Proving and Explaining the Unfeasibility of Message Sequence Charts for Hybrid Systems

Alessandro Cimatti Sergio Mover Stefano Tonetta
FBK-irst, I38050, Trento, Italy
{cimatti,mover,tonettas}@fbk.eu

***Abstract*—Networks of Hybrid Automata are a clean modelling framework for complex systems with discrete and continuous dynamics. Message Sequence Charts (MSCs) are a consolidated language to describe desired behaviors of a network of interacting components. Techniques to analyze the feasibility of an MSC over a given HA network are based on specialized bounded model checking techniques, and focus on efficiently constructing traces of the network that witness the MSC behavior. Unfortunately, these techniques are unable to deal with the "unfeasibility" of the MSC, i.e. that no trace of the network satisfies the MSC.**

In this paper, we tackle the problem of MSC unfeasibility: first, we propose specialized techniques to *prove* that an MSC can not be satisfied by any trace of a given HA network; second, we show how to *explain* why an MSC is unfeasible.

The approach is cast in an SMT-based verification framework, using a local time semantics, where the timescales of the automata in the network are synchronized upon shared events. In order to prove unfeasibility, we generalize k-induction to deal with the structure of the MSC, so that the simple path condition is localized to each fragment of the MSC. The explanations are provided as formulas in the variables representing the time points of the events of the MSCs, and are generated using unsatisfiable core extraction and interpolation. An experimental evaluation demonstrates the effectiveness of the approach in proving unfeasibility, and the adequacy of the automatically generated explanations.

I. Introduction

Complex embedded systems (e.g. control systems for railways, avionics, and space) are made of several interacting components, and feature both discrete and continuous variables. Networks of communicating hybrid automata [18] (HAs) are increasingly used as a formal framework to model and analyze the behavior of such systems: local activities of each component amount to transitions local to each HA; communications and other events that are shared between/visible for various components are modelled as synchronizing transitions of the automata in the network; time elapse is modelled as implicit shared timed transitions.

A fundamental step in the design of these networks is the validation of the models performed by checking if they accept some desired interactions among the components. The language of Message Sequence Charts (MSCs) and its extensions are often used to express scenarios of such interactions. MSCs are especially useful for the end users because of their clarity and graphical content.

The ability to check whether a network of HAs may exhibit behaviors that satisfy a given MSC is an important feature to support user validation. Efficient techniques to analyze the feasibility of an MSC over a given HA network are based on specialized bounded model checking techniques, and focus on efficiently constructing traces of the network that witness the MSC behavior. Unfortunately, these techniques are unable to deal with the unfeasibility of the MSC, i.e. the case where no trace of the network satisfies the MSC.

In this paper, we tackle the problem of MSC unfeasibility, along two main directions: first, we propose specialized techniques to prove that an MSC cannot be satisfied by any trace of a given HA network; second, we show how to explain why an MSC is unfeasible.

In order to *prove unfeasibility*, we propose a specialized algorithm, which generalizes k-induction to deal directly with the structure of the MSC. The search is structured around the events in the MSC, which are used as intermediate "islands". In addition to pre-simplifying the encoding of the fragments of the MSC between events, we apply the simple path condition to each fragment, so that the encoding length of each fragment is no longer increased as soon as we detect that no new states can be reached. The MSC is deemed unfeasible for the network when no fragment can be further extended.

In order to *explain why* an MSC is unfeasible, our approach can generate various information. One is a subset of the MSC that is itself unfeasible for the network, which helps to focus on a subset of the messages, and on the HAs in the network that are involved. Another one is a set of timing conditions over the events in the MSC, which are themselves sufficient to conclude unfeasibility. The explanations are provided as formulas in linear arithmetic, constraining the assignments to the variables representing (some of) the time points of the events of the MSCs. To the best of our knowledge, this is the first work explaining MSC unfeasibility. We remark that here we are trying to provide diagnostic information in case of a *false existential property*, and thus the traditional diagnostics used in model checking for universal properties (e.g. simulation traces) provides no help.

The technical underpinning of this work is the "local time" semantics [6] for HAs, which exploits the fact that automata can be "shallowly synchronized". The intuition is that each automaton can proceed based on its individual "local time scale", unless they perform a synchronizing transition, in which case they must realign their absolute time. The framework allows to reason locally about the simple path conditions for each process, and also to extract more structured explanations, possibly not involving all the processes in the network and

the MSC events.

We implemented the approach and carried out an extensive evaluation, over a wide set of networks and benchmark MSCs. The new approach is able to effectively refute MSCs, significantly outperforming the corresponding approaches based on automata construction, and to provide interesting explanations.

The paper is structured as follows. In Section II, we present some background on networks of HAs, on SMT, and on k-induction. In Section III-A, we describe the language we use to describe the scenarios and the SMT encoding based on their structure. In Section IV, we discuss MSC-direct induction. In Section V, we discuss method to find explanations of unfeasibility. In Section VI, we discuss related work. In Section VII, we experimentally evaluate our approach. In Section VIII, we draw some conclusions.

II. Background

A. Networks of hybrid automata

A *Labelled Transition System* (LTS) is a tuple $\langle Q, A, Q_0, R\rangle$ where Q is the set of states, A is the set of actions/events (also called alphabet), $Q_0 \subseteq Q$ is the set of initial states, $R \subseteq Q \times A \times Q$ is the set of labeled transitions.

A *trace* is a sequence of events $w = a_1, \ldots, a_k \in A^*$. Given $A' \subseteq A$, the projection $w_{|A'}$ of w on A' is the subtrace of w obtained by removing all events in w that are not in A'. A *path* π of S over the trace $w = a_1, \ldots, a_k \in A^*$ is a sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} q_k$ such that $q_0 \in Q_0$ and, $\langle q_{i-1}, a_i, q_i\rangle \in R$ for all i such that $1 \leq i \leq k$. We say that π accepts w.

The *parallel composition* $S_1||S_2$ of two LTSs $S_1 = \langle Q_1, A_1, Q_{01}, R_1\rangle$ and $S_2 = \langle Q_2, A_2, Q_{02}, R_2\rangle$ is the LTS $\langle Q_1 \times Q_2, A_1 \cup A_2, Q_{01} \times Q_{02}, R\rangle$ where:
$R := \{\langle\langle q_1, q_2\rangle, a, \langle q_1', q_2'\rangle\rangle \,|\langle q_1, a, q_1'\rangle \in R_1, \langle q_2, a, q_2'\rangle \in R_2\}$
$\cup\{\langle\langle q_1, q_2\rangle, a, \langle q_1', q_2\rangle\rangle \mid \langle q_1, a, q_1'\rangle \in R_1, a \notin A_2\}$
$\cup\{\langle\langle q_1, q_2\rangle, a, \langle q_1, q_2'\rangle\rangle \mid \langle q_2, a, q_2'\rangle \in R_2, a \notin A_1\}$.

The parallel composition of two or more LTSs $S_1||\ldots||S_n$ is also called a *network*. If an event is shared by two or more components, we say that the event is a synchronization event; otherwise, we say that the event is local. We denote with τ_i the set of local events of the i-th component.

Given a network $\mathcal{N}$ and a state $q \in Q_1 \times \ldots \times Q_n$, the *reachability problem* is the problem of checking if there is a path $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} q_k$ of S with $q_k = q$.

Hybrid Automata (HAs) [18] enrich the discrete states and transitions of LTSs with continuous variables and further conditions that constrain how these variables continuously evolve within a discrete state. In particular, a HA is a tuple $\langle Q, A, Q_0, R, X, \mu, \iota, \xi, \theta\rangle$ where:

- Q is the set of states,
- A is the set of events,
- $Q_0 \subseteq Q$ is the set of initial states,
- $R \subseteq Q \times A \times Q$ 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,
- $\theta : R \to P(X, X')$ is the jump condition,

where X' represent the value of variables X after a discrete transition, $\dot{X}$ represent the derivative of variables X during a continuous evolution, and P represents the set of predicates over the specified variables.

A Linear HA (LHA) is an HA where all the conditions are Boolean combinations of linear inequalities and the flow conditions contain variables in $\dot{X}$ only. We assume also that the invariant conditions of a LHA are conjunctions of inequalities.

A *network* $\mathcal{H}$ of HAs is the parallel composition of two or more HAs. We consider the local-time semantics, which is equivalent to the standard global-time semantics of [18], but instead of synchronizing the components on a shared timed event, it enriches all shared events with time-stamps, introduces local timed events, and synchronizes the components on shared events forcing the time-stamps to be equal [6], [10].

In the following, we consider a network $\mathcal{H} = H_1||\ldots||H_n$ of HAs with $H_i = \langle Q_i, A_i, Q_{0i}, R_i, X_i, \mu_i, \iota_i, \xi_i, \theta_i\rangle$ such that, for all $1 \leq i < j \leq n$, $X_i \cap X_j = \emptyset$ (i.e. the set of continuous variables of the hybrid automata are disjoint).

The *local-time semantics* (or time-stamps semantics) of $\mathcal{H}$ is the network of LTSs $\mathcal{N}_{\text{LocTime}}(\mathcal{H}) = S_1||\ldots||S_n$ with $S_i = \langle Q_i', A_i', Q_{0i}', R_i'\rangle$ where:

- $Q_i' = \{\langle q, \overline{x}, t\rangle \mid q \in Q_i, \overline{x} \in \mathbb{R}^{|X_i|}, t \in \mathbb{R}_{\geq 0}\}$,
- $A_i' = \{\langle a, t\rangle \mid a \in A_i, t \in \mathbb{R}_{\geq 0}\} \cup \{\text{Time}_i\}$,
- $Q_{0i}' = \{\langle q, \overline{x}, 0\rangle \mid q \in Q_{0i}, \overline{x} \in \iota_i(q)\}$,
- $R_i' = \{\langle\langle q, \overline{x}, t\rangle, \langle a, t\rangle, \langle q', \overline{x}', t\rangle\rangle \mid \langle q, a, q'\rangle \in R_i, \langle \overline{x}, \overline{x}'\rangle \in \theta_i(q, a, q'), \overline{x} \in \xi_i(q), \overline{x}' \in \xi_i(q')\} \cup \{\langle\langle q, \overline{x}, t\rangle, \text{Time}_i, \langle q, \overline{x}', t'\rangle\rangle \mid$ there exists f satisfying $\mu_i(q)$ s.t. $f(t) = \overline{x}, f(t') = \overline{x}', f(\epsilon) \in \xi_i(q), \epsilon \in [t, t'], t \leq t'\}$.

The definition of the local-time semantics is such that the set of actions of each LTS contains a local timed event Time_i and couples containing a discrete action and a time-stamp (i.e. the amount of time elapsed in the automaton). Thus, each automaton performs the time transition locally, changing its local time-stamp. When two automata synchronize on $\langle a, t\rangle$ they agree on the action a and on the time-stamp t. Instead, in the global-time semantics, all the automata are forced to synchronize on the time transition $\langle \text{Time}, \delta\rangle$, agreeing on the time elapsed during the transition (δ variable).

B. SMT encoding of hybrid automata

As described in [18], LHAs can be analyzed with symbolic techniques. Let us consider a network $\mathcal{H} = H_1||\ldots||H_n$ of LHAs whose semantics is given by the network of LTSs $S_1||\ldots||S_n$ where $S_i = \langle Q_i, A_i, Q_{i0}, R_i\rangle$. The states Q_i can be represented by a set V_i of symbolic variables. The events of A_i can be represented by a set of symbolic variables W_i. Sets of states are represented with formulas over V_i, while sets of transitions are represented with formulas over V_i, W_i, and V_i', which are the next values of V_i. In particular, it is possible to define a formula $I_i(V_i)$ that represents the initial states and a formula T_i that represents the transitions of H_i.

The details of the encoding we use can be found in [8]. Here, we just notice that we use a scalar input variable ε to represent the events of H_i adding two distinguished values, namely T and S, to represent a timed transition and stuttering, respectively. When stuttering, the system does not change any variable. Moreover, when using the local-time semantics, the variable t_i represents the local time of H_i and is also used as time-stamp of the events (thus, to ensure that shared events happen at the same time).

As standard in Bounded Model Checking, given an integer k, we can build a formula whose models correspond to all paths of length k of the represented LTS S. The formula introduces $k+1$ copies of every variable in the encoding of the automata. Given a formula ϕ, we denote with ϕ^i the result of substituting the current and next variables of ϕ with their i-th and $(i+1)$-th copy, respectively. The paths of S of length k can be encoded into the formula $\mathit{path}(k) := I^0 \wedge \bigwedge_{0 \leq i < k} T^i$.

A typical optimization used in BMC for timed and hybrid systems is to force the alternation of timed and discrete transitions [1], [4].

Most of modern solvers, both for SAT and SMT, have an *incremental* interface such that, if a problem is fed to the solver incrementally, the solver can first tackle smaller parts of the problem and then pass to large parts managing to reuse the lemmas discovered during the previous searches.

C. K-induction

K-induction [32] is a technique that proves that if a set of states is not reachable in k steps, then it is not reachable at all. On the lines of the induction principle, it consists of a base step, which solves the bounded reachability problem with a given bound k of steps, and an inductive step, which concludes that k is sufficient to solve the (unbounded) reachability problem. The idea of the inductive step is to check either if the initial states cannot reach new (non-visited) states in $k+1$ steps, or if the target set of states cannot be reached in $k+1$ steps (hereafter, we will consider only the first condition). These checks can be solved by means of satisfiability.

The formula $\mathit{simple}(k) := \bigwedge_{0 \leq i < j \leq k} \neg \bigwedge_{v \in V} v^i = v^j$ can be used to strengthen the path encoding to represent only simple (loop-free) paths. If the formula $\mathit{kind}(k) := I(V^0) \wedge \pi(k+1) \wedge \mathit{simple}(k+1)$ is unsatisfiable, then there is no initial simple path with more than k states. Thus, if, for all $i \leq k$, $\mathit{path}(k) \wedge \mathit{target}^k$ is unsatisfiable and $\mathit{kind}(k)$ is unsatisfiable as well, then *target* is not reachable.

If the target is not reachable in a finite-state LTS, there is a k for which the above conditions are unsatisfiable. In hybrid systems, it is very common that the LTSs contain infinite paths, typically with monotonically increasing variables (such as the local time) and, therefore, it is difficult to apply k-induction.

In [33], k-induction has been integrated with predicate abstraction [16] to deal with infinite-state systems. Typically, an abstraction defines an equivalence relation EQ_α among the the concrete states that are not distinguished by the abstraction. As for predicate abstraction, given a certain set $\mathbb{P}$ of predicates over the variables V, the equivalence relation is defined as $EQ_{\mathbb{P}}(V, \overline{V}) := \bigwedge_{P \in \mathbb{P}} P(V) \leftrightarrow P(\overline{V})$.

Abstract k-induction embeds the definition of the predicate abstraction in the encoding of the path. In particular, the formula $\mathit{path}_\alpha(k) := \bigwedge_{1 \leq h < k} (T(V^{h-1}, \overline{V^h}) \wedge EQ_\alpha(\overline{V^h}, V^h)) \wedge T(V^{k-1}, V^k)$ is satisfiable iff there exist a path of k steps in the abstract state space. The formula $\mathit{simple}_\alpha(k)$ is defined as $\mathit{simple}_\alpha(k) := \bigwedge_{0 \leq i < j \leq k} \neg EQ_\alpha(V^i, V^j)$. The formula $\mathit{path}_\alpha(k) \wedge \mathit{simple}_\alpha(k)$ is satisfiable iff there exists a simple path of length k in the abstract state space. Finally, the formula kind_α, defined as $\mathit{kind}_\alpha(k) := I(\overline{V^0}) \wedge EQ_\alpha(\overline{V^0}, V^0) \wedge \mathit{path}_\alpha(k) \wedge \mathit{simple}_\alpha(k)$, is satisfiable iff there exists an initial simple path of length k.

Similarly to the concrete case, if, for all $i \leq k$, $\mathit{path}_\alpha(k) \wedge EQ_\alpha(V^k, \overline{V^k}) \wedge \mathit{target}^k$ is unsatisfiable and $\mathit{kind}_\alpha(k)$ is unsatisfiable as well, then *target* is not reachable in the abstraction (and therefore also in the concrete state space).

III. MSC Feasibility

A. Constrained Message Sequence Charts

A Message Sequence Chart (MSC) [20] defines a single (partial-order) interaction of the components of a network $\mathcal{N}$. MSCs have been extended in several ways. We consider here a particular variant, enriched with additional constraints, which turns out to be very useful and easy to handle with the SMT-based approach.

An MSC m is associated with a set of events $A_m \subseteq A_\mathcal{N}$, subset of the events of the network. We assume that A_m contains all and only the shared events of the network ($A_m = \bigcup_{1 \leq i < j \leq n} A_i \cap A_j$). In particular, in the case of hybrid automata the timed events are not part of A_m.

The MSC defines a sequence of events for every component S of the network, called instance of S. An instance σ for the LTS S is a sequence $a_1; \ldots; a_l \in (A_m \cap A_S)^*$ of events of S. S accepts the instance ($S \models \sigma$) iff there exists a trace w accepted by S such that the sub-sequence of events in A_m is equal to σ ($w_{|A_m} = \sigma$). In other words, S accepts the instance iff there exists a path π of S over a trace compatible with the instance σ. In such cases, we say that $\pi \models \sigma$.

We denote the j-th event a_j of the instance σ_i with $\sigma_i[j]$, the number l of events in σ_i with $|\sigma_i|$, the local segment between the event $\sigma_i[j]$ and $\sigma_i[j+1]$ of σ_i with $\mathit{lsg}(\sigma_i[j])$, where the first local segment before a_1 is $\mathit{lsg}(\sigma_i[0])$ and the final local segment after $a_{|\sigma_i|}$ is $\mathit{lsg}(\sigma_i[|\sigma_i|])$.

If $\pi \models \sigma$, π must be in the form $q_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_{h_1} \xrightarrow{\sigma[1]} q_{h_1+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_{h_{|\sigma|}} \xrightarrow{\sigma[|\sigma|]} q_{h_{|\sigma|+1}} \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_{h_{|\sigma+1|}}$, where $q_h \in Q$ and τ are local events of S. We denote the sub-sequences of the path π in which it is split by σ as follows:

- $pre_j(\pi) = q_{h_j}$, it is the source state of the transition labeled with $\sigma[j]$ in π.
- $post_j(\pi) = q_{h_j+1}$, it is the destination state of the transition labeled with $\sigma[j]$ in π.
- $loc_j(\pi) = q_{h_j+1}; \ldots; q_{h_{j+1}}$, it is the sequence of states between the j-th and the $j+1$-th shared events, where we denoted 0 with h_0.

An MSC is the parallel composition $\sigma_1||\dots||\sigma_n$ where σ_i is an instance of S_i. The network $\mathcal{N}$ of LTSs accepts the MSC m ($\mathcal{N} \models m$) iff there exists a trace w accepted by $\mathcal{N}$ such that, for every S_i, the sub-sequence of events in $A_m \cap A_{S_i}$ is equal to σ_i ($w_{|(A_m \cap A_{S_i})} = \sigma_i$). In other words, $\mathcal{N}$ accepts the instance iff there exists a path of $\mathcal{N}$ over a trace compatible with every instance of the MSC. If $\mathcal{H}$ is a network of HAs, then we say that $\mathcal{H} \models m$ iff $\mathcal{N}_{\text{LocTime}}(\mathcal{H}) \models m$.

We define a Constrained MSC (CMSC) as a pair $\langle m, \phi \rangle$ where m is an MSC $\sigma_1||\dots||\sigma_n$ and ϕ is a formula over the variables $v_i[j]$ with $1 \leq i \leq n$ and $1 \leq j \leq |\sigma_i|$, where $v_i[j]$ represents the value of the variable v of the i-th component at the time of the j-th event $\sigma_i[j]$ of σ_i. $\mathcal{N} \models \langle m, \phi \rangle$ iff there exists a path $\pi = \pi_1||\dots||\pi_n$ such that $\pi_i \models \sigma_i$ and the assignments of $pre_j(\pi_i)$ to $v_i[j]$ satisfy ϕ.

The model checking problem for a CMSC $\langle m, \phi \rangle$ is the problem of checking if a network satisfies a CMSC. The classical approach is based on the construction of a monitor (or a network of monitors) that, composed with $\mathcal{N}$, forces $\mathcal{N}$ to follow only paths that satisfy the MSC.

An MSC $\sigma_1||\dots||\sigma_n$ is consistent iff for every pair of instances σ_i and σ_j the projection on the common alphabet is the same, i.e., if $A = A_i \cap A_j$, $\sigma_{i|A} = \sigma_{j|A}$. Henceforth, we assume that the MSCs are consistent. The check of consistency is trivial and can be done syntactically with a simple traversal of the MSC's structure.

B. Scenario-driven encoding

The drawbacks of the traditional SMT-based encoding is that it cannot exploit the sequence of messages prescribed by the MSC in order to simplify the search because of the uncertainty on the number of local steps between two events. We encode the path of each automaton independently, exploiting the local time semantics, and then we add constraints that force shared events to happen at the same time, as in *shallow synchronization* [8]. Moreover, we fix the steps corresponding to the shared events and we parametrize the encoding of the local steps with a maximum number of transitions.

We extend the encoding presented in [10] with different numbers of steps for different local segments of the MSC.

Let us consider a network $\mathcal{H} = H_1||\dots||H_n$ of LHAs and the encoding $\langle V_i, W_i, I_i, T_i \rangle$ representing the LHA H_i, for $1 \leq i \leq n$, in the local-time semantics. We denote with $T_{i|\phi}$ the transition condition restricted to the condition ϕ, i.e., $T_{i|\phi} = T_i \wedge \phi$. We abbreviate $T_{i|\varepsilon=a}$ with $T_{i|a}$ and $T_{i|\varepsilon \in \tau_i \cup \{\text{S}\}}$ with $T_{i|\tau}$ (notice that τ_i, the set of local actions, contains also the timed event T).

We associate a bound $k_i[j]$ to the j-th segment $lsg(\sigma_i[j])$ of the i-th instance. $k_i[j]$ is used to limit the number of transitions in the local path $loc_j(\pi)$ of a path π satisfying the instance σ_i. We use $\overline{k}_i$ to denote $\langle k_i[0], \dots, k_i[h_i] \rangle$ and $\overline{k}$ to denote $\langle \overline{k}_1, \dots, \overline{k}_n \rangle$.

Note that the event $\sigma_i[j]$ is preceded by $\sum_{v=0}^{j-1} k_i[v] + j - 1$ transitions consisting of local transitions ($\sum_{v=0}^{j-1} k_i[v]$) and shared events ($j-1$). $idx_i[j]$ defines the index used to encode the event $\sigma_i[j]$ as $idx_i[j] := \sum_{v=0}^{j-1} k_i[v] + j - 1$.

The following encoding represents all paths of the network compatible with the MSC where the local transitions of the j-th segment of the i-th instance have been unrolled up to $k_i[j]$ times (note that the "up to" is due to the ability of stuttering):

$$
\begin{aligned}
enc(m, \overline{k}) \quad &:= \quad \bigwedge_{1 \leq i \leq n} enc(\sigma_i, \overline{k}_i) \wedge \\
&\qquad \bigwedge_{1 \leq j < i \leq n} sync(\sigma_j, \sigma_i) \wedge (t_j^{\sum_{v=0}^{|\sigma_j|} k_j[v]} = t_i^{\sum_{v=0}^{|\sigma_i|} k_i[v]}) \\
enc(\sigma_i, \overline{k}_i)) \quad &:= \quad I_i^0 \wedge \bigwedge_{1 \leq h \leq k_i^0} T_{i|\tau}^{h-1} \wedge \\
&\qquad \bigwedge_{1 \leq j \leq |\sigma_i|} (T_{i|a_j}^{idx_i[j]} \wedge \bigwedge_{1 \leq h \leq k_i[j]} T_{i|\tau}^{idx_i[j]+h}) \\
sync(\sigma_j, \sigma_i) \quad &:= \quad \bigwedge_{1 \leq z \leq |\sigma_{j|_A}| = |\sigma_{i|_A}|} t_i^{idx_i[f_i^{ij}(z)]} = t_j^{idx_j[f_j^{ij}(z)]}
\end{aligned}
$$

where $A = A_i \cap A_j$ and the function f_i^{ij} maps the z-th event a_z shared between σ_i and σ_j to the index of a_z in σ_i. More, specifically, if $\sigma_{j|_A} = \sigma_{i|_A} = a_1; \dots a_l$, then $f_i^{ij}, f_j^{ij} : \mathbb{N} \to \mathbb{N}$ are such that $a_z = \sigma_i(f_i^{ij}(z)) = \sigma_j(f_j^{ij}(z))$, for $1 \leq z \leq l$.

Intuitively, $enc(m, \overline{k})$ encodes the unrolling of each component according to its instance and guarantees that the different unrollings have the same time for every occurrence of a shared event and the same final time.

In order to encode the paths that satisfy a CMSC we have just to conjoin the additional constraints:

$$
enc(\langle m, \phi \rangle, \overline{k}) \quad := \quad enc(m, \overline{k}) \wedge \phi[v_i^{idx_i[j]} / v_i[j]]
$$

where for all the instances i, $1 \leq i \leq n$, and all events j, $1 \leq j \leq |\sigma_i|$, we substitute $v_i[j]$ in ϕ with the timed variable $v_i^{idx_i[j]}$.

Theorem 1: If $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable then $\mathcal{H} \models \langle m, \phi \rangle$. Vice versa, if $\mathcal{H} \models \langle m, \phi \rangle$, then there exists integers $\overline{k}$ such that $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable.

IV. Scenario-driven induction

In this section, we describe how the structure of the MSC can be exploited to tailor k-induction to prove the unfeasibility of the scenario. For the base case, we use the encoding of [10]. For the inductive step, we apply the simple path condition to each segment of the scenario and prove that such partitioned simple-path condition is equivalent to the path condition applied to composition of the network and the scenario monitor. The use of different local bounds as presented in Section III-B allows k-induction to stop the unrolling of the local path at different depths according to the local structure of the component at the considered segment.

A. Partitioned simple-path condition

Our goal is to find an inductive condition $kind(\langle m, \phi \rangle, \overline{k})$ such that, in the finite-state case, $\mathcal{N} \not\models \langle m, \phi \rangle$ if and only if there exist $\overline{k}$ such that $enc(\langle m, \phi \rangle, \overline{k})$ and $kind(\langle m, \phi \rangle, \overline{k})$ are unsatisfiable. In the hybrid case, we would like that the "if" condition still holds, while the "only if" condition

should hold when the corresponding inductive condition for the composition of the network with the MSC monitor holds (relatively complete). The difficulties are that:

- the projection of a simple path on a component may be not a simple path;
- if a simple path is the concatenation or the parallel composition of two paths, these may be not the longest simple paths of their segments.

The CMSC $\langle m, \phi \rangle$ defines a partial order $<_m$ among the segments of m defined as the reflexive and transitive closure of the smallest relation such that:

- $lsg(\sigma_i[j]) <_m lsg(\sigma_i[j'])$ if $0 \leq j < j' \leq h_i$;
- $lsg(\sigma_i[j]) <_m lsg(\sigma_{i'}[j'])$ if there exists $lsg(\sigma_{i''}[j''])$ such that there is a synchronization between $\sigma_i[j]$ and $\sigma_{i''}[j'']$ and $lsg(\sigma_{i''}[j'']) <_m lsg(\sigma_{i'}[j'])$.

Given a CMSC $\langle m, \phi \rangle$ and the local path $lsg(\sigma_i[j])$ we define the partial CMSC $\langle \overline{m}_i[j], \overline{\phi}_i[j] \rangle$ where:

- $\overline{m}_i[j] = \overline{\sigma}_1 || \ldots || \overline{\sigma}_n$ such that for all $1 \leq v \leq n$, $|\overline{\sigma}_v| \leq |\sigma_v|$ and for all $1 \leq z \leq |\overline{\sigma}_v|$ $\overline{\sigma}_v[z] = \sigma_v[z]$ and $lsg(\overline{\sigma}_v[z]) <_m lsg(\sigma_i[j])$ or $lsg(\overline{\sigma}_v[z]) = lsg(\sigma_i[j])$, while for all $|\overline{\sigma}_v| < z \leq |\sigma_v|$ $lsg(\sigma_v[z]) \not<_m lsg(\sigma_i[j])$.
- $\overline{\phi}_i[j]$ contains only the constraints of ϕ which are over variables in $\overline{m}_i[j]$.

We define the local simple path condition as follows:

$$\begin{aligned} kind_i[j] &:= enc(\langle \overline{m}_i[j], \overline{\phi}_i[j] \rangle, \overline{k}) \wedge simple_i[j] \\ simple_i[j] &:= \bigwedge_{1 \leq h, z \leq k_i[j]} s_i^{idx_i[j]+h} \neq s_i^{idx_i[j]+z} \end{aligned}$$

Theorem 2: If there exist $\overline{k}$ s.t. $enc(\langle m, \phi \rangle, \overline{k})$ is unsatisfiable and, for all i, j, $kind_i[j]$ is unsatisfiable, then $\mathcal{N} \not\models m$.

In order to check if k-induction holds incrementally, we visit the MSC m according to the partial order $<_m$. We incrementally apply the partitioned simple path condition to the local segments of m. The incremental checks exploit the standard Push/Pop/Assert incremental interface of the solver.

B. K-induction for hybrid systems

1) Alternation of timed and discrete transitions: The alternation of timed and discrete transitions has been proposed in different works to optimize the search of BMC for timed and hybrid systems [1], [4]. With k-induction, the alternation is fundamental to allow a concrete search to close. In fact, without forcing the alternation, the system will likely have infinite loop-free paths where timed transitions change some continuous variables infinitely often.

In order to enhance k-induction with alternation, the following points must be taken into account:

- since consecutive discrete transitions are possible, the timed transition must permit the elapsed time to be zero; therefore, the loop-free condition of k-induction must be relaxed in order to allow self loops with a timed transition with no elapsed time;
- the scenario-based encoding of the bounded model checking problem exploits stutter transitions in order to encode paths with up to k steps (instead of exactly k steps); the stuttering makes the alternation ineffective because it allows infinite loop-free paths alternating timed and stutter transitions; therefore, it is fundamental to avoid stuttering when considering the simple path condition.

2) Enabling a partitioned abstraction: The structure of local transitions between two shared events is often simple and without loops. In these cases, the alternation without stuttering allows k-induction to prove the unfeasibility of scenarios. If instead there are loops in the local structure, they may correspond to infinite loop-free paths. In order to prove the unfeasibility of scenarios also in these cases, we combine k-induction with predicate abstraction as in [33].

We can associate to different segments of the MSC different abstractions of the local transition relation. This way, we can obtain a fined-grained abstraction which abstract away the continuous components only where necessary.

V. Unfeasibility explanation

We identify the following types of explanations to understand the reasons of the unfeasibility of the CMSC $\langle m, \phi \rangle$:

1) which parts of the CMSC cannot be executed by the network;
2) why the paths of the network consistent with m cannot satisfy ϕ;
3) why the paths of a component consistent with the corresponding instance of m are inconsistent with the rest of the CMSC.

We answer these questions by exploiting both unsat cores and interpolation. The unsatisfiable core for an unsatisfiable formula ϕ is a formula ψ iff ψ is unsatisfiable and $\phi = \psi \wedge \psi'$, for a (possibly empty) formula ψ'. Given two formulas A and B, with $A \wedge B \models \bot$, the Craig interpolant of $A \wedge B$ is a formula I such that $\models A \rightarrow I$, $B \wedge I \models \bot$, and which contains only variables common to A and B. Intuitively, the interpolant is an over-approximation of A "guided" by B.

In particular, after reaching the maximum bound in every local segment of the CMSC, we can build the proof of unsatisfiability of the BMC problem with such bounds. The unsat core extracted from the proof contains a subset of the unrolling of the components along the MSC and a (possibly empty) subset of the CMSC constraints which are incompatible. Since the local paths, events, and constraints are asserted in different conjuncts of the encoding, the unsat core is fine-grained enough to distinguish them.

By partitioning the encoding into the constraints obtained by unrolling the network (A) and the constraints of the CMSC (B), we can compute an interpolant of their unsatisfiability. This way, we obtain a formula over the variables at the time of the events implied by the network executing the MSC and inconsistent with the constraints of the CMSC. Note that if the interpolant is false, we can deduce that the constraints are not responsible of the unfeasibility and that the unrolling of the network is inconsistent by itself.

Finally, by partitioning the encoding into the unrolling of one component along its instance (A) and the rest (B, i.e.,

other components and constraints), the interpolation produces a formula over the variables at the time of the events implied by the component executing the instance and inconsistent with the other components or with the constraints of the CMSC. Note that if the interpolant is true, it means that the component does not play a role in the unfeasibility. On the contrary, if the interpolant is false, the component does not have a path compatible with the instance.

Note that, when the abstraction is used to prove the unfeasibility of the scenario, the explanations based on unsat core and interpolation are still valid.

VI. Related Work

MSCs [20] are a basic building block to describe the interactions among components. Several works, such as High-Level Message Sequence Charts [26] and Live Sequence Charts (LSC) [12], extend the language of the MSCs increasing their expressive power. We consider a basic version of MSCs which describes a single (partial-order) composition of sequences of events, augmented with additional constraints [2], [5]. We consider a trace-based semantics for the MSC, where the MSC predicates over the observable events of a system [23], [24]. While several works use MSCs to describe the entire system [3], [28], we instead use the MSC as a specification language.

A common approach to deal with the verification of MSC specifications consists in translating the scenario into automata or temporal logic formulas. LSCs are translated into timed automata in the UPPAAL model checker [25], while in [22] the authors propose a translation from charts with timing constraints and synchronous events to Timed Büchi Automata. These works deal with expressive specification languages but they do not exploit the structure of the scenario. Moreover, in case of unfeasibility, these techniques do not provide explanations that narrow the events of the scenario or that gives meaningful information about a specific component.

The approach which translates the MSC into an automaton reduces the feasibility problem of the MSC to a reachability problem. Thus, the works on Bounded Model Checking (BMC) for hybrid systems [1], [4], [8], [14], [15], [34] can be used to solve the feasibility problem. The BMC encodings for hybrid automata can be further optimized exploiting the step semantics [17], [21], which allows independent transitions of different automata to be executed in parallel. However, BMC is unable to prove the unfeasibility of the MSC. When we encode the MSC into an automaton the unfeasibility problem can be solved using unbounded model checking techniques, such as k-induction [32]. K-induction is complete for finite state systems, but it was applied also to infinite state systems in [13], [29], [33]. In [13] the authors use k-induction to verify timed and hybrid automata and they generalize the simple path condition to simulation relations. K-induction is combined with predicate abstraction [16] in [33]. These works are not tailored to the problem of deciding the unfeasibility of a scenario and do not provide explanations in the case of unsatisfiability.

In [10] we propose a Bounded Model Checking encoding tailored to check the feasibility of a scenario in a network of hybrid automata. This approach turns out to be very efficient in dealing with complex scenarios, since it exploit the local-time semantics [6] in order to partition the encoding with respect to the MSC structure. However, the approach is unable to prove the unfeasibility of the scenario. We extend that work in order to prove the unfeasibility of a scenario and to provide meaningful explanations of unfeasibility.

Unsat cores and interpolation are often used to explain and generalize the source of unsatisfiability. Unsat cores are typically subsets of the conjuncts forming the unsatisfiable formula. However, other forms are possible, especially in the context of temporal unsatisfiability [31]. Interpolation for temporal properties is proposed in [30] as a theoretical framework for analyzing vacuity for discrete systems; the practical implications are not addressed in depth. In [31], it is suggested that k-induction can be used to find a k for which the BMC encoding of a temporal formula yields its unsatisfiability and that the unsat core contains the relevant parts of the formula that cause the unsatisfiability. However, mapping the BMC unsat core back to the original problem is not always easy. We achieve this by exploiting the scenario-based encoding that respects the structure of the scenario.

VII. Experimental evaluation

The techniques discussed in the previous sections were implemented in an extension of the NuSMV model checker [9], which is able to deal with networks of HAs, formalized in the HYDI language [11]. The NuSMV extension features an SMT-based approach to the verification of hybrid systems, and is tightly integrated with MathSAT [7], a state-of-the-art, full-fledged Satisfiability-Modulo-Theory solver (SMT). MathSAT provides the functionalities of incremental reasoning, unsatisfiable core extraction, and interpolation, which are used for bounded model checking, inductive reasoning, and explanation extraction.

In the experimental evaluation, we used the following benchmarks: the *Distributed Controller* [19], the *Audio Protocol* proposed in [19], the *Nuclear Reactor* [35], a hybrid version of the Fischer mutual exclusion protocol, and the *Electronic Height Control System (EHC)* described in [27]. All the test cases, the executable and the results of the evaluation are available at http://es.fbk.eu/people/mover/tests/FMCAD11/.

A. Scenario-driven Induction vs K-Induction

First, we compared the scenario-based induction with k-induction applied to the monolithic encoding of the network of HAs and the automata translated from the MSC.

The monolithic encoding is obtained composing the network with the automata obtained from the MSC. The construction of the monitor automata is described in details in [10]. In particular, we rely on the "distributed" monitor automata, where we build a monitor for each instance of the MSC, and the step semantics, which enables multiple transition to be executed in parallel. The combination of both approaches demonstrated to be the most efficient among the different automata construction and encoding presented in [10].

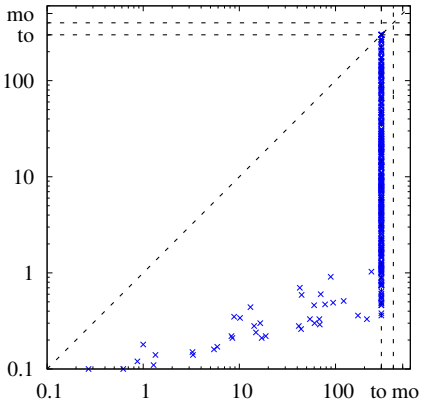

Fig. 1. Run times (sec.): monolithic induction (x axes) vs. scenario-induction (y axes)

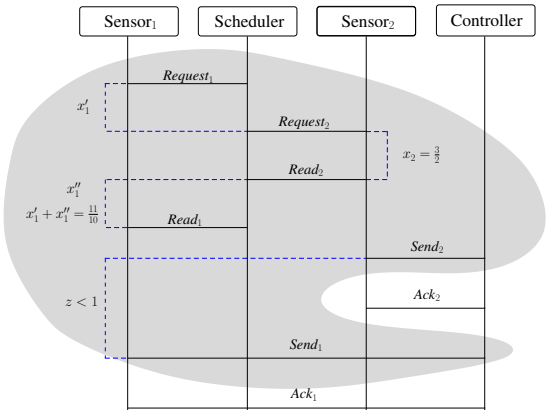

Fig. 2. The MSC for the distributed controller

In order to test the scalability of both approaches, we considered a set of unfeasible MSCs of different lengths, and parameterized the number of HAs in the network. We set a time out of 300 seconds and a memory out of 2 GB. The scatter plot in Figure 1 shows the execution time for both methods on all the instances. The Scenario-based induction is clearly superior to monolithic k-induction. This is due to the exploitation of the structure of the scenario: this results in localized simple path conditions, that are both simpler, and more effective, so that unsatisfiability is detected with a much shorter unrolling.

B. Unfeasibility Explanation

Then, we analyzed the unfeasibility explanations on the three benchmarks with non-trivial scenarios, showing their usefulness in identifying the causes of unfeasibility.

1) Distributed Controller [19]: the benchmark models the interactions of two sensors (sensor_1 and sensor_2) with a controller of a robot. The two sensors interact with a scheduler to access a shared processor. The time needed for computation by the two sensors is bounded but it is non-deterministic, and is tracked in the scheduler with two stopwatches (x_1 and x_2). Also the controller sets a time-out (variable $z = 0$) after the receipt of the first message. If the time-out expires ($z = 10$) the controller discards all the received data.

The MSC shown in Figure 2 models the interaction where sensor_1 requests the processor; the scheduler grants it for a total duration of x_2 time; sensor_2, which has a higher priority, requests and receives grant to the processor; when sensor_2 finishes its computation (event $read_2$), sensor_1 finishes to read data while, in parallel, sensor_2 sends its data to the controller; finally, the sensor_1 and the controller synchronize on $send_1$ and ack_1. The time spent to process the data of sensor_1 is given by the stopwatch x_1. In Figure 2 x_1 is the sum of the intervals x_1' and x_1''. Moreover, we add two additional conditions on the duration of x_1 and x_2 in the scheduler ($x_2 = 1.5$ and $x_1 = 1.1$), and we fix the maximum time spent by the controller before receiving the data from sensor_1 ($z < 1$). The MSC augmented with these constraints is unfeasible.

We prove the unfeasibility of the scenario directly on the concrete system, since all the automata cannot loop performing only local transitions. The analysis takes 3 seconds and the longest simple path is 2 in the controller automaton, and 1 in the other automata. In the Figure 2 we outline in gray the elements of the scenario, events and constraints, which contribute to the unfeasibility. In particular, we find that the unfeasibility depends on all the events of the MSC apart from the events Ack_1 and Ack_2. Moreover, we discover that all the additional constraints of the scenario, $x_2 = 1.5$, $x_1 = 1.1$ and $z < 1$, contribute to the unfeasibility.

We exploit the interpolation techniques to get the constraints $z >= x_1$. In fact, z counts the time elapsed in the controller between the send_1 event and the send_2 event. This means that the controller cannot receive the send_1 message before x_1 seconds, which is the time spent to process data from sensor_1. If we fix $z >= 1.1$ then the scenario is feasible. We find a similar result if we look at the interpolant obtained partitioning the encoding in the constraints from sensor_1 (the A formula) and the rest of the network and the scenario (the B formula). We denote with $time^{event}_{\text{component}}$ the time variable of $component$ when performing $event$. The interpolant is $6 <= time^{request_1}_{\text{sensor}_1} - time^{read_1}_{\text{sensor}_1} + time^{send_1}_{\text{sensor}_1}$. Since $time^{request_1}_{\text{sensor}_1}$ is 6, from the initial condition and invariants of sensor_1, we can infer that the scenario and the other processes in the network do not allow $time^{read_1}_{\text{sensor}_1} <= time^{send_1}_{\text{sensor}_1}$, which is a necessary condition for sensor_1.

2) Audio Control Protocol [19]: this protocol transmits an arbitrary-length bit sequence from a sender to a receiver based on the timing-based Manchester encoding. The protocol relies on division of the elapsed time in slots. Every slot corresponds to a bit. The sender transmits a signal up in the slots corresponding to bits with value 1 (thus, a slot without signals correspond to bit 0). The protocol is robust to bounded errors in the timers used by the sender and receiver.

The considered scenarios consist of unfeasible timed sequences of up. For example, the sequence $\langle up, 4\rangle$, $\langle up, 8\rangle$,

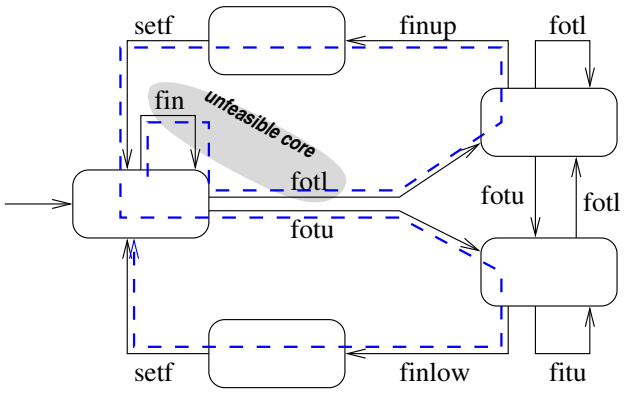

Fig. 3. The automaton structure of the EHC controller with a line that traces the scenario sequence of events.

$\langle up, 12\rangle, \langle up, 16\rangle, \langle up, 19\rangle, \langle up, 23\rangle$ does not respect the protocol, since the 4-th and 5-th events must be separated by 3 seconds.

Scenario-based induction proves that the scenario is unfeasible in 41 seconds. The explanation extracted from the unsat core identifies the 4-th and 5-th events as the cause of unfeasibility. Interpolation "explains" that the inconsistency arises because the sender requires the 5-th event to happen after at least 3.8 seconds; it also shows that the receiver does not play any role in the inconsistency.

3) Electronic Height Control System [27]: this benchmark presents a case where the concrete k-induction is not able to prove the unfeasibility. We therefore rely on abstraction and we show that, despite the over-approximation, the explanation is effective in pinpointing the cause of unfeasibility.

This industrial case study models a system that controls the height of a car's chassis. A timer tells the controller when to read the height from a filter, while disturbances which changes the height of the vehicle are modelled by the environment. The structure of the controller is depicted in Figure 3. The MSC describes a scenario where the height of the chassis falls outside the allowed thresholds, first below and then above the permitted height intervals. The sequence of events in the scenario is highlighted by the dashed line.

The scenario is not feasible due to the timing constraints imposed by the timer on each event and to the dynamics of the environment which requires an incompatible time to pass from the initial level of the chassis to a value read outside the allowed threshold. More precisely, the timer forces every event to happen every second, while the filter chassis level f read by the sensors evolve according to the differential equation $\dot{f} = \frac{h-f}{T}$, where h represents the current level. This is approximated by the linear-phase portrait partitioning which linearizes the differential equation into flow conditions of the form $\dot{f} \in [a, b]$. The constants fixed by the authors of [27] are sufficient to prove the inconsistency.

K-induction proves that the controller and the timer do not have a simple path longer than 1 alternating timed and discrete transitions (since there is no local transition). While, on the concrete state space of the environment, the portrait partitioning creates discrete loops that correspond to infinite simple paths. Therefore we rely on abstraction. We use a set of predicates in the form $t \in [i, i+1]$, $h \in [at, bt]$ and $f \in [at, bt]$ where i is an integer while a and b are the constants used in the partitioning. We localize the abstraction by using $t \in [i, i+1]$ only in the i-th event and considering the partition consistent with the initial values.

With this setting, the tool proves the unfeasibility of the scenario in 4.4 seconds reaching a depth of the longest abstract simple path equal to 6 for the local path before the first event and 9 as for the local path before the second event. The tool correctly reports an unsat core which identifies the first two events as the cause of unfeasibility. The interpolation with regards to components reports that while the timer requires that the second event must happen in no more then 3 seconds, the environment requires the same event to happen at least after 3.3 seconds.

VIII. Conclusions and Future Work

In this paper we have proposed a new approach to proving that a network of hybrid automata has no trace that satisfies a given MSC. We have also proposed the first algorithm to explain the unfeasibility of a scenario. The approach is made practical by the use of segments of the MSC to guide the search, and on the localization of simple paths. The experiments show that the proposed method significantly outperforms techniques based on the reduction to reachability, and is able to construct interesting explanations.

In the future, we will address the issue of non-linear hybrid systems, the use of hierarchical information that is often available in the network, and an automation of the abstraction-refinement loop.

Acknowledgments

This work is sponsored by the European Space Agency under contract number A0/1-6226/10/D/HK Innovative Rover OperatioNs Concepts - Autonomous Planning (IRONCAP).

References

[1] E. Ábrahám, B. Becker, F. Klaedtke, and M. Steffen. Optimizing bounded model checking for linear hybrid systems. In *VMCAI*, pages 396–412, 2005.

[2] S. Akshay, B. Bollig, and P. Gastin. Automata and logics for timed message sequence charts. In *FSTTCS*, pages 290–302, 2007.

[3] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *CONCUR*, pages 114–129, 1999.

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

[5] H. Ben-Abdallah and S. Leue. Timing constraints in message sequence chart specifications. In *FORTE*, pages 91–106, 1997.

[6] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial Order Reductions for Timed Systems. In *CONCUR*, pages 485–500, 1998.

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

[8] L. Bu, A. Cimatti, X. Li, S. Mover, and S. Tonetta. Model Checking of Hybrid Systems using Shallow Synchronization. In *FORTE*, pages 155–169, 2010.

[9] A. Cimatti, E. 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] A. Cimatti, S. Mover, and S. Tonetta. Efficent Scenario Verification for Hybrid Automata. In *CAV*, 2011.

[11] A. Cimatti, S. Mover, and S. Tonetta. Hydi: a language for symbolic hybrid systems with discrete interaction. In *EUROMICRO-SEAA*, 2011.

[12] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[13] L. de Moura, H. Rueß, and M. Sorea. Bounded Model Checking and Induction: From Refutation to Verification. In *CAV*, pages 14–26, 2003.

[14] M. Fränzle and C. Herde. Efficient Proof Engines for Bounded Model Checking of Hybrid Systems. *ENTCS*, 133:119–137, 2005.

[15] 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.

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

[17] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.

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

[19] T. A. Henzinger and P. Ho. Hytech: The cornell hybrid technology tool. In *Hybrid Systems II, LNCS 999*, pages 265–293, 1995.

[20] ITU-T. Recommendation Z.120 - Message Sequence Charts. 1996.

[21] Dubrovin J., T. Junttila, and K. Heljanko. Exploiting step semantics for efficient bounded model checking of asynchronous systems. *Sci. Comput. Program.*, 2011.

[22] J. Klose and H. Wittke. An automata based interpretation of live sequence charts. In *TACAS*, pages 512–527, 2001.

[23] P. Ladkin and S. Leue. On the semantics of message sequence charts. In *FBT*, pages 88–104, 1992.

[24] Peter B. Ladkin and Stefan Leue. Interpreting message flow graphs. *Formal Asp. Comput.*, 7(5):473–509, 1995.

[25] S. Li, S. Balaguer, A. David, K. G. Larsen, B. Nielsen, and S. Pusinskas. Scenario-based verification of real-time systems using uppaal. *Formal Methods in System Design*, pages 200–264, 2010.

[26] S. Mauw and M. A. Reniers. High-level message sequence charts. In *SDL Forum*, pages 291–306, 1997.

[27] O. Müller and T. Stauner. Modelling and verification using linear hybrid automata - a case study. *Mathematical and Computer Modelling of Dynamical Systems*, 71, 2000.

[28] M. Pan, L. Bu, and X. Li. Tass: Timing analyzer of scenario-based specifications. In *CAV*, pages 689–695, 2009.

[29] L. Pike. Real-time system verification by k-induction. Technical Report NASA/TM-2005-213751, NASA, 2005.

[30] M. Samer and H. Veith. On the Notion of Vacuous Truth. In *LPAR*, pages 2–14, 2007.

[31] V. Schuppan. Towards a notion of unsatisfiable and unrealizable cores for LTL. *Science of Computer Programming*, In press, 2010. DOI: 10.1016/j.scico.2010.11.004.

[32] M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In *FMCAD*, pages 108–125, 2000.

[33] S. Tonetta. Abstract Model Checking without Computing the Abstraction. In *FM*, pages 89–105, 2009.

[34] David Walter, Scott Little, Chris J. Myers, Nicholas Seegmiller, and Tomohiro Yoneda. Verification of analog/mixed-signal circuits using symbolic methods. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(12):2223–2235, 2008.

[35] F. Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data structures. *IEEE TSE*, 31(1):38–51, 2005.