Efficient Scenario Verification for Hybrid Automata*

Alessandro Cimatti, Sergio Mover, and Stefano Tonetta

Fondazione Bruno Kessler

Abstract. Hybrid Automata (HAs) are a clean modeling framework for systems with discrete and continuous dynamics. Many systems are structured into components, and can be modeled as networks of communicating HAs. Message Sequence Charts (MSCs) are a consolidated language to describe desired behaviors of a network of interacting components and have been extended in numerous ways. The construction of traces witnessing such behaviors for a given system is an important part of the validation. However, specialized tools to solve this problem are missing. The standard approach encodes the constraints in a temporal logic formula or in additional automata, and then use an off the shelf model checker to find witnesses. However, these approaches are too generic and often turn out to be inefficient.

In this paper, we propose a specialized algorithm to find the behaviors of a given network of HAs that satisfies a given scenario. The approach is based on SMT-based bounded model checking. On one side, we construct an encoding which exploits the events of the scenario and enables the incremental use of the SMT solver. On the other side we simplify the encoding with invariants discovered applying discrete model checking on an abstraction of the HAs. The experimental results demonstrate the potential of the approach.

1 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 [16] (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 modeled as synchronizing transitions of the automata in the network; time elapse is modeled 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 construct traces of a network of HAs that satisfy a given MSC is an important feature to support user validation. However, there has been little research to

* Work supported by the E.U. project MISSA, contract no. ACP7-GA-2008-212088.

address the specific problem, and the typical approach reduces the problem to "off-the-shelf" model checking solutions: the MSC is translated into an equivalent automaton and the required trace is found by reachability in the cross-product of the automaton with the network. This can turn out to be ineffective, exacerbating the difficulty of the reachability problem, which is already very complex in the case of HAs.

In this paper, we tackle the problem of efficiently finding the traces of a network of HAs that satisfy an MSC. We work in the framework of Bounded Model Checking (BMC), which uses at its core an incremental encoding into Satisfiability-Modulo-Theory (SMT).

We first investigate different automata constructions, exploiting optimizations such as locality and partial-order reduction. Second, we propose a specialized, direct algorithm, where the search is structured around the events in the MSC, which are used as intermediate "islands". The idea is to pre-simplify fragments of the encoding based on the events attached to the islands and to incrementally increase the local paths between two consecutive islands. Further simplifications are achieved by means of invariants that are discovered by applying discrete model checking on an abstraction of the HAs. The generated invariants are either over-approximations of the states that are visited between two subsequent events, just before or just after an event of the scenario.

A key enabler for our work is the use of an alternative, "local time" semantics [6] for HAs, which exploits the fact that automata can be "shallowly synchronized" [7]. 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. This results in a more concise semantics, where traces of the network are obtained by composing traces of local automata, each with local time elapse, by superimposing structure based on shared communication.

We implement the various approaches in the sub-case of linear hybrid automata, and we use an incremental SMT solver to check the satisfiability of the formulas encoding the reachability problem. We compare the proposed solutions over a wide set of networks and benchmark MSCs. The results show that the direct algorithm is able to construct witnesses for very wide networks, with very long traces, significantly outperforming the other approaches based on the automata construction, and that the use of invariants can be helpful in further reducing computation time.

The paper is structured as follows. In Section 2 we present some background on networks of HAs, and the SMT-based methods for their reachability analysis. In Section 3 we present the language for describing scenarios, and the methods based on automata construction. In 4 we discuss the proposed direct approach to MSC checking. In Section 5 we discuss related work. In Section 6 we experimentally evaluate our approach. In Section 7 we draw some conclusions.

2 Networks of Hybrid Automata

2.1 Networks of Transition Systems

We first define *Labelled Transition Systems* (LTSs), which are then used to define the semantics of Hybrid Automata. An 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 sub-trace 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 *language* $L(S)$ of an LTS S is the set of traces accepted by some path of S. Given a state $q \in Q$, the language $L_q(S)$ of an LTS S is the set of traces accepted by some path $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} q_k$ of S with $q_k = q$.

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 $\langle Q, A, Q_0, R \rangle$ where:

- $Q = Q_1 \times Q_2$,
- $A = A_1 \cup A_2$,
- $Q_0 = Q_{01} \times Q_{02}$,
- $R := \{\langle \langle q_1, q_2 \rangle, a, \langle q_1', q_2' \rangle \rangle \mid \langle q_1, a, q_1' \rangle \in R_1, \langle q_2, a, q_2' \rangle \in R_2\}$
  $\cup \{\langle \langle q_1, q_2 \rangle, a, \langle q_1', q_2 \rangle \rangle \mid \langle q_1, a, q_1' \rangle \in R_1, a \notin A_2\}$
  $\cup \{\langle \langle q_1, q_2 \rangle, a, \langle q_1, q_2' \rangle \rangle \mid \langle q_2, a, q_2' \rangle \in R_2, a \notin A_1\}$.

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, i.e., $\tau_i = A_i \setminus \bigcup_{j \neq i} A_j$.

Given a network, the *language emptiness problem* is the problem of checking if the language of a network is empty. Given a network $\mathcal{N}$ and a predicate $q \in Q_1 \times \ldots \times Q_n$, the *reachability problem* is the problem of checking if the language $L_q(\mathcal{N})$ is empty.

2.2 Hybrid Automata

A *Hybrid Automaton* (HA) [16] 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,
- $\mu : Q \rightarrow P(X, \dot{X})$ is the flow condition,
- $\iota : Q \rightarrow P(X)$ is the initial condition,
- $\xi : Q \rightarrow P(X)$ is the invariant condition,
- $\theta : R \rightarrow P(X, X')$ is the jump condition,

where 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 is a conjunction of inequalities.

A *network* $\mathcal{H}$ of HAs is the parallel composition of two or more HAs. We consider two semantics for networks of HAs: the global-time semantics, where all components synchronize on timed events, and the local-time (or time-stamps) semantics, where the timed events are local and components must synchronize the time on shared events.

In the following, we consider a network $\mathcal{H} = H_1 || \dots || H_n$ 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 *global-time semantics* (or time-action semantics) [16] of $\mathcal{H}$ is the network of LTSs $\mathcal{N}_{\text{GLTIME}}(\mathcal{H}) = S_1 || \dots || S_n$ with $S_i = \langle Q'_i, A'_i, Q'_{0i}, R'_i \rangle$ where

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

The *local-time semantics* (or time-stamps semantics) [6] of $\mathcal{H}$ is the network of LTSs $\mathcal{N}_{\text{LOCTIME}}(\mathcal{H}) = S_1 || \dots || S_n$ with $S_i = \langle Q'_i, A'_i, Q'_{0i}, R'_i \rangle$ where

- $Q'_i = \{\langle q, \overline{x}, t \rangle \mid q \in Q_i, \overline{x} \in \mathbb{R}^{|X_i|}, t \in \mathbb{R}_{\geq 0}\}$,
- $A'_i = \{\langle a, t \rangle \mid a \in A_i, t \in \mathbb{R}_{\geq 0}\} \cup \{\textsc{Time}_i\}$,
- $Q'_{0i} = \{\langle q, \overline{x}, 0 \rangle \mid q \in Q_{0i}, \overline{x} \in \iota_i(q)\}$,
- $R'_i = \{\langle \langle q, \overline{x}, t \rangle, \langle a, t \rangle, \langle q', \overline{x}', t \rangle \rangle \mid \langle q, a, q' \rangle \in R_i, \langle \overline{x}, \overline{x}' \rangle \in \theta_i(q, a, q'), \overline{x} \in \xi_i(q), \overline{x}' \in \xi_i(q')\} \cup \{\langle \langle q, \overline{x}, t \rangle, \textsc{Time}_i, \langle q, \overline{x}', t' \rangle \rangle \mid$ 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 LTSs contains a local timed event $\textsc{Time}_i$ 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 $\langle \textsc{Time}, \delta \rangle$, agreeing on the time elapsed during the transition (δ variable).

If $q = \langle \langle q_1, \overline{x_1}, t_1 \rangle, \dots, \langle q_n, \overline{x_n}, t_n \rangle \rangle$ is a state of $\mathcal{N}_{\text{LOCTIME}}$, we say that q is synchronized iff $t_i = t_j$ for $1 \leq i < j \leq n$, i.e., the local times are equal.

Theorem 1 ([6]). *$\langle \langle q_1, \overline{x_1} \rangle, \dots, \langle q_n, \overline{x_n} \rangle \rangle$, is reachable in $\mathcal{N}_{\text{GLTIME}}(\mathcal{H})$ iff there exists a synchronized state $\langle \langle q_1, \overline{x_1}, t \rangle, \dots, \langle q_n, \overline{x_n}, t \rangle \rangle$ reachable in $\mathcal{N}_{\text{LOCTIME}}(\mathcal{H})$.*

In an extended version of this paper, available at `http://es.fbk.eu/people/mover/hybrid_scenario/`, we prove a stronger version of the theorem, which shows that also a time-abstract version of the traces is preserved.

2.3 SMT Encoding of Hybrid Automata

As described in [16], LHAs can be analyzed with symbolic techniques. Let us consider a network $\mathcal{H} = H_1 || \dots || H_n$ of LHAs whose semantics (either global or local time) is given by the network of LTSs $S_1||\dots||S_n$ 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 such that there exists a mapping χ from the assignments of V_i to the states of the S_i. The events of A_i can be represented by a set of symbolic variables W_i such that χ maps every assignment of variables in W_i to an event in A_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 $I_i(V_i)$ such that $\mu(V_i) \models I_i$ iff $\chi(\mu) \in Q_{i0}$, and a formula T_i such that $\mu(V_i), \nu(W_i), \mu'(V_i') \models T_i$ iff $(\chi(\mu), \chi(\nu), \chi(\mu')) \in R_i$, where $\mu(V_i), \nu(W_i)$ and $\mu(V_i')$ are assignments to the set of variables V_i, W_i and V_i'.

The details of the encoding we use can be found in [7]. 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 encoding the global-time semantics a further real input variable δ_i represents the time elapsed in a timed transition. Instead, 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 $path(k) := I^0 \wedge \bigwedge_{0\leq i<k} T^i$.

Theorem 2. *There exists a mapping χ from the models of $path(k)$ to the paths of S of length k.*

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. Suppose the problem is parametrized by a certain k, i.e. $PB = \exists k.PB(k)$. In order to exploit the incremental interface of the solver, the problem $PB(k)$ is formulated into two parts $PB(k) = \alpha(k) \wedge \beta(k)$, as in [9], such that $\alpha(0) = \gamma(0)$ and $\alpha(k+1) = \alpha(k) \wedge \gamma(k+1)$. This way, the solver faces sub-problems of incremental difficulties and can reuse previous results:
$\gamma(0) \wedge \beta(0)$
$\gamma(0) \wedge \gamma(1) \wedge \beta(1)$
$\gamma(0) \wedge \gamma(1) \wedge \gamma(2) \wedge \beta(2)$
...

If we want to solve the reachability problem, we look for a k for which the problem $PB(k) = path(k) \wedge target^k$ is satisfiable. The problem is usually presented to the solver in the following form: $\gamma(0) := I^0$, $\gamma(k) := T^{k-1}$, $\beta(k) := target^k$, for $k > 0$.

The non-monotonicity of the encoding is handled with a standard stack-based interface of the SMT solver (PUSH, ASSERT, SOLVE, POP primitives). This allows, after asserting $\gamma(k)$, to set a backtrack point (PUSH), assert $\beta(k)$ (ASSERT), check the satisfiability of the conjunction of the asserted formulas (SOLVE), and to restore the state of the solver (i.e. asserted formulas and learned clauses) at the backtrack point (POP). This way, the k+1-th problem is solved keeping all the learned clauses related to $\gamma(k)$.

3 Message Sequence Charts

3.1 MSCs for Networks of Hybrid Automata

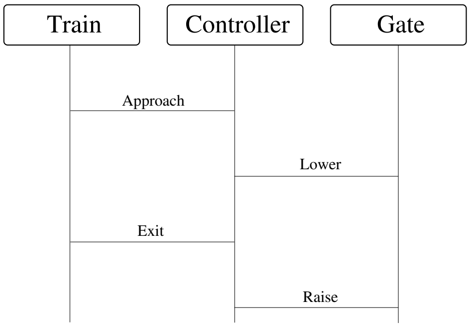


Fig. 1. An MSC for the Train-Gate-Controller model [16]

A Message Sequence Chart (MSC) [23] defines a possible interaction of components in a network $\mathcal{N}$. An MSC m is associated with a set of events $A_m \subseteq A_{\mathcal{N}}$, subset of the events of the network. The MSC defines a sequence of events for every component S of the network, called instance of S.

The typical implicit assumption is that the set A_m contains all the synchronization events of the network. However, if $\mathcal{N}$ is a network of hybrid automata, even if we consider the global-time semantics in which the timed event is shared, the timed event is not part of A_m and, thus, is not present in the sequence of events specified by the MSC. Therefore, we assume that independently from the semantics, if $\mathcal{N}$ is a network of the hybrid automata $H_1, \ldots, H_n$ with alphabet respectively $A_1, \ldots A_n$, then $A_m = \bigcup_{1 \leq i < j \leq n} A_i \cap A_j$ and thus the (global or local) timed event is not part of A_m[1].

An instance σ for the LTS S is a sequence $a_1; \ldots; a_h \in (A_m \cap A_S)^*$ of events of S. S accepts the instance ($S \models \sigma$) iff there exists a trace w accepted by S ($w \in L(S)$) 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 $\pi \models \sigma$.

An MSC is the parallel composition $\sigma_1 || \ldots || \sigma_n$ of $\sigma_1, \ldots, \sigma_n$ 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 $\mathcal{N}$ ($w \in L(\mathcal{N})$) such that, for every S_i, the sub-sequence of events in $A_m \cap A_{S_i}$ is equal to σ ($w_{|(A_m \cap A_{S_i})} = \sigma$). 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 $\mathcal{N}_{\text{GLTIME}}(\mathcal{H}) \models m$.

The model checking problem for an MSC m is the problem of checking if a network satisfies an MSC. If we define the language $L(m)$ of an MSC m as the traces compatible with the instances of m, the model checking problem can be seen as the problem of checking if $L(\mathcal{N}) \cap L(m) \neq \emptyset$.

An MSC $\sigma_1 || \ldots || \sigma_n$ 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 $A = A_i \cap A_j$, $\sigma_{i|A} = \sigma_{j|A}$. In other words, the MSC m is consistent iff $L(m) \neq \emptyset$. Henceforth, we assume that the MSCs

[1] The techniques presented in this paper can be adapted to consider also the case where a synchronization is not in A_m.

are consistent. The check of consistency is trivial and can be done syntactically on the graphical representation of the MSC.

Example 1. Figure 1 shows an MSC for the railroad model from [16]. There is an instance for each automaton in the network, *Train*, *Controller* and *Gate*. The MSC represents a scenario where the *Train* communicates with the controller when approaching the *Gate* and the controller synchronizes with the *Gate* to close it. When the *Train* is far, it synchronizes with the *Controller*, which opens the *Gate*.

3.2 Standard Global Automata Construction

The classic construction of an automaton which monitors the satisfaction of an MSC m proceeds by building one state for combination of locations. Every instance has one location before each event and a final location after the last event. A cut through m is a set of locations, one for each instance (we do not require the cuts to be downward closed with regard to the events because we guarantee that the reachable cuts respect the events). It is called "cut" because it cuts the graphical representation of the MSC into two parts, the one already visited and the one to be monitored.

Formally, if the instance $\sigma_i = a^i_1; \ldots ; a^i_{h_i}$, we represent a location with indexes from 0 to h_i. A cut is therefore a tuple of indexes. Given a cut $c = \langle c_1, \ldots, c_n\rangle$, an event a is said enabled in c iff there exists a set of indexes $J \subseteq [1, n]$ such that for all $j \in J$, $a \in A_j$, the event after the location c_j is a (i.e., $a^i_{c_j+1} = a$), and, for all $j \notin J$, $a \notin A_j$. Note that for a given event a, the set J is unique and we denote it with J^c_a.

The LTS S_m corresponding to an MSC $m = \sigma_1|| \ldots ||\sigma_n$, with $\sigma_i = a^i_1; \ldots ; a^i_{h_i}$, is defined as follows:

- $Q = [0, h_1] \times \ldots \times [0, h_n]$,
- $A = A_m$,
- $Q_0 = \langle 0, \ldots, 0\rangle$,
- $R = \{(c, a, c') \mid a$ is enabled in c and, for all $j \in J^c_a$, $i'_j = i_j + 1$, and, for all $j \notin J^c_a$, $i'_j = i_j\}$.

Theorem 3. $L(m) = L(S_m)$.

3.3 Exploiting Independent Events

Reduced Global Automata. In the case of synchronizations on discrete events as for hybrid automata (thus without shared variables), two transitions on different components with different events are independent. This means that the order of the transitions does not affect the state after their application. If the order is irrelevant for the search problem, we can fix an arbitrary interleaving. As noted in [6], this reduction can be amplified if we adopt the local-time semantics, since timed transitions become local and independent from the timed transitions of other components.

In the case of model checking MSC, since we are interested in finding one trace compatible with the MSC, if we use the local-time semantics, we can fix an arbitrary interleaving of parallel events in an MSC, and produce an automaton which is linear in the number of events.

If J and J' are subsets of $[1, n]$, we say that $J < J'$ if J contains an integer lower than any integer in J'. If a_1 and a_2 are enabled in c, we say that $a_1 < a_2$ iff $J^c_{a_1} < J^c_{a_2}$. Clearly, given a cut c, there exists a minimum enabled event.

We can build a reduced LTS S_m corresponding to an MSC m as follows:

- $Q = [0, h_1] \times \ldots \times [0, h_n]$,
- $A = A_m \times \mathbb{R}_{\geq 0}$,
- $Q_0 = \langle 0, \ldots, 0 \rangle$,
- $R = \{\langle c, \langle a, t \rangle, c' \rangle \mid c \in Q, t \in \mathbb{R}_{\geq 0}$ and a is the minimum enabled event in c and, for all $j \in J^c_a$, $i'_j = i_j + 1$, and, for all $j \notin J^c_a$, $i'_j = i_j\}$.

Theorem 4. *If* $\mathcal{N}_{\text{LocTime}}(\mathcal{H}) = S_1 || \ldots || S_n$*,* $\mathcal{H} \models m$ *iff* $L(S_1 || \ldots || S_n || S_m) \neq \emptyset$.

Distributed Automata. Another way to exploit the independence of the parallel events in a MSC is to build a distributed version of the LTS S_m, with one component for every component of the network. We then apply the step-semantics encoding [14] to parallelize the encoding of independent transitions. Let us define the S^i_m as follows:

- $Q = [0, h_i]$,
- $A = A_m \times \mathbb{R}_{\geq 0}$,
- $Q_0 = 0$,
- $R = \{\langle u, \langle a, t \rangle, u' \rangle \mid u \in Q, a = a^i_{u+1}$ and $u' = u + 1\}$.

Theorem 5. *If* $\mathcal{N}_{\text{LocTime}}(\mathcal{H}) = S_1 || \ldots || S_n, \mathcal{H} \models m$ *iff* $L(S_1 || S^1_m || \ldots || S_n || S^1_m) \neq \emptyset$.

A similar result can be obtained for the global-time semantics, as proposed in [19].

4 Scenario-Driven Encoding

4.1 Encoding Tailored to MSCs

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* [7]. 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.

Let us consider a network $\mathcal{H} = H_1 || \ldots || H_n$ 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).

Let us fix a maximum number k of local events between two shared events. We encode the path of the i-th component along the instance $\sigma_i = a_1; \ldots; a_{h_i}$ of an MSC into the following formula:

$$enc(\sigma_i, k) := I_i^0 \wedge \bigwedge_{1 \leq j \leq k} T_{i|\tau}^{j-1} \wedge \bigwedge_{1 \leq u \leq h_i} (T_{i|a_u}^{(u*k)+u-1} \wedge \bigwedge_{1 \leq j \leq k} T_{i|\tau}^{(u*k)+u+j-1}) \quad (1)$$

Intuitively, $enc(\sigma_i, k)$ encodes the alternation of sequences of at most k local steps with the events in the instance σ_i. Note that the u-th event is encoded at the $((u*k)+u-1)$-th step because it is preceded by $u*k$ many local events and $u-1$ shared events.

The run of a network along a consistent MSC $m = \sigma_1 || \ldots || \sigma_n$ can be encoded into:

$$enc(m, k) := \bigwedge_{1 \leq j \leq n} enc(\sigma_j, k) \wedge \bigwedge_{1 \leq j < i \leq n} sync(\sigma_j, \sigma_i) \wedge (t_j^{last_j} = t_i^{last_i}) \quad (2)$$

where $last_i = (k+1) * h_i + k$ is the index of the last state of the encoding and $sync(\sigma_j, \sigma_i)$ says that the h-th occurrence of a shared event must occur at the same time in σ_j and σ_i. More, specifically, if $A = A_i \cap A_j$ and $\sigma_{j|_A} = \sigma_{i|_A} = a_1; \ldots a_l$ and $f_i, f_j : \mathbb{N} \rightarrow \mathbb{N}$ are such that $a_z = \sigma_i(f_i(z)) = \sigma_j(f_j(z))$, for $1 \leq z \leq l$, then:

$$sync(\sigma_j, \sigma_i) := \bigwedge_{1 \leq z \leq l} t_i^{(f_i(z)*k)+f_i(z)-1} = t_j^{(f_j(z)*k)+f_j(z)-1} \quad (3)$$

The function $f_i(z)$ maps the z-th event a_z in $\sigma_{i|_A}$ to the index of a_z in σ_i. Thus, the index $(f_i(z) * k) + f_i(z) - 1$ is the same index used in the encoding of $enc(\sigma_i, k)$ to encode the transition labeled with the shared event a_z.

Note that the encoding allows to express very complex constraints on the MSC. In fact, it is possible to formulate constraints on the states of the network along events just referring to the symbolic variables that represent them.

Example 2. We show the encoding of the MSC of Example 1 for the *Train* automaton with 2 local steps and the synchronization constraints with the *Controller* automaton. I_{Train} and T_{Train} are the initial condition and the transition relation *Train*.

$$\begin{aligned} enc(\sigma_{train}, 2) := {} & I_{Train}^0 \wedge T_{Train|\tau}^0 \wedge T_{Train|\tau}^1 \wedge \\ & T_{Train|Approach}^2 \wedge T_{Train|\tau}^3 \wedge T_{Train|\tau}^4 \wedge \\ & T_{Train|Approach}^5 \wedge T_{Train|\tau}^6 \wedge T_{Train|\tau}^7 \wedge \\ sync(\sigma_{Train}, \sigma_{Controller}) := {} & t_{Train}^2 = t_{Controller}^2 \wedge t_{Train}^5 = t_{Controller}^{11} \end{aligned}$$

Theorem 6.[2] *Given a consistent MSC m for the network $\mathcal{H}$, if $enc(m, k)$ is satisfiable then $\mathcal{H} \models m$. Vice versa, if $\mathcal{H} \models m$, then there exists an integer k such that $enc(m, k)$ is satisfiable.*

4.2 Incrementality

The encoding is conceived in order to maximize the incrementality of the solver, as described in Section 2.3, along the increase of k, the length of a sequence of local steps.

[2] The proof of the theorem is available at `http://es.fbk.eu/people/mover/hybrid_scenario/`.

The idea is that we keep encodings of the sequences of local transitions separated and we unroll them incrementally, while we add and remove accordingly the constraints which glue such sequences.

Let us fix a maximum K as a bound for the number k of local transitions between two shared events. We use K for distancing enough the (fixed) positions of events. With regard to the formulas introduced in Section 2.3, we define the partial encoding for an instance σ_i as follows:

$$\gamma_{enc(\sigma_i)}(0) := I_i^0 \wedge \bigwedge_{1 \leq u \leq h_i} T_{i|a_u}^{(u*K)+u-1}$$

$$\gamma_{enc(\sigma_i)}(k) := \bigwedge_{0 \leq u \leq h_i} T_{i|\tau}^{(u*K)+u+k-1}$$

$$\beta_{enc(\sigma_i)}(k) := \bigwedge_{0 \leq u \leq h-1} V^{(u*K)+u+k} = V^{(u*K)+u+K}$$

For each instance σ_i we encode the initial condition and all the h_i events in $\gamma_{enc(\sigma_i)}(0)$. We incrementally increase the length of the local step in $\gamma_{enc(\sigma_i)}(k)$ and in $\beta_{enc(\sigma_i)}(k)$, which glues the last state of a sequence of local steps with the first state that performs the next shared event.

The incremental encoding considering the whole MSC m is defined as follows:

$$\gamma(0) := \bigwedge_{1 \leq i \leq n} \gamma_{enc(\sigma_i)}(0) \wedge \bigwedge_{1 \leq i < j \leq n} sync(\sigma_i, \sigma_j)$$

$$\gamma(k) := \bigwedge_{1 \leq i \leq n} \gamma_{enc(\sigma_i)}(k)$$

$$\beta(k) := \bigwedge_{1 \leq i \leq n} \beta_{enc(\sigma_i)}(k) \wedge \bigwedge_{1 \leq j < n} t_j^{last_j} = t_{j+1}^{last_{j+1}}$$

Theorem 7. *There exists a renaming of variables ι such that $enc(m, k)$ and $\bigwedge_{0 \leq j \leq k} \gamma(j) \wedge \beta(k)$ are the syntactical equal modulo the renaming.*

4.3 Scenario-Driven Invariants Generation

In order to strengthen the scenario-driven encoding of Section 4.1, and thus speed up the search, we generate invariants from abstractions of the hybrid automata in the network.

Each instance σ of the MSC restricts the behavior of the automaton S. We abstract S to a finite state system $\hat{S}$ and we use standard techniques, in our case BDDs, to generate invariants which holds in different sections of σ. In particular, we find the reachable states of $\hat{S}$ between two events, just before an event, and just after an event. The invariants are then conjoined to the scenario encoding.

Consider the instance $\sigma = a_1; \ldots; a_h$ of an MSC m. If $S \models \sigma$, by definition, there exists a path π of S over trace w such that $w_{|A_m} = \sigma$. In order to satisfy σ, π alternates sequences of consecutive local events with shared events. More, specifically, if $\pi \models \sigma$, π must be in the form $q_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_{j_1} \xrightarrow{a_1} q_{j_1+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_{j_h} \xrightarrow{a_h} q_{j_h+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_{j_{h+1}}$, where $q_i \in Q$ and τ are local events of S. We split the path π into a set Υ of sub-sequences such that $\Upsilon = \{\pi_{pre_i}, \pi_{post_i} \mid i \in 1, \ldots, h\} \cup \{\pi_{\alpha_i} | i \in 0, \ldots, h\}$, where:

- $\pi_{pre_i} = \{q_{j_i}\}$, it is the source state of the transition labeled with a_i in π.
- $\pi_{post_i} = \{q_{j_i+1}\}$, it is the destination state of the transition labeled with a_i in π.
- $\pi_{\alpha_i} = \{q_{j_i+1}, \ldots, q_{j_{i+1}}\}$ where we denoted 0 with $j_0 + 1$.

Given an MSC instance $\sigma = a_1; \ldots; a_h$ for the system S, we find the constraints $pre_i, post_i$, $1 \leq i \leq h$ and α_i, $0 \leq i \leq h$, such that, for every π such that $\pi \models \sigma$:

- for all u, $0 \leq u \leq h$, for all $q \in \pi_{\alpha_u}$, $q \models \alpha_u$;
- for all u, $1 \leq u \leq h$, for all $q \in \pi_{pre_u}$, $q \models pre_u$;
- for all u, $1 \leq u \leq h$, for all $q \in \pi_{post_u}$, $q \models post_u$.

Thus, the constraints are necessary conditions for the paths to satisfy the instance. We can safely strengthen the encoding of the scenario with such constraints in order to speed up the search.

We perform the invariant generation process for S and σ in three different steps: we compute the abstraction $\hat{S}$, we perform a forward reachability computing a first set of invariants and finally we refine the invariants with a backward reachability analysis. We compute the Boolean Abstraction $\hat{S}$ of S, replacing each predicate of S with a fresh Boolean variable, and we represent $\hat{S}$ with Binary Decision Diagrams (BDDs). Then, we perform a forward reachability analysis on $\hat{S}$ computing an over-approximation of $post_i$ and α_i. We start the reachability analysis from the initial states of $\hat{S}$, and we compute α_0 with a fixed-point of the image restricted to the local events τ. Then, starting from α_0, we compute $post_1$ with a single image computation restricted to a_1. We alternate these two steps for all a_i of σ. Finally, we perform a backward reachability analysis on $\hat{S}$ to compute pre_i and to refine $post_i$ and α_i. We start from $post_h$ and we compute the precise pre_h as the intersection of α_{h-1} and the pre-image of $post_h$ restricted to the event a_h. Then, we refine α_{h-1}, intersecting it with the fixed-point of the pre-image which starts from pre_h and is restricted to τ. At this point we refine $post_{h-i}$, intersecting it with α_{h-1}. We iterate these steps following σ in reverse order.

5 Related Work

There have been a lot of extensions to MSC: High-Level Message Sequence Charts [20], UML's sequence diagrams and Live Sequence Charts [10]. While several works aim to increase the expressiveness of these languages, MSCs are a basic building blocks, used to describe the parallel composition of sequences of shared events among components. In this paper, we consider the basic version of MSCs, which describe the parallel composition of sequences of events and, as in [18], we consider the semantics of MSCs in terms of traces, i.e., they constrain the sequence of observable/shared events.Our approach can be extended to manage more expressive languages, such as High-Level Message Sequence Chart, which adds the alternative, sequential and parallel composition of MSCs, using the scenario-based encoding as a building block. We can easily manage time constraints in the MSC as introduced in [2, 5].

MSC and its extensions have been used in [3, 22] to describe an entire system. In [3] they solve the model checking problem for a system expressed with MSCs translating then into automata, while in [22] the authors check the consistency of UML Message

Sequence Charts using linear programming. Both approaches differ from ours, since MSCs are used as a modeling language and not as a specification language.

Many works present different translations of MSCs into temporal logics or automata used for model checking a design. Most of these works focus on covering different aspects of the extensions proposed by LSCs. In [11], the authors study the expressive power of LSCs compared to CTL*. In [17], the authors consider charts with universal semantics concentrating on (discrete-time) timing constraints and synchronous events; they translate the charts into Timed Büchi Automata which are then translated into temporal logic to be checked with the model checker STATEMATE. The work is extended in [19] to handle more expressive timing constraints, translating an LSC either into a global timed automaton or into a network of timed automata; moreover, the translation is integrated with the UPPAAL model checker. In these works, the model checking techniques are used off-the-shelf, but the verification engine is not optimized to deal with the scenario specification. Surprisingly, there are not many works that optimize the algorithms in order to scale up the analysis exploiting the structure of the scenarios. The key difference with such works is in that here we focus on efficiency, rather than on covering all the features provided by MSC extensions. We note that our approach can be easily extended to deal with several features of extended MSCs.

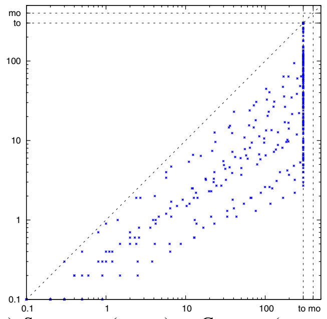
Bounded model checking for hybrid systems using SMT solvers has been investigated in [1, 4, 7, 12, 13]. These techniques can be used to verify a scenario by translating the scenario into an automaton. Still, such techniques are not tailored to the verification of a scenario, and they result in a loss of efficiency since the structure of the problem under analysis is not taken into account. Existing optimizations to the BMC encoding [1] are orthogonal to the scenario-based encoding and can be applied when encoding sequence of local events. Our specific encoding guided by the scenario is inspired by [7], where the authors present a different semantics of the BMC problem for hybrid systems, obtained by composing traces of the local automata, and superimposing compatibility constraints resulting from the synchronizations. In fact, in our approach, the encodings of the automata are local, and a synchronization between two or more automata can happen at different times in the encoding.

6 Experimental Evaluation

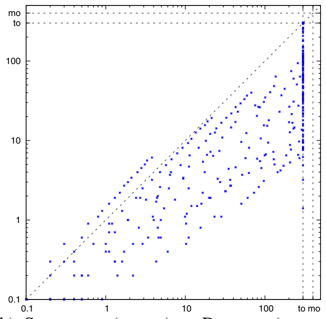
The techniques discussed in previous sections were implemented on top of the model checker NUSMV3, that features a bounded model checking approach to the reachability in networks of HAs., and uses at its core the MATHSAT SMT solver. We implemented the approach based on the automata constructions, for which we perform a reachability analysis using the incremental BMC search of NUSMV3 on the composition of the network and the scenario automaton. The reachability target is given by the final states of the automaton. The scenario-driven encodings were implemented in the same framework; for the invariant computation we use the BDD-based model checking of NUSMV3 applied on a Boolean abstraction of the HAs. Also the scenario-driven encoding search was implemented exploiting the incrementality of the SMT solver. In the following, we call SCENARIO the scenario-driven encoding and SCENARIOINVAR its variant simplified with invariants. As for the translations of the MSC into automata, GLOBAL is the reduced global automata which uses the local time semantics, DISTRIB is the distributed automata with global time semantics, and DISTRIBLOCAL is the distributed automata with local time semantics.

In the experimental evaluation, we used the following benchmarks taken from the literature and formalized using the HYDI language [8]. *Star-shape Fischer* is a hybrid version of the Fischer mutual exclusion protocol, that uses a shared variable to control the access to a critical session. *Ring-shape Fischer* is hybrid variant where processes are in a ring, and each process shares a lock variable with its neighbors. *Nuclear Reactor* [25] model the control of a nuclear reactor with n rods. *Distributed Controller* [15] models the interactions of n sensors with a preemptive scheduler and a controller. *Electronic Height Control System (EHC)* [21] is an industrial case study of a system which controls the height of a chassis by pneumatic suspension. The original non-linear model is linearized using *linear-phase portrait partitioning*, as proposed in [21]. For each benchmark, we defined meaningful MCSs that describe the interaction of all the automata in the benchmarks, possibly containing parallel event synchronizations. All the MSCs used in the experimental evaluation are satisfiable.

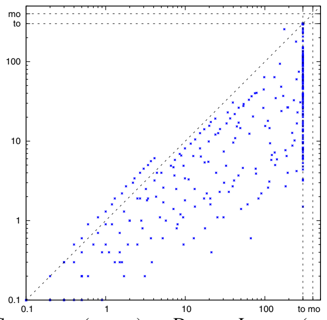
We evaluated the scalability of the proposed approaches with respect to the number of components in the network and with respect to the length of the MSCs. We increase the number of the components for all the benchmarks, except for the *EHC* which has a fixed number of processes. We increase the length of the MSCs repeating the sequence



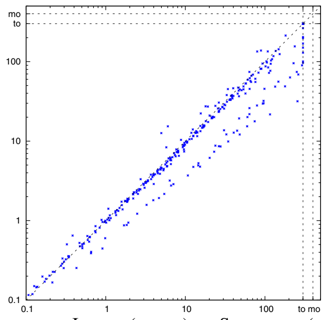
(a) SCENARIO (y axes) vs. GLOBAL (x axes)



(b) SCENARIO (y axes) vs. DISTRIB (x axes)

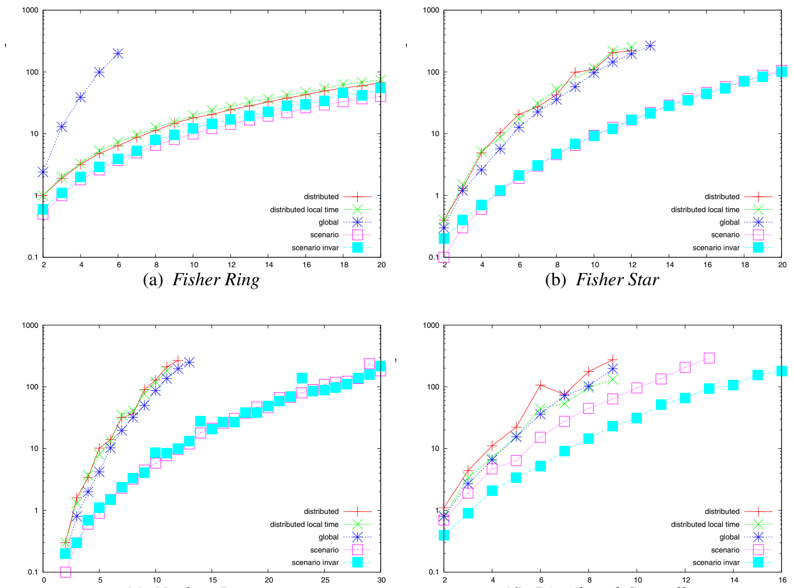


(c) SCENARIO (y axes) vs. DISTRIBLOCAL (x axes)



(d) SCENARIOINVAR (y axes) vs. SCENARIO (x axes)

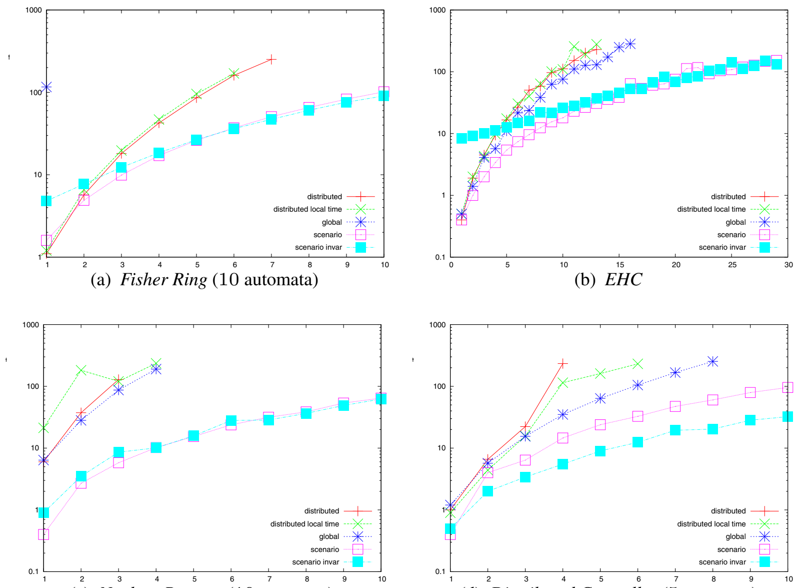
Fig. 2. (a)-(c) Scatter plots of run times (sec.). (d) the reduction due to invariants on search time.



(a) *Fisher Ring* (b) *Fisher Star*

(c) *Nuclear Reactor* (d) *Distributed Controller*

Fig. 3. Run times (sec. on the y axes) increasing the number of automata (x axes)



(a) *Fisher Ring* (10 automata) (b) *EHC*

(c) *Nuclear Reactor* (10 automata) (d) *Distributed Controller* (5 automata)

Fig. 4. Run times (sec. on the y axes) repeating the scenario n times (y axes)

of messages of the scenario for an arbitrary number of times. All the experiment were run on a Linux machine equipped with an Intel i7 CPU at 2.93 GHz, setting the timeout and the memory out for a single benchmark to 300 seconds and to 2 GB of RAM. All the results, the test cases and the executable used in the experimental evaluation are available at `http://es.fbk.eu/people/mover/tests/CAV11/`.

The main findings of the experimental evaluation regard the effectiveness of the scenario-based encoding, which outperforms the approaches based on the optimized automata construction. In Figure 2 we compare the run times (in logarithmic scale) for all the tested instances of benchmarks and scenarios of the scenario-driven encoding and the automata approaches. The scenario-driven encoding demonstrates its efficiency outperforming the automata approaches in nearly all the benchmarks, sometimes by orders of magnitude, terminating the execution on benchmarks where the automata approach reaches the time-out.

The plots in Figure 3 show the scalability with respect to the number of the automata in the network for all the benchmarks, except the *EHC*, for MSC of fixed structure. Each plot from Figure 3 shows the run time (in seconds) of the different methods on the y axes and the number of automata on the x axes. A point in one of these plots is the run time of a specific method for a specific number of automata. These plots show that the scenario-based encoding scales much better than the automata-based approaches.

Figure 4 (a)-(e) shows the effect of increasing the size of the scenario, fixing the number of automata in the network. In general, again, the scenario-driven encoding is more efficient and scales better than the automata-based approaches.

The simplification brought by the invariants computed from the abstraction is not always useful, although they rarely are detrimental (see Figure 3 and 4). When the system under analysis has a small discrete state space the invariants do not bring a speed up to the search. The impact of invariants is very strong when the complexity of the automata increases, as in the case of the *Distributed Controller* benchmark. The effect of the invariants on the search is highlighted in Figure 2(d), where we plot the *search* times, excluding the time taken to generate the invariants.

7 Conclusions

In this paper we have addressed the problem of finding traces satisfying MSCs. The problem is highly relevant to verification, in that MSCs are very useful to allow end users to validate both requirements and designs. We investigated the use of a specialized algorithm that uses the segments of the MSC to guide the search, based on the use of a local time semantics. The experiments show that the proposed method significantly outperforms optimized techniques based on automata construction.

In the future, we will extend the language support for MSCs and we apply techniques such as k-induction and abstraction [24] to prove that a network does not satisfy an MSC.

References

1. Ábrahám, E., Becker, B., Klaedtke, F., Steffen, M.: Optimizing bounded model checking for linear hybrid systems. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 396–412. Springer, Heidelberg (2005)

2. Akshay, S., Bollig, B., Gastin, P.: Automata and logics for timed message sequence charts. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 290–302. Springer, Heidelberg (2007)
3. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, p. 114. Springer, Heidelberg (1999)
4. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying Industrial Hybrid Systems with MathSAT. ENTCS 119(2), 17–32 (2005)
5. Ben-Abdallah, H., Leue, S.: Timing constraints in message sequence chart specifications. In: FORTE, pp. 91–106 (1997)
6. Bengtsson, J.E., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 485–500. Springer, Heidelberg (1998)
7. Bu, L., Cimatti, A., Li, X., Mover, S., Tonetta, S.: Model checking of hybrid systems using shallow synchronization. In: Hatcliff, J., Zucca, E. (eds.) FMOODS 2010. LNCS, vol. 6117, pp. 155–169. Springer, Heidelberg (2010)
8. Cimatti, A., Mover, S., Tonetta, S.: Hydi: a language for symbolic hybrid systems with discrete interaction. Technical report, Fondazione Bruno Kessler (2011)
9. Claessen, K., Sörensson, N.: New techniques that improve mace-style finite model finding. In: Baader, F. (ed.) CADE 2003. LNCS (LNAI), vol. 2741, Springer, Heidelberg (2003)
10. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. Formal Methods in System Design 19(1), 45–80 (2001)
11. Damm, W., Toben, T., Westphal, B.: On the expressive power of live sequence charts. Program Analysis and Compilation, 225–246 (2006)
12. Fränzle, M., Herde, C.: Efficient Proof Engines for Bounded Model Checking of Hybrid Systems. ENTCS 133, 119–137 (2005)
13. Fränzle, M., Herde, C.: HySAT: An efficient proof engine for bounded model checking of hybrid systems. Formal Methods in System Design 30(3), 179–198 (2007)
14. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. Theory and Practice of Logic Programming 3(4-5), 519–550 (2003)
15. Henzinger, T.A., Ho, P.: Hytech: The cornell hybrid technology tool. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 265–293. Springer, Heidelberg (1995)
16. Henzinger, T.A.: The Theory of Hybrid Automata. In: LICS, pp. 278–292. IEEE Computer Society Press, Los Alamitos (1996)
17. Klose, J., Klose, H.: An automata based interpretation of live sequence charts. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, p. 512. Springer, Heidelberg (2001)
18. Ladkin, P.B., Leue, S.: Ladkin and Stefan Leue. On the semantics of message sequence charts. In: FBT, pp. 88–104 (1992)
19. Li, S., Balaguer, S., David, A., Larsen, K.G., Nielsen, B., Pusinskas, S.: Scenario-based verification of real-time systems using uppaal. Formal Methods in System Design, 200–264 (2010)
20. Mauw, S., Reniers, M.A.: High-level message sequence charts. In: SDL Forum, pp. 291–306 (1997)
21. Müller, O., Stauner, T.: Modelling and verification using linear hybrid automata - a case study. Mathematical and Computer Modelling of Dynamical Systems 71 (2000)
22. Pan, M., Bu, L., Li, X.: TASS: Timing analyzer of scenario-based specifications. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 689–695. Springer, Heidelberg (2009)
23. Reniers, D.M.A.: Message Sequence Chart: Syntax and Semantics. PhD thesis (1999)
24. Tonetta, S.: Abstract model checking without computing the abstraction. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 89–105. Springer, Heidelberg (2009)
25. Wang, F.: Symbolic parametric safety analysis of linear hybrid systems with BDD-like data structures. IEEE Trans. Soft. Eng. 31(1), 38–51 (2005)