Compositional Reachability of Hybrid Systems

Alessandro Cimatti Sergio Mover Stefano Tonetta
Fondazione Bruno Kessler – FBK-irst – Embedded System Unit – Italy
{cimatti,mover,tonettas}@fbk.eu

***Abstract*—Hybrid automata provide a comprehensive framework for the representation of practical systems with discrete and continuous dynamics. Their interaction is paradigmatic of partially synchronized systems, which may run asynchronously or synchronize on common events.**

In this paper, we propose a compositional approach to the reachability analysis of hybrid systems. Differently from most works in compositional reasoning, that focus on proving universal properties, here the purpose is to provide witnesses to existential properties.

The role of compositionality is to reduce the effort of building such witnesses, by replacing a monolithic search on the whole network of systems with a set of searches on suitably constrained components.

The proposed method selects a component in the network under analysis, and applies, to each of the other components, either an over-approximation or an under-approximation. Intuitively, the over-approximations can be seen as abstractions of the environment visible to the selected component, while the under-approximations impose constraints resulting from previously found partial solutions. The approach carries out a backtracking search in the space of network approximations.

We instantiated the compositional analysis for checking bounded reachability. We rely on advanced use of SMT techniques to implement the necessary steps, including abstraction, generalization, and learning. The experimental results demonstrate the potential of the approach.

I. Introduction

Hybrid automata ([16]) are increasingly recognized as a clean modeling framework for systems with discrete and continuous variables. Many systems are structured into components, and can often be naturally modeled as networks of communicating hybrid automata: local activities of each component amounts to transitions local to each hybrid automaton; communications and other events that are shared between/visible for various components are modeled as synchronizing transitions of the automata in the network; time elapse is modeled as shared timed transitions.

In this paper, we propose a compositional approach to the reachability analysis of hybrid systems. The role of compositionality is to reduce the effort of building reachability witnesses, by replacing a monolithic search on the whole network of systems with a set of searches on suitably constrained components.

The proposed method selects a component in the network under analysis, and applies, to each of the other components, either an over-approximation or an under-approximation. Intuitively, the over-approximations can be seen as abstractions of the environment visible to the selected component, while the under-approximations impose constraints resulting from previously found partial solutions. The approach carries out a backtracking search in the space of network approximations. When a conflict is found, the cause of the conflict is analyzed, and the under-approximated components that are responsible for the inconsistency are reconsidered to choose an alternative under-approximation.

Compared to most works in compositional reasoning (e.g. [17], [13]), that aim at proving universal properties, the proposed approach focusses on providing witnesses to existential properties. In addition, we combine reasoning based on overapproximations with a dual form of reasoning that exploits under-approximations. Compared to approaches tailored to the falsification of safety properties (e.g. [12], [23]), we notice that they implement a monolithic search without exploiting the compositionality of the network, and can thus be used as a backend within the approach proposed here.

Our framework is quite general and can be instantiated in several different ways. Here, we choose 1) approximations based on state-space restrictions and constraints relaxation, 2) a bounded model checking engine, 3) a search on the network components based on a fixed order of the processes.

We rely on advanced use of SMT techniques to implement the necessary steps, exploiting the solver's incrementality to perform bounded model checking and conflict analysis.

The experimental results demonstrate the potential of the approach. In particular, while the compositional approach in general pays a substantial overhead due to the number of performed searches, compared to a monolithic approach, the benefits are evident when we scale up the size of the network and the complexity of the components.

Outline: The paper is structured as follows. In Section II we present some background. In Section III we discuss the exploited rules of compositionality and describe the compositional analysis algorithm. In Section we discuss some specific choices in our approach. In particular, in Section IV-A we present the component approximations, in Section IV-B we describe the bounded reachability search based on SMT technologies, and in Section IV-C we discuss the backtracking details. In Section V we discuss some relevant related work. In Section VI we experimentally evaluate the approach. In Section VII we draw some conclusions and future work.

II. Background

A. Networks of transition systems

The following definitions are standard notions of concurrency theory (see, e.g., [24]).

Definition 1 (Labelled Transition System): A Labelled Transition System (LTS) is a tuple $\langle Q, A, I, T \rangle$ where

- Q is the set of states,

- A is the set of actions/events,
- $I \subseteq Q$ is the set of initial states,
- $T \subseteq Q \times A \times Q$ is the set of labeled transitions.

Definition 2 (Trace): A trace is a sequence of actions $\pi = a_1, \ldots, a_k \in A^*$. Given $A' \subseteq A$, the projection $\pi \restriction_{A'}$ of π on A' is the sub-trace of π obtained by removing all actions in π that are not in A'.

Definition 3 (Run): A run σ of S over the trace $\pi = a_1, \ldots, a_k \in A^*$ is a sequence $q_0, q_1, \ldots, q_k$ of states of S such that $q_0 \in I$ and, $\langle q_{i-1}, a_i, q_i \rangle \in T$ for all i such that $1 \leq i \leq k$. We say that σ accepts π. Given a predicate $p \subseteq Q$ we say that σ terminates in p iff $q_k \in p$.

Definition 4 (Language): The language $L(S)$ of an LTS S is the set of traces accepted by some run of S. Two LTSs are equivalent if they have the same language. Given a predicate $p \subseteq Q$, the language $L_p(S)$ of an LTS S is the set of traces accepted by some run of S terminating in p.

Definition 5 (Parallel composition): The parallel composition $S_1||S_2$ of two LTSs $S_1 = \langle Q_1, A_1, I_1, T_1 \rangle$ and $S_2 = \langle Q_2, A_2, I_2, T_2 \rangle$ is the LTS $\langle Q, A, I, T \rangle$ where

- $Q = Q_1 \times Q_2$,
- $A_1 \cup A_2$,
- $I = I_1 \times I_2$,
- $T :=$
  $\{\langle q_1 \times q_2, a, q_1' \times q_2' \rangle \mid \langle q_1, a, q_1' \rangle \in T_1, \langle q_2, a, q_2' \rangle \in T_2\}$
  $\cup \{\langle q_1 \times q_2, a, q_1' \times q_2 \rangle \mid \langle q_1, a, q_1' \rangle \in T_1, a \notin A_2\}$
  $\cup \{\langle q_1 \times q_2, a, q_1 \times q_2' \rangle \mid \langle q_2, a, q_2' \rangle \in T_2, a \notin A_1\}$.

Similarly, we define the parallel composition $\sigma_1||\sigma_2$ of the run σ_1 of S_1 and the run σ_2 of S_2.

Theorem 1 (Associativity and commutativity [24], [8]): The parallel composition is associative and commutative modulo language equivalence.

Definition 6 (Network): A network $\mathcal{N}$ is a set of LTSs $\{S_1, \ldots S_n\}$. The language of a network is the language of the composition of the LTSs, i.e., $L(\mathcal{N}) = L(S_1|| \ldots ||S_n)$.

Definition 7 (Language emptiness problem): Given a network, the language emptiness problem is the problem of checking if the language of a network is empty.

Definition 8 (Reachability problem): Given a network $\mathcal{N}$ and a predicate $p \subseteq Q_1 \times \ldots \times Q_n$, the reachability problem is the problem of checking if the language $L_p(\mathcal{N})$ is empty.

Theorem 2 (Composition language [24], [8]): A trace π is accepted by $S_1||S_2$ iff $\pi \restriction_{A_1}$ is accepted by S_1 and $\pi \restriction_{A_2}$ is accepted by S_2. Thus,

$$L(S_1||S_2) = \{\pi \in (A_1 \cup A_2)^* \mid \pi \restriction_{A_1} \in L(S_1), \pi \restriction_{A_2} \in L(S_2)\}$$

Remark 1: The set of actions is not restricted to be finite. Thus, it is possible to model also variable sharing where the two systems may synchronize the value of the variables on a common action. In fact, it is sufficient to encode the value of the variable in the action itself: for example, if on action a two system synchronize the value of the real variable x, we use the set of actions $\{\langle a, \underline{x} \rangle\}_{\underline{x} \in \mathbb{R}}$ and force that a transition $\langle q, \langle a, \underline{x} \rangle, q' \rangle$ exists only if the value of x in q is $\underline{x}$.

B. Hybrid systems

Definition 9 (Hybrid Automata): A Hybrid Automaton (HA) is a tuple
$\langle Q, A, I, T, X, \mu, \iota, \xi, \theta \rangle$ where

- Q is the set of states,
- A is the set of actions,
- $I \subseteq Q$ is the set of initial states,
- $T \subseteq Q \times A \times Q$ is the set of discrete transitions,
- X is the set of continuous variables,
- $\mu : Q \to P(X, \dot{X})$ is the flow condition,
- $\iota : Q \to P(X)$ is the initial condition,
- $\xi : Q \to P(X)$ is the invariant condition,
- $\theta : T \to P(X, X')$ is the jump condition.

Definition 10: A network $\mathcal{H}$ of HAs is a set of HAs.

Definition 11 (Semantics): Consider a network $\mathcal{H} = \{H_1, \ldots, H_n\}$ of HAs with $H_i = \langle Q_i, A_i, I_i, T_i, X_i, \mu_i, \iota_i, \xi_i, \theta_i \rangle$. The semantics of $\mathcal{H}$ is the network of LTSs $\mathcal{N}_{DELTA} = \{S_1, \ldots, S_n\}$ with $S_i = \langle Q_i', A_i', I_i', T_i' \rangle$ where

- $Q_i' = \{\langle q, \overline{x} \rangle \mid q \in Q_i, \overline{x} \in \mathbb{R}^{|X_i|}\}$,
- $A_i' = A_i \cup \{\langle \textsc{Time}, \delta \rangle \mid \delta \in \mathbb{R}\}$,
- $I_i' = \{\langle q, \overline{x} \rangle \mid q \in I_i, \overline{x} \in \iota_i(q)\}$,
- $T_i' =$
  $\{\langle q, \overline{x}, a, q', \overline{x}' \rangle \mid \langle q, a, q' \rangle \in T_i, \langle \overline{x}, \overline{x}' \rangle \in \theta_i(q, a, q'), \overline{x} \in \xi_i(q), \overline{x}' \in \xi_i(q')\} \cup$
  $\{\langle q, \overline{x}, \langle \textsc{Time}, \delta \rangle, q, \overline{x}' \rangle \mid$ there exists f satisfying $\mu_i(q)$ s.t. $f(0) = \overline{x}, f(\delta) = \overline{x}', f(\epsilon) \in \xi(q), \epsilon \in [0, \delta]\}$.

III. Search-based compositional reachability analysis

A. Compositional reasoning based on under-approximations

In this paper, for *compositional analysis*, we mean checking if there exists a trace $\pi \in A = \bigcup_{S \in \mathcal{N}} A_S$ such that $\pi \restriction_{A_S} \in L(S)$ for all $S \in \mathcal{N}$; for *monolithic analysis*, we mean building the parallel composition of the systems in the network and checking if there exists a trace accepted by the composition system.

The standard application of compositional reasoning is the verification of universal properties such as safety and liveness. Thus, the typical approaches use over-approximations of the components to prove the properties on a simplified network, and conclude that the properties hold also on the original concrete network.

Since we are interested in existential properties, such as finding a witness to the violation of a safety property or the emptiness of the language of a network, over-approximations are not suitable.

Given a component S, we denote with $\hat{S}$ an over-approximation of S and with $\check{S}$ an under-approximation of S.

Definition 12 (Over-Approximation): An LTS $\hat{S}$ is an over-approximation of the LTS S iff $L(S) \subseteq L(\hat{S})$.

Algorithm *CompositionalAnalysis*($Network\mathcal{N}$)

1. $\hat{\mathcal{N}}$ = *InitialOverApprox* ($\mathcal{N}$)
2. *Over* = $\mathcal{N}$
3. *Under* = $\emptyset$
4. **while** *Over* $\neq \emptyset$
5. S_c = *PickComponent* (*Over*)
6. *Over* = *Over* $\setminus \{S_c\}$
7. $\tilde{\mathcal{N}} = S_c \times \prod_{S \in Under} \check{S} \times \prod_{S \in Over} \hat{S}$
8. π = *ReachAnalysis*($\tilde{\mathcal{N}}$)
9. **if** $IsTrace(\pi)$
10. **then** $\check{S}_c$ = *ExtractUnderApprox* (S_c, π)
11. *Under* = *Under* $\cup \{S_c\}$
12. **else** X = *ConflictAnalysis* (π)
13. **if** $X \subseteq$ *Over*
14. **then return** unreachable
15. **else** $\hat{S}_c$ = *RefineOverApprox* (S_c)
16. *Over* = *BackTrack* (*Over*, X)
17. *Under* = $\mathcal{N} \setminus$ *Over*
18. **return** reachable

Fig. 1. The compositional reachability algorithm

Definition 13 (Under-Approximation): An LTS $\check{S}$ is an under-approximation of the LTS S iff $L(\check{S}) \subseteq L(S)$.

Our approach complements over-approximations with under-approximations. We use the following deduction rule (left) for compositional reasoning (we assume that the property can be expressed as a boolean combination of conditions over state variables of the components):

$$\frac{\begin{array}{c} L(S_1) \subseteq L(\hat{S}_1) \\ \vdots \\ L(S_n) \subseteq L(\hat{S}_n) \\ L(S_c \times \hat{S}_1 \times \ldots \times \hat{S}_n) = \emptyset \end{array}}{L(S_c \times S_1 \times \ldots \times S_n) = \emptyset} \qquad \frac{\begin{array}{c} L(\check{S}_1) \subseteq L(S_1) \\ \vdots \\ L(\check{S}_n) \subseteq L(S_n) \\ L(S_c \times \check{S}_1 \times \ldots \times \check{S}_n) \neq \emptyset \end{array}}{L(S_c \times S_1 \times \ldots \times S_n) \neq \emptyset}$$

The rule on the right is dual to the one on the left, and uses the under-approximations of the components of the network.

Intuitively, the rule on the left is used to prune the search: if we can't find a trace in a network obtained by over-approximating all the components but S_c, then no trace exists in the original network. The rule on the right allows us to conclude: if a trace exists in a network where all components but S_c have been under-approximated, then a trace exists in the original network.

B. Compositional Reachability Algorithm

The compositional reasoning rules presented above are the basis for a search-based algorithm which analyzes the reachability of a network in a compositional way. Intuitively, the algorithm explores the space of networks that can be obtained by over- and/or under-approximating the components of the original network.

The algorithm is presented in Figure 1. The input $\mathcal{N}$ is a network annotated with a set of target locations; the algorithm terminates either returning unreachable when the target is unreachable, or a witness trace when the target is reachable. During the search, each component is associated with an over-approximation, initially constructed by *InitialOverApprox*. The over-approximation is is increasingly tightened with constraints resulting from the analysis of other components.

The algorithm partitions the network between the components in *Under* already analyzed which are under-approximated and the components in *Over* to be analyzed which are over-approximated.

The analysis iterates over the components of the network to be analyzed, stored in the *Over* list. During each iteration, a component S_c is selected by *PickComponent* and removed from *Over*. S_c is checked, in its concrete form, for compatibility with respect to an environment composed by the under-approximations of previously analyzed components, and by the over-approximations of the other components (lines 4-6). The result of the call to *ReachAnalysis*, π, is either a trace, or a proof that the language of $\tilde{\mathcal{N}}$ is empty.

If a trace is returned (lines 8-9), then S_c is added to the *Under* list, and then $\check{S}_c$ = *ExtractUnderApprox* (S_c, π) constructs an under-approximation for S_c, by generalizing the found trace.

If π is not a trace, then *ConflictAnalysis* (π) returns a set of components X responsible for the inconsistency with S_c. If $X \subseteq$ *Over*, this means that S_c turns out to be inconsistent with the over-approximation of some other components, and thus it is possible to conclude that the network has an empty language, according to the composition rules presented above.

If X contains the under-approximation of some component, *RefineOverApprox* learns the reason within S_c for the inconsistency, and the search backtracks to a suitable point, so that some of the processes that had been under-approximated are reinserted in *Over* for reconsideration. Then, the search is resumed. We notice that the learned reason for inconsistency results in a tightening of the over-approximation of S_c. Intuitively, the learned reason encodes a set of traces that prevent S_c to reach the target, and thus are blocked. This has the effect of preventing the under-approximation(s) responsible for inconsistency from being reconsidered in further iterations.

The search terminates successfully when *Over* is empty, i.e. if the last component turns out to be consistent with the under-approximations of all the other components. In such a situation, we can conclude that the language is not empty, and a witness is given by any trace reaching the target and interleaving the candidate traces of each component.

Algorithm *CompositionalAnalysis* has several degrees of freedom in various primitives. Let $\tilde{\mathcal{N}}$ be $S_c \times \prod_{S \in Under} \check{S} \times \prod_{S \in Over} \hat{S}$.

If $L(\tilde{\mathcal{N}}) \neq \emptyset$ and $\check{S}_c$ = *ExtractUnderApprox*(S_c), then $\emptyset \neq L(\check{S}_c) \subseteq L(S_c)$. Thus, the language of the selected under-approximation can range from a trace to the whole language of S_c.

If $L(\tilde{\mathcal{N}}) = \emptyset$ and $\hat{S}_c$ = *RefineOverApprox*(S_c), then (i) $\hat{S}_c$ is an over-approximation of S_c, i.e. $L(S_c) \subseteq L(\hat{S}_c)$, and (ii) $\hat{S}_c$ is sufficiently strong to block the under-approximation

responsible for the inconsistency. That is, if $\tilde{\mathcal{N}}[S_c/\hat{S}_c]$ is $\hat{S}_c \times \prod_{S \in Under} \check{S} \times \prod_{S \in Over} \hat{S}$, then $L(\tilde{\mathcal{N}}[S_c/\hat{S}_c]) = \emptyset$ Thus, the language of the chosen over-approximation can range from $L(S)$ to $\neg L(\prod_{S \in Under} \check{S} \times \prod_{S \in Over} \hat{S})$.

If the sub-routines of Algorithm *CompositionalAnalysis* are correct and terminating, so is the algorithm.

Theorem 3: If the sub-routines of Algorithm *CompositionalAnalysis* are correct, in particular *ReachAnalysis*$(\tilde{\mathcal{N}}) \subseteq L(\tilde{\mathcal{N}})$ iff $L(\tilde{\mathcal{N}}) \neq \emptyset$, $L($*ExtractUnderApprox*$(S_c)) \subseteq L(S_c)$, and $L(S_c) \subseteq L($*RefineOverApprox*$(S_c))$, then Algorithm *CompositionalAnalysis* returns reachable [unreachable] iff $L(\mathcal{N}) \neq \emptyset$ [$L(\mathcal{N}) = \emptyset$].

Theorem 4: If the sub-routines of Algorithm *CompositionalAnalysis* are terminating and a sequence of calls to *RefineOverApprox*(S_c) converges with a finite number of steps to S_c, then Algorithm *CompositionalAnalysis* is terminating.

Running example: In order to illustrate the technique, we consider now the example of network reported in Figure 2. The network has three components (C1, C2 and C3, counterclockwise from the top). C1 and C2 share the labels a and b; C1 and C3 share c; C2 and C3 share d. The components are HA with one or two continuous variables per component. For example, C1 has the continuous variable x which is set to 0 during the transitions $0 \xrightarrow{a} 1$ and $0 \xrightarrow{b} 2$, incremented with $\dot{x} > 10$ while staying in the state 1 and with $\dot{x} \leq 10$ while staying in the state 2, and tested in $1 \rightarrow 3$ and $2 \rightarrow 3$. For simplicity, the local transitions (e.g. $1 \rightarrow 0$ in C1) with non-shared events are not labeled. We denote a network configuration as a tuple of locations: the initial configuration is (0,0,0), i.e. all the components in 0, while the target configuration is (3,3,2). A possible run of the algorithm is the following.

1) The over-approximations created by *InitialOverApprox* contain only the discrete parts of the components and disregard the constraints on the continuous variables.
2) Checking $C1 \times \hat{C}2 \times \hat{C}3$, *ReachAnalysis* finds the C1's run $0 \xrightarrow{a} 1 \xrightarrow{c} 3$. C1 is under-approximated restricting the HA to the states 0, 1, and 3.
3) Checking $\check{C}1 \times C2 \times \hat{C}3$, *ReachAnalysis* finds the C2's run $0 \xrightarrow{d} 1 \xrightarrow{a} 3$. C2 is under-approximated restricting the HA to the states 0, 1, and 3.
4) Checking $\check{C}1 \times \check{C}2 \times C3$, *ReachAnalysis* finds that the language of the approximated network is empty. The reason of the inconsistency is that $\check{C}1$ and $\check{C}2$ require that the time elapsing between d and c is less than 8 while the C3's constraints on z require such time to be greater or equal to 9. Thus the search backtracks to $C2$ and refines $\hat{C}3$ by adding the constraints on z.
5) Re-checking $\check{C}1 \times C2 \times \hat{C}3$, *ReachAnalysis* finds the C2's run $0 \xrightarrow{d} 2 \xrightarrow{a} 3$. C2 is under-approximated restricting the HA to the states 0, 2, and 3.
6) Re-checking $\check{C}1 \times \check{C}2 \times C3$, *ReachAnalysis* finds that the language of the approximated network is not empty and thus Algorithm *CompositionalAnalysis* concludes that the original concrete network admits a run to the target.

Note that if at step 1, *ReachAnalysis* finds the run $0 \xrightarrow{d} 2 \xrightarrow{a} 3$ of C1, this would be inconsistent with the constraints on w of C3. Thus, in such case, at step 4, the search would back-jump to C1.

IV. Framework instantiation

In this section, we discuss a practical instantiation of the general framework of Section III-B. The instantiation depends on three main choices:

1) the state-space of approximations from which we choose the under and over-approximations of the components;
2) the verification problem and the engine with which we analyze the network;
3) the order with which we analyze the processes and we backtrack (which can be fixed, dynamic, based on the topology of the network, etc...).

A. Constraint-based Approximations of Hybrid Automata

The algorithm presented in previous section leaves many degrees of freedom regarding the systems being analyzed, and the formalism used to represent the under- and over-approximations. Here, we present an approach for approximating HAs based on the constraints defining the different parts of the automata. Basically, under-approximations are obtained by removing states and transitions, and/or adding further constraints to initial, invariant, flow, of jump conditions; over-approximations are obtained by adding constraints to initial, invariant, flow, of jump conditions.

Given a HA $H = \langle Q, A, I, T, X, \mu, \iota, \xi, \theta \rangle$, we consider only under-approximations in the form $\check{H} = \langle \check{Q}, A, \check{I}, \check{T}, X, \check{\mu}, \check{\iota}, \check{\xi}, \check{\theta} \rangle$ where

- $\check{Q} \subseteq Q$
- $\check{I} \subseteq I$
- $\check{T} \subseteq T$
- $\mu(q) \subseteq \check{\mu}(q)$ for all $q \in Q$;
- $\iota(q) \subseteq \check{\iota}(q)$ for all $q \in Q$;
- $\xi(q) \subseteq \check{\xi}(q)$ for all $q \in Q$;
- $\theta(q) \subseteq \check{\theta}(q)$ for all $q \in Q$.

Similarly, we consider only over-approximations in the form $\hat{H} = \langle Q, A, I, T, X, \hat{\mu}, \hat{\iota}, \hat{\xi}, \hat{\theta} \rangle$ where the discrete part is the same of H and

- $\hat{\mu}(q) \subseteq \mu(q)$ for all $q \in Q$;
- $\hat{\iota}(q) \subseteq \iota(q)$ for all $q \in Q$;
- $\hat{\xi}(q) \subseteq \xi(q)$ for all $q \in Q$;
- $\hat{\theta}(q) \subseteq \theta(q)$ for all $q \in Q$.

Based on this state-space of approximations, we instantiate the sub-routines *InitialOverApprox*, *ExtractUnderApprox*, and *RefineOverApprox* as follows:

- *InitialOverApprox* returns the over-approximation $\hat{S}$ where initial, invariant, and flow conditions are abstracted away (removed):
  - $\hat{\mu}(q) = \emptyset$ for all $q \in Q$;
  - $\hat{\iota}(q) = \emptyset$ for all $q \in Q$;
  - $\hat{\xi}(q) = \emptyset$ for all $q \in Q$;
  - $\hat{\theta}(q) = \theta(q)$ for all $q \in Q$.

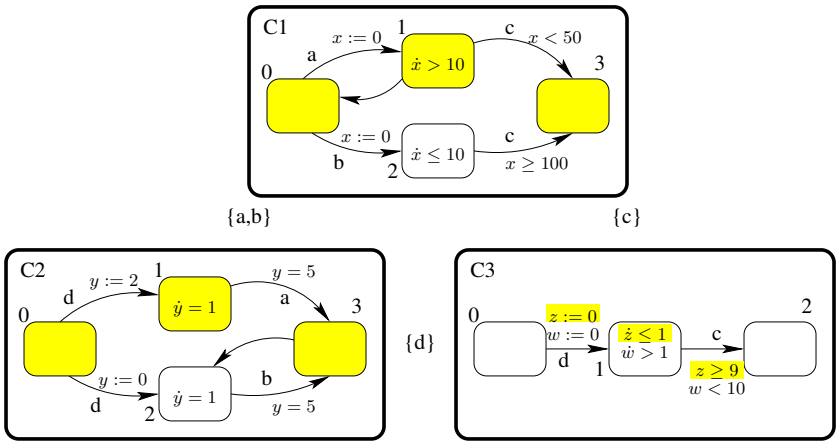


Fig. 2. An example of network.

- Given a trace $\pi \in L(S_c)$, we consider a run σ over π. Let Q_σ be the set of states forming σ. Let A_π be the set of actions forming π. *ExtractUnderApprox* returns the under-approximation $\check{S}_c$ which is obtained from S_c by
  - removing the states which are not visited by σ, i.e. $\check{Q} = Q_\sigma$;
  - removing transition labeled with actions which are not present in π, i.e. $\check{T} = \{\langle q, a, q' \rangle \mid a \in A_\pi\}$.
- Given an approximated network $\tilde{\mathcal{N}}$ whose language is empty, and the over-approximation $\hat{S}_c$ of S_c, *RefineOverApprox* returns a new over-approximation $\hat{S}'_c$ obtained by $\hat{S}_c$ by adding for each $q \in Q$, for each $\star \in \{\mu, \iota, \chi, \theta\}$ a minimal set of constraints in $\star(q)$ to $\hat{\star}(q)$ such that $\tilde{\mathcal{N}}[S_c/\hat{S}_c]$ has an empty language.

B. Compositional bounded reachability analysis with SMT

The compositional analysis of Section III is orthogonal to the addressed reachability problem and the engine used to solve such problem. Here, we focus on the bounded reachability problem of a network of hybrid automata. In particular, we fix a bound k and we check if there exists a path of up to k steps in the network reaching the target, and we discuss its implementation by way of SMT techniques.

1) Bounded reachability analysis: The procedure *ReachAnalysis* uses an SMT solver as back-end engine to solve the satisfiability of formulas encoding reachability problems in the network. Given a network $S_c \times \prod_{S \in Under} \check{S} \times \prod_{S \in Over} \hat{S}$ with up to k steps, the formula encoding the target reachability is the standard:

$$\textsc{Bmc}_k(S_c) \wedge \bigwedge_{S \in Under} \textsc{Bmc}_k(\check{S}) \wedge \bigwedge_{S \in Over} \textsc{Bmc}_k(\hat{S}) \wedge \textsc{Sync}_k \wedge \textsc{Target}_k \quad (1)$$

where SYNC is a formula which forces the synchronization of the runs of the components and varies among different techniques [5].

2) Conflict analysis with unsat cores: ConflictAnalysis modifies the encoding 1 in order to analyze the causes of an inconsistency. The information is used for early termination, back-jumping, and over-approximation refinement.

We add additional variables to flag the sub-formulas forced to be true in the SMT problem. In particular, we build the following formula:

$$\textsc{Bmc}'_k(S_c) \wedge \bigwedge_{S \in Under} (A_{\check{S}} \rightarrow \textsc{Bmc}_k(\check{S})) \wedge (\hat{A} \rightarrow \bigwedge_{S \in Over} \textsc{Bmc}_k(\hat{S})) \wedge \textsc{Sync}_k \wedge \bigvee_{0 \leq i \leq k} \textsc{Target}_i \quad (2)$$

where $\textsc{Bmc}'_k(S_c) = \bigwedge_{\psi \in \textsc{Bmc}_k(S_c)} (A_\psi \rightarrow \psi)$ (note that $\textsc{Bmc}_k(S_c)$ is seen as a set of conjoined formulas).

Thus, the overall set of flags is $\mathcal{F} = \{A \mid A = \hat{A}$ or $A = A_{\check{S}}$, for some $\check{S} \in$ *Under*, or $A = A_\psi$, for some $\psi \in \textsc{Bmc}_k(S_c)\}$. The conflict analysis is performed by considering the formula 2 in conjunction with a subset $\mathcal{X}$ of $\mathcal{F}$. In the following, we analyze different subsets used for different purposes. We will $\mathcal{X}_0$ to denote the set $\{A \mid A = A_\psi$, for some $\psi \in \textsc{Bmc}_k(S_c)\}$ and $\check{\mathcal{X}}$ to denote the set $\{A \mid A = A_{\check{S}}$, for some $\check{S} \in$ *Under*$\}$.

Considering $\mathcal{X} = \mathcal{X}_c \cup \{\hat{A}\}$, we analyze if the process S_c is inconsistent with some over-approximation. In such cases, *ConflictAnalysis* returns *Over* and Algorithm *CompositionalAnalysis* terminates concluding that the target cannot be reached within k steps (*early termination*).

Considering $\mathcal{X}' = \mathcal{X}_0 \cup \{\hat{A}\} \cup \check{\mathcal{X}}'$ for some $\check{\mathcal{X}}' \subseteq \check{\mathcal{X}}$, we check if a set of under-approximations is inconsistent with S_c. This is used to backtrack to the last choice of under-approximation which caused the inconsistency (*back-jumping*).

Considering the set $\mathcal{X}' = \{\hat{A}\} \cup \check{\mathcal{X}} \cup \mathcal{X}'_0$ for different $\mathcal{X}'_0 \subseteq \mathcal{X}_0$, we can analyze the cause of inconsistency within S_c. Given a minimal $\mathcal{X}'$ for which the problem is unsatisfiable, we refine the over-approximation of S_c by adding the constraints corresponding to the formulas in the set $\mathcal{X}'_0$ (*learning*).

The minimal set $\mathcal{X}'$ is found with a linear number of checks providing a set such that the removal of any further element

would result into a satisfiable problem.

3) SMT Incrementality: The two procedures (*ReachAnalysis* and *ConflictAnalysis*) interact with the solver to solve different problems incrementally adding new constraints. This way, the solver keeps the lemmas found in previous checks and speed up the search. This is exploited in *ReachAnalysis* unrolling the systems step by step and interactively checking the reachability of the target. In *ConflictAnalysis*, the incrementality is exploited by querying the solver with different set of flags, while keeping the same instance of the reachability problem.

C. Backtracking order

Finally, the order with which Algorithm *CompositionalAnalysis* picks a components and back-tracks may depend on different heuristic such as the topology of the network. Here, we simply use a fixed order of the components. We assume the order is given by the indexes of the processes $S_1, \ldots, S_n$. At every step of Algorithm *CompositionalAnalysis*, *Under* $= \{S_1, \ldots, S_{j-1}\}$ and *Over* $= \{S_j, \ldots, S_n\}$ for some j, *PickComponent* returns S_j, and we analyze the network $\prod_{1 \leq i \leq j-1} \check{S}_i \times \hat{S}_i \times \prod_{j+1 \leq i \leq n} \hat{S}_n$. When there is a conflict, *BackTrack* chooses the last component in the order $S_1, \ldots, S_{j-1}$ which is in conflict with S_j.

V. Related work

A well-known paradigm of compositional verification is *assume-guarantee reasoning* [15]. In [17], [13], the paradigm has been applied to networks of hybrid systems. However the standard assume-guarantee reasoning focus on guaranteeing that each component satisfies an universal property given over-approximating assumptions on the behavior of the other components. Thus, the approach is not adapt to compositionally construct a model common to all the systems in the network.

Another type of compositional analysis composes the components of a network incrementally producing an observationally equivalent system. In [26], such approach is applied to the reachability analysis of networks of timed systems with discrete time.

Abstraction is a standard technique to tackle the state-space explosion problem. Many techniques adopt over-approximations such as localization reduction [21] or predicate abstraction [14]. BMC [3] can be considered an instance of under-approximation. Constraints-based approximations are also used in the context of HA verification [20]. The work described in [25] proposes an event-order abstraction to verify timed automata. The idea is to analyze the discrete and continuous aspects separately by first finding a discrete path causing an error and then computing a set of timing constraints that make the path realistic. A similar approach is adopted by [6] which exploits a time-stamps semantics of hybrid automata and check the consistency of runs of discrete over-approximations of the components.

Abstraction refinement [10], [11] iteratively refines an over-approximation until either an universal property is proved to be satisfied or a counterexample is given. In *counterexample-guided abstraction refinement* [10], the abstraction is refined according to spurious behaviors created by the over-approximation. Abstraction refinement is applied to hybrid automata in [1]. In [8], abstraction refinement is used to analyze concurrent system, though without a compositional approach.

In [22], [7], the abstraction refinement is applied in combination with under-approximations. In [7], abstraction refinement is applied to both under and over-approximations to check the substitutability of components. In [22], an under-approximated search is performed on the concrete system using predicate abstraction to over-approximate the visited states.

Bounded model checking (without compositional analysis) for hybrid systems using SMT solvers has been investigated in [2], [12], [5]. The proposed compositional analysis is orthogonal to the different engines used to check the approximated networks.

VI. Experiments

A. Implementation

We implemented the algorithm described in Sections III and IV within the setting of NUSMT, a model checker that extends NuSMV2 [9] with SMT techniques. The solver used to check the satisfiability of the formulas is MathSat [4], which provides an incremental interface.

We enabled the tool to take in input a network of symbolically represented hybrid automata. The discrete part of a component is described by means of discrete variables, which may be scalar, integer, or real. The discrete transitions are modeled with formulas as well as the initial, invariant, jump, and flow conditions.

B. Benchmarks

We focus the evaluation on the trade-off provided by the compositional approach between the speed up on a single search given by the approximations and the overhead of performing several searches instead of a monolithic one. As evident from the example of Section III-B, if the components of the network are simple, the benefit of reducing the components will not pay off. Thus, we focus on the scalability of the algorithm with regards to the number of components of a network and the complexity of each component. To this purpose, we consider four academic benchmarks which are scalable in the number of components and we augment them with artificial scalable complexity. In particular, we add to each component a new continuous variable and a scalar variable whose domain is $[1..P]$. Moreover a constraint forces the value of the continuous variable and its flow condition in a different interval for each possible value of the scalar variable. We applied this transformation to the following benchmarks:

- *Star-shape Fischer*: this is the hybrid Fischer algorithm for the mutual exclusion protocol that uses a shared variable to control the access to a critical session.

- *Ring-shape Fischer*: this variant contains a ring of processes where each process shares a variable with its left and right neighbor; the variables are used to access critical sections in mutual exclusion with the neighbors.
- *FDDI Protocol*: this example is a ring topology model based on the system in [27]. It is a set of standards for data transmission on fiber optic lines in a LAN. Each component in the system waits for the signal of previous one to transmit data.
- *Motorcycle*: this example is inspired by the automated highway system from [20]. This system models a sequence of n motorcycles. Each motorcycle i needs to wait the signal from the previous one to move, and it needs to keep the sequence during the parade by synchronizing shared labels with neighbors.

We then evaluated the approach on a more realistic benchmark which is a scalable though simplified version of the ETCS protocol for the interoperability of n trains on a track, as described in [19]. The non-linear equation coming from the flow condition relating the location and the speed of a train is linearized with portrait partitioning [18] with P partitions.

For each benchmark we check reachable targets to verify if the compositional approach manages to speed-up the search by interleaving simpler components. Both methods reach a target with the same number of steps. We show in Table I the length of paths found for each benchmark in function of the number of automata in the network. As for the ETCS protocol, the runs are linear in the partitions used to approximate the non-linear equation.

Benchmark	Path length
Ring-shape Fischer	7
Star-shape Fischer	$3n$
FDDI Protocol	$2n + 2$
Motorcycle	$4n + 3$

TABLE I
LENGTH OF FOUND PATHS IN FUNCTION OF THE NUMBER OF AUTOMATA.

C. Results

The plots in Figure 3 and 4 show, for each benchmark, the difference between the total search times of the monolithic approach and of the compositional approach. In order to show the scalability with respect to the number and the complexity of the components, we use a 3-D plot with, on X and Y axes, the number n of automata in the network and the number P of partitions. On the Z axis, we plot, for each instance, the difference between the total search time of monolithic and the total search time of compositional: thus, a point with a positive value means that the search time of the compositional approach was lower than the search time for the monolithic approach. The instances where both approaches time out are not plotted. The instances where only monolithic [compositional, resp.] times out are highlighted with a triangle [square].

A key finding of the experimental evaluation is that the compositional approach outperforms the monolithic search

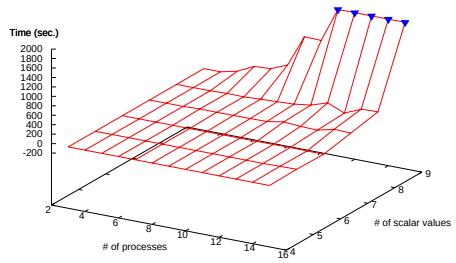


Fig. 3. ETCS Protocol

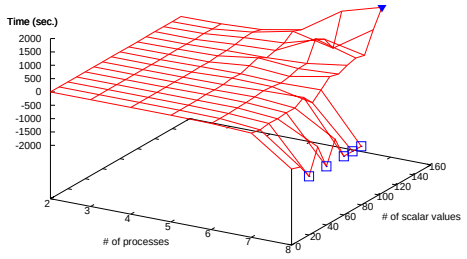
when the complexity of the components increases. This is because when the automata in the network become more complex it is more convenient to perform several searches on an approximated network, rather than a unique search on the concrete network. In some benchmarks, 4(a) and 4(c), we notice that the monolithic approach scales better than compositional when we fix the complexity of the components while increasing their number. In these cases, the reduction given by the approximated components is not enough to counterbalance a higher number of searches.

In all the benchmarks, the adopted over-approximation is strong enough to constrain the search to choose the right under-approximation. This means that the compositional algorithm never backtracks. We also tested our approach using a different, coarser over-approximation, chosen using an heuristic based on the topology of the network: intuitively, only the components topologically close to the concrete component are abstracted as in the previous case, while the others are approximated only by the information learnt during conflic analysis. This may indeed results in backtracking, but without a qualitative change in the plots, and moderate impact on performance. We notice a moderate slow-down in case of backtracking, compensated by moderate speed-up due to the simplified search space. For lack of space, we do not report these results here. The complete set of results, together with the binaries and test cases necessary to reproduce them, are available at http://es.fbk.eu/people/mover/tests/fmcad10/.

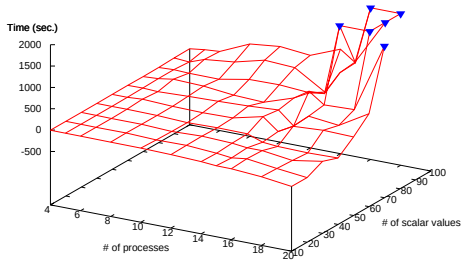
VII. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a compositional approach to reachability of networks of transition systems. The idea is to explore the space of approximations of the components of the network, using both over-approximations and under-approximations. The framework is instantiated to the case of hybrid automata, and implemented by means of SMT techniques. The experimental evaluation shows that the approach is able to scale better than monolithic reachability, and is amenable to several heuristic implementations.

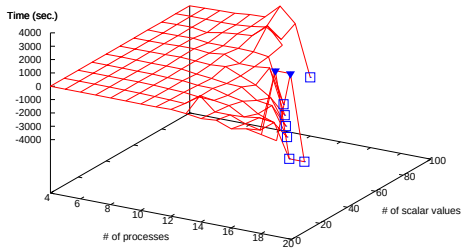
In the future, we plan to extend this research according to the following lines. First, we plan to rely on a more aggressive use of incrementality, representing in a unique formula the whole space of approximations. Second, we will investigate



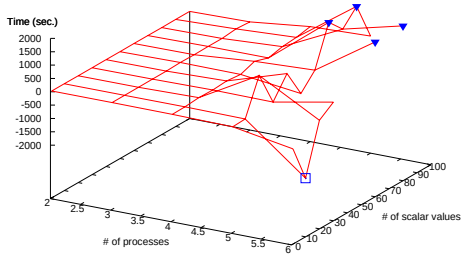
(a) *Star-shape Fischer*



(b) *Ring-shape Fischer*



(c) *FDDI Protocol*



(d) *Motorcycle*

Fig. 4.

the use of a shallow synchronization semantics to exploit the encoding of local runs as done in [5]. Third, we will generalize to the case of unbounded reachability, and we will explore different forms of over and under-approximations. Finally, we plan to investigate the effectiveness of the proposed techniques in the setting of networks with nonlinear solvers.

REFERENCES

[1] R. Alur, T. Dang, and F. Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 5(1):152–199, 2006.

[2] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. *ENTCS*, 119(2):17–32, 2005.

[3] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, pages 193–207, 1999.

[4] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *CAV*, pages 299–303. Springer, 2008.

[5] L. Bu, A. Cimatti, X. Li, S. Mover, and S. Tonetta. Model Checking of Hybrid Systems using Shallow Synchronization. In *FORTE*, 2010.

[6] L. Bu, Y. Li, L. Wang, X. Chen, and X. Li. BACH2: Bounded reachAbility CHecker for Compositional Linear Hybrid Systems. In *DATE*, pages 1512–1517. EDAA, 2010.

[7] S. Chaki, E. Clarke, N. Sharygina, and N. Sinha. Verification of evolving software via component substitutability analysis. *Form. Methods Syst. Des.*, 32(3):235–266, 2008.

[8] Sagar Chaki, Joël Ouaknine, Karen Yorav, and Edmund M. Clarke. Automated Compositional Abstraction Refinement for Concurrent C Programs: A Two-Level Approach. *ENTCS*, 89(3), 2003.

[9] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *CAV*, pages 359–364, 2002.

[10] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV*, pages 154–169, 2000.

[11] S. Das and D.L. Dill. Successive Approximation of Abstract Transition Relations. In *LICS*, pages 51–60, 2001.

[12] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[13] G. Frehse. Compositional verification of hybrid systems with discrete interaction using simulation relations. In *CACSD*, 2004.

[14] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *CAV*, pages 72–83, 1997.

[15] O. Grumberg and D.E. Long. Model Checking and Modular Verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

[16] T.A. Henzinger. The Theory of Hybrid Automata. In *LICS*, pages 278–292. IEEE Computer Society, 1996.

[17] T.A. Henzinger, M. Minea, and V.S. Prabhu. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *HSCC*, pages 275–290, 2001.

[18] T.A. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In *Hybrid Systems*, pages 377–388, 1996.

[19] C. Herde, A. Eggers, M. Fränzle, and T. Teige. Analysis of Hybrid Systems Using HySAT. In *ICONS*, pages 196–201, 2008.

[20] S.K. Jha, B.H. Krogh, J.E. Weimer, and E.M. Clarke. Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction. In *HSCC*, pages 287–300, 2007.

[21] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton University Press, 1994.

[22] C.S. Pasareanu, R. Pelánek, and W. Visser. Predicate Abstraction with Under-approximation Refinement. *CoRR*, abs/cs/0701140, 2007.

[23] E. Plaku, L.E. Kavraki, and M.Y. Vardi. Falsification of LTL Safety Properties in Hybrid Systems. In *TACAS*, pages 368–382, 2009.

[24] A. W. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, November 1997.

[25] U. Shinya. Event order abstraction for parametric real-time system verification. In *EMSOFT*, pages 1–10. ACM, 2008.

[26] Farn Wang. Scalable compositional reachability analysis of real-time concurrent systems. In *RTAS*, pages 182–191, 1996.

[27] J. Zhao, X. Li, T. Zheng, and G. Zheng. Removing Irrelevant Atomic Formulas for Checking Timed Automata Efficiently. In *FORMATS*, pages 34–45, 2003.