cutting set is the set of transitions labelled ii in Fig. 7.A. Fragment $(2, 6)$ is spurious, this transition can be removed from the abstraction. We also remove $(2, 6)$ from $\mathcal{F}$, and can replace it by $(1, 2, 6)$, since $(1, 2)$ was removed in the previous iteration. There is no $\varrho \in \mathcal{F}$ with $\varrho \sqsubseteq (1, 2, 6)$. We proceed in the same way with transition $(2, 10)$ which is also spurious. The next cutting set is computed (iii). It contains no spurious transitions, and the next cutting set will be selected (iv). From this set only fragment $(7, 6)$ is spurious, and A and $\mathcal{F}$ will be refined accordingly.

Figure 7.B depicts the abstraction after the first four iterations. At this stage the abstraction cannot be cut by single transitions. We select fragment $(1, 2, 3)$ (labelled v). Given a pair of transitions we use first method $\overline{succ}_{coarse}$ for validation. We associate a weight of 2 with this operation. Fragment $(1, 2, 3)$ is valid and we select a new cutting set with fragments $(3, 4)$ and $(2, 3, 7)$ (vi). Both of them are valid. The next cutting set (vii) has two spurious fragments, $(7, 8)$ and $(3, 7, 9)$. Hence we refine the abstraction accordingly, and add new fragments for the new transitions. Figure 7.C depicts the abstraction after refinement.

Fragment $(4, 8, 9)$ of the next cutting set $(viii)$ is spurious. We refine the abstraction by splitting state 8 (Fig. 7.D). The next cutting set (ix) is the single transition from the newly created state 14 to state 11. This transition is not spurious. Next we try to refute fragment $(14, 11, 9)$, $(4, 14, 11)$, and $(2, 3, 4)$, which are cutting sets x, xi, and xii, respectively.

Up to now, only $\overline{succ}_{coarse}$ was used to validate fragments. The next cutting set is $(14, 11, 9)$. We validated this fragment before with $\overline{succ}_{coarse}$, but we use this time the second method $\overline{succ}_{tight}$. This validation has an associated weight of 6. Using the more
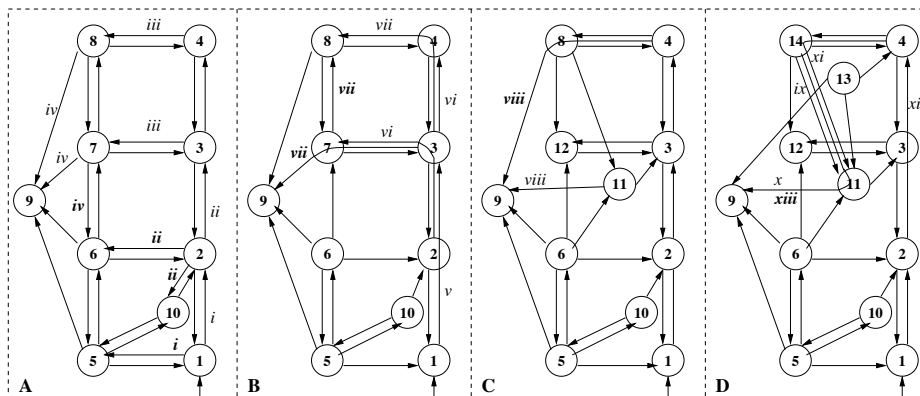
Fig. 7. Adaptive cruise control example. The initial sate is state 1, the final state, that models collision, is state 9. Fragments labelled with bold-face, roman numerals have been shown to be spurious.

accurate method we find that $(14, 11, 9)$ is spurious. Since it is the only fragment of the cutting set, the verification is done and state 9 proven to be not reachable.

7 Conclusions and Future Work

This paper presents a method for guiding abstraction refinement for hybrid systems using sets of fragments of counterexamples, building on the notion of fragments that was introduced in [3]. We use the novel concept of cutting sets of fragments. These cutting sets of fragments focus the analysis, very much like cut sets in networks flows focus on bottlenecks. The aim is to refute as many counterexample as possible while minimizing the expected computational effort.

The notion of optimal cutting set of fragments may also be useful to obtain diagnostic information from counterexamples [7]. This area of research tries to identify parts of counterexample that occur frequent and are causal to the error.

We are currently working on a prototype implementation that build on the implementation used in [3] and will apply them to benchmarks presented in [5]. The procedure presented in this paper leaves room for many heuristic choices, for example what mix of overapproximation methods is useful for what fragments, and how to assign weights to validations. Proper heuristics will be further developed as soon as a prototype becomes available. We plan to provide first results for the final version of the paper.

References

1. R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid system. In *TACAS*, volume 2619 of *LNCS*. Springer, 2003.

2. A. Chutinan and B.H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In F.W. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 76–90. Springer Verlag, 1999.

3. E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(4):583–604, 2003.

4. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer-Aided Verification*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

5. A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *Proc. of HSCC'04*, LNCS. Springer, 2004.

6. A.R. Girard, J.B. Souza, J.A. Misener, and J.K. Hedrick. A control architecture for integrated cooperative cruise control and collision warning systems. In *Proc. 40th IEEE Conf. on Decision and Control*, 2001.

7. A. Groce and W. Visser. What went wrong: Explaining counterexamples. In *SPIN Workshop on Model Checking of Software*, LNCS 2648. Springer, 2003.

8. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

9. R. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.

10. O. Sheyner. *Scenario Graphs and Attack Graphs*. PhD thesis, SCS, Carnegie Mellon University, 2004.

11. S. Skiena. *The Algorithm Design Manual*. Telos/Springer-Verlag, 1998.

12. O. Stursberg, A. Fehnker, Zhi Han, and B. Krogh. Specification-guided analysis of hybrid systems using a hierachy of validation methods. In *Proc. IFAC Conference ADHS*. Elsevier, 2003.

13. A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, March 2002.
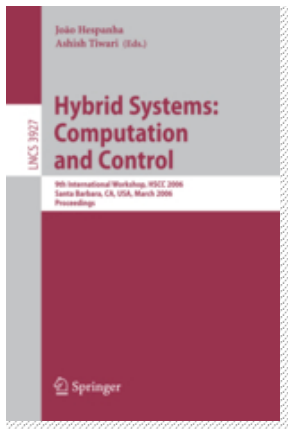
Hybrid Systems: Computation and Control

9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings
Series: » Lecture Notes in Computer Science, Vol. 3927
Subseries: » Theoretical Computer Science and General Issues

Hespanha, Joao; Tiwari, Ashish (Eds.)

2006, XII, 584 p.

Softcover
ISBN 978-3-540-33170-4
Usually dispatched within 3 to 5 business days.
(net)   **$109.00**

[f Like]   [▼ Tweet] ⟨0⟩   [8+1] ⟨0⟩

About this book

This book constitutes the refereed proceedings of the 9th International Workshop on Hybrid Systems: Computation and Control, HSCC 2006, held in Santa Barbara, CA, USA in March 2006.

The 39 revised full papers presented together with the abstracts of 3 invited talks were carefully reviewed and selected from 79 submissions. Among the topics addressed are tools for analysis and verification, control and optimization, modeling, engineering applications, and emerging directions in programming language support and implementation. The papers focus on modeling, analysis, and implementation of dynamic and reactive systems involving both discrete and continuous behaviors.

Content Level » Research

Keywords » dependable systems - dynamical systems - embedded systems - formal verification - hybrid control systems - hybrid systems - real-time systems - stochastic hybrid systems - systems analysis - systems modeling - systems verification

Related subjects » Communication Networks - Software Engineering - Theoretical Computer Science

DOWNLOADS

7 days   30 days   90 days