Noname manuscript No.
(will be inserted by the editor)

SMT-based Scenario Verification for Hybrid Systems

A. Cimatti · S. Mover · S. Tonetta

Abstract Hybrid automata are a widely used framework to model complex critical systems, where continuous physical dynamics are combined with discrete transitions. The expressive power of Satisfiability Modulo Theories (SMT) solvers can be used to symbolically model networks of hybrid automata, using formulas in the theory of reals, and SAT-based verification algorithms, such as bounded model checking and k-induction, can be naturally lifted to the SMT case.

In this paper, we tackle the important problem of scenario-based verification, i.e. checking if a network of hybrid automata accepts some desired interactions among the components, expressed as Message Sequence Charts (MSCs). We propose a novel approach, that exploits the structure of the scenario to partition and drive the search, both for bounded model checking and k-induction. We also show how to obtain information explaining the reasons for infeasibility in the case of invalid scenarios.

The expressive power of the SMT framework allows us to exploit a local time semantics, where the timescales of the automata in the network are synchronized upon shared events. The approach fully leverages the advanced features of modern SMT solvers, such as incrementality, unsatisfiable core extraction, and interpolation. An experimental evaluation demonstrates the effectiveness of the approach in proving both feasibility and unfeasibility, and the adequacy of the automatically generated explanations.

Keywords SMT-based verification, network of hybrid automata, message sequence charts, bounded model checking, k-induction

1 Introduction

Complex Embedded Systems (CES) consist of software and hardware components that operate autonomous devices interacting with the physical environment. They are now part of our daily life and are used in many industrial sectors including automotive, aerospace, consumer electronics, communications, medical and manufacturing. CES are used to carry out highly complex and often critical functions. They are used to monitor and control industrial plants, complex transportation equipment, and communication infrastructure. The development process of CES is widely recognized as a highly complex task. A thorough validation and verification activity is necessary to enhance the quality of the CES and, in particular, to fulfill the quality criteria mandated by the relevant standards.

CES are composed of many heterogeneous components, interacting with external environments, and deal with continuous and discrete dynamics. Networks of communicating

Alessandro Cimatti, Sergio Mover, Stefano Tonetta (Fondazione Bruno Kessler, Trento, Italy)
E-mail: {cimatti,mover,tonettas}@fbk.eu

*This paper presents in a coherent and expanded form material that appears in the conference venues [11] and [13].

Hybrid Automata (HAs) [20] are increasingly used as a formal framework to model discrete and continuous components and their interaction: local activities of each component amount to transitions local to each hybrid automaton; communications and other events that are shared between/visible for various components are modeled as synchronizing transitions of the automata in the network; time elapse is modeled as shared timed transitions.

The framework of Satisfiability Modulo Theories (SMT) allows to symbolically model networks of *Linear HAs*, using the *Linear Real Arithmetic (LRA)* Theory. SAT-based verification algorithms, such as bounded model checking (BMC) and k-induction, can be naturally lifted to the SMT case in order to tackle reachability problems (see for instance [4]).

In this paper, we concentrate on the problem of scenario-based verification, checking if a network of hybrid automata accepts some desired interactions among the components. We use the language of Message Sequence Charts (MSCs) extended with constraints, 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.

In principle, the problem could be reduced to standard SMT-based verification, following an automata-based approach compiling the MSC into an observer. The approach proposed here, however, exploits the structure of the scenario to partition and drive the search, both for BMC and k-induction. We rely on the use of an alternative, local time semantics [6] for HAs, where each automaton can proceed based on its individual "local time scale", realigning its local clock on synchronizations with other automata. The local time semantics enables a "shallow synchronization" [8], where traces of the network are obtained composing traces of the local automata by superimposing structure based on shared communication.

In the case of BMC, we propose an encoding that 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. The algorithm proceeds by incrementally increasing the local paths between two consecutive islands and linking the local path to the next island by means of equalities. The k-induction algorithm, that is used to prove the MSCs unfeasibility, is specialized to the structure of the MSC, so that the "simple path" condition is localized to the fragments between the events, rather than being imposed globally on the whole network.

When a scenario is proved unfeasible, particularly in the case where the scenario is expected to be feasible, the end user is typically confronted with an "unsat" answer. In this paper, we also provide techniques to obtain information explaining the reasons for unfeasibility (e.g. which components are involved, which temporal restrictions between events in the MSC are too strong).

The SMT framework is a key enabler for the whole approach. In terms of expressiveness, the SMT language provides a natural symbolic representation for the local time semantics. In terms of search, the approach fully leverages the advanced features of modern SMT solvers, such as incrementality, unsatisfiable core extraction, and interpolation.

We implemented this approach within an extension of the NuSMV system, tightly integrated with the MathSAT SMT solver. We carried out an extensive evaluation, over a wide set of networks and benchmark MSCs. The tailored algorithms turn out to outperform the corresponding approaches based on MSC-to-automata construction, and the application of SMT-based techniques off-the-shelf. We also illustrate the effectiveness of the approach in automatically generating adequate explanations with detailed case studies.

The paper is structured as follows. In Section 2 we present some background on SMT-based verification. In Section 3 we discuss the framework of hybrid systems, and in Section 4 we present the scenario language. In Section 5 and in Section 6 we discuss the optimized

BMC and k-induction, respectively. In Section 7 we present the explanation techniques. In Section 8 we discuss the experimental results. Comparison with related works is presented in Section 9. In Section 10 we draw some conclusions and discuss future research directions.

2 SMT-based Verification of First-Order Transition Systems

2.1 Satisfiability Modulo Theories

Let Σ be a first-order *signature* containing predicates and function symbols with their arity. As in [17], in order to ease the presentation, we prefer not to use free variables in the formulas, but to represent them with 0-arity predicates (Boolean variables) and (uninterpreted) 0-arity functions. A *Σ-term* is built applying function symbols in Σ to Σ-terms. If p is a predicate with arity n and $t_1, \ldots, t_n$ are Σ-terms, then $p(t_1, \ldots, t_n)$ is a *Σ-atom.* A *Σ-formula* is a Σ-atom or a Boolean combination of Σ-atoms obtained using the usual Boolean connectives $\wedge, \vee, \neg$. In this paper we do not deal with quantified formulas (i.e. formulas obtained using the universal $\forall$ and existential $\exists$ quantifiers).

A first-order *Σ-theory* $\mathcal{T}$ is a set of first-order sentences with signature Σ, where a sentence is a Σ-formula without free variables. We assume that the symbols $=$, $\perp$, and $\top$ are part of the language, even if it is not explicitly contained in the signature, and are interpreted as the identity, false, and true, respectively. A *Σ-structure* $\mathcal{M}$ is a model of a Σ-theory $\mathcal{T}$ if $\mathcal{M}$ satisfies every sentence in $\mathcal{T}$. A Σ-formula is satisfiable in $\mathcal{T}$ ($\mathcal{T}$-satisfiable) if it is satisfiable in a model of $\mathcal{T}$. We write $\Gamma \models \phi$ to denote that the Σ-formula ϕ is a logical consequence of of a set of formulas Γ. *Satisfiability Modulo Theory* $\mathcal{T}$ (SMT($\mathcal{T}$)) is the problem of checking if a Σ-formula ϕ is satisfiable, for some background theory $\mathcal{T}$.

In this paper, we consider the theory of *Linear Arithmetic over Reals/Rationals*, $\mathcal{T}(\mathbb{Q})$, which uses the 0-arity function symbols $\{a\}_{a\in\mathbb{Q}}$, interpreted over the corresponding rational numbers, the unary function symbols $\{a\cdot\}_{a\in\mathbb{Q}}$, interpreted over the multiplication of the argument by the rational number a, a set of 0-arity uninterpreted function symbols, and the binary function symbols and predicate symbols $+, <, \leq, >, \geq, \neq$, interpreted with the corresponding operations and relations over the rationals. The resulting language consists of quantifier-free Boolean combinations of atoms in the form $\sum a_i \cdot x_j \bowtie a$, where x_j is a 0-arity uninterpreted function symbol, $a_i, a \in \mathbb{Q}$ and $\bowtie \in \{<, \leq, >, \geq, \neq\}$.

A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. It is well known that given a non-CNF formula we can obtain an equi-satisfiable CNF formula in polynomial time. The *unsatisfiable core* for an unsatisfiable CNF formula ϕ is a formula ψ such that ψ is unsatisfiable and $\phi = \psi \wedge \psi'$, for a (possibly empty) formula ψ'.

Given two formulas A and B, with $A \wedge B \models \perp$, the Craig *interpolant* of $A \wedge B$ is a formula I such that $\models A \rightarrow I$, $B \wedge I \models \perp$, and every uninterpreted symbol of I occurs both in A and B. Intuitively, the interpolant is an over-approximation of A “guided” by B.

SMT solvers are tools to check if a first-order formula is satisfiable modulo a background theory. They often provide functionalities to construct models and proofs of unsatisfiability, and to extract unsatisfiable cores and Craig interpolants. Most modern SMT solvers also feature an *incremental* interface, i.e. they are able to tackle sequences of satisfiability problems efficiently, by reusing theory information discovered during the previous searches.

We assume that the sequences of problems have the following form, where each problem may inherit from the preceding one the variables and a substantial subset of sub-formulas:
$\gamma(0) \wedge \beta(0)$
$\gamma(0) \wedge \gamma(1) \wedge \beta(1)$

$\gamma(0) \wedge \gamma(1) \wedge \gamma(2) \wedge \beta(2)$
...

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 $\gamma(0), \ldots, \gamma(k)$.

2.2 First-Order Transition Systems (FOTS)

Given a first-order signature Σ, Σ' is the signature obtained by replacing each symbol s in Σ with a copy s'. A *first-order transition system* is a tuple $S = \langle V, \mathcal{W}, I, Z, T \rangle$ such that:

- V is a first-order signature;
- $\mathcal{W}$ is a first-order signature disjoint from V and V';
- I is a first-order formula over V (called initial condition);
- Z is a first-order formula over V (called invariant condition);
- T is a first-order formula over $V \cup \mathcal{W} \cup V'$ (called transition condition).

Given a theory $\mathcal{T}$ over the signature $V \cup \mathcal{W}$, a *$\mathcal{T}$-sequence* is a sequence of $\mathcal{T}$-models $s_0; a_1; s_1; \ldots; a_k; s_k$ with the same domain, alternating states s_i and inputs a_i. A $\mathcal{T}$-sequence $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ is a model of the transition system $S = \langle V, \mathcal{W}, I, Z, T \rangle$ iff:

- s_0 satisfies I;
- for every $0 \leq i \leq k$, s_i satisfies Z;
- for every $0 \leq i < k$, $s_i \cup a_{i+1} \cup s'_{i+1}$ satisfies T, where the interpretation of a symbol in V' by s'_{i+1} is the same as the interpretation of the "unprimed" version by s_{i+1}.

We say that π is a *path* of the FOTS S and that the sequence of inputs $w = a_1; \ldots; a_k$ is a *trace* of S. A path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ accepts the trace $w = a_1; \ldots; a_k$. Given a first-order formula ϕ over input variables $\mathcal{W}$, the projection $w_{|\phi}$ of w on ϕ is the sub-trace of w obtained by removing all a_i in w such that $a_i \not\models \phi$. In practice, the sub-trace is restricted to the assignments of the formula ϕ, which define a set of inputs of the FOTS.

Remark 1 In order to keep the notation as simple as possible, the above definition does not distinguish symbols that are rigid and interpreted by $\mathcal{T}$ from the actual variables of the system. We assume that such symbols do not occur as "primed" in the transition condition although their interpretation is the same in all states of a sequence.

2.3 SMT-based Bounded Model Checking of FOTS

Given a set of symbols V, we denote with $V^i = \{s^i | s \in V\}$ the set of copies indexed with i. Given a formula $\psi(V)$ over the variables V, we denote with $\psi(V^i)$ (or simply ψ^i when V is clear from the context) the formula obtained by replacing all the variables $v \in V$ with v^i. Given a formula $\psi(V, \mathcal{W}, V')$ over the variables $V, \mathcal{W}, V'$, we denote with ψ^i the formula obtained replacing all the variables $v \in V$ with v^i, $w \in \mathcal{W}$ with w^i, and $v' \in V'$ with v^{i+1}.

Given a FOTS $S = \langle V, \mathcal{W}, I, Z, T \rangle$ and a target property *target* over the variables in V, the *Bounded Model Checking* (BMC) problem consists of determining if *target* is reachable in S in k steps (i.e. if there is a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ in S such that $s_k \models$ *target*). We encode the BMC problem as follows:

$$\mathit{path}(k) := I^0 \wedge \bigwedge_{i=0}^{k} Z^i \wedge \bigwedge_{i=0}^{k-1} T^i, \;\; BMC(k) := \mathit{path}(k) \wedge \mathit{target}^k$$

The formula $BMC(k)$ is satisfiable iff there exists a path from the initial state of S to a state in *target* of length k. An SMT solver is used to query for the satisfiability of $BMC(k)$.

When we solve the reachability problem incrementally, we look for a k such that the formula $BMC(k)$ is satisfiable. The problem is presented to the solver in the following form: $\gamma(0) := I^0 \wedge Z^0$, $\gamma(i) := T^{i-1} \wedge Z^i$, for $i > 0$, and $\beta(i) := \mathit{target}^i$, for $i \geq 0$.

2.4 SMT-based K-induction of FOTSs

K-induction [35] 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, 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 the target set of states cannot be reached in $k+1$ steps by a path $s_0, \ldots, s_{k+1}$, where the target does not hold in s_i, for $i < k+1$. These checks can be solved by means of satisfiability. Hereafter, we will consider only the first condition because it is more effective for the problem of MSC feasibility, but the whole framework can be easily extended to use both conditions.

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 $\mathit{kind}(k) := \mathit{path}(k+1) \wedge \mathit{simple}(k+1)$ 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 FOTS, there is a k for which the above conditions are unsatisfiable. In hybrid systems, it is very common that the FOTSs contain infinite paths, typically with monotonically increasing variables (such as the local time) and, therefore, it is difficult to apply k-induction.

In [36], k-induction has been integrated with predicate abstraction [18] to deal with infinite-state systems. Typically, an abstraction defines an equivalence relation EQ_α among 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 $\mathit{path}_\alpha(k) := I(\overline{V^0}) \wedge EQ_\alpha(\overline{V^0}, V^0) \wedge \bigwedge_{1 \leq h \leq k} (T(V^{h-1}, \mathcal{W}^h, \overline{V^h}) \wedge EQ_\alpha(\overline{V^h}, V^h)) \wedge \bigwedge_{0 \leq h \leq k} (Z(V^h) \wedge Z(\overline{V}^h))$ is satisfiable iff there exists an initial path of k steps in the abstract state space. The formula $\mathit{simple}_\alpha(k)$ is defined as $\mathit{simple}_\alpha(k) := \bigwedge_{0 \leq i < j \leq k} \neg EQ_\alpha(V^i, V^j)$. Finally, the formula kind_α, defined as $\mathit{kind}_\alpha(k) := \mathit{path}_\alpha(k) \wedge \mathit{simple}_\alpha(k)$, is satisfiable iff there exists an initial simple path of length k.

Similarly to the concrete case, if, for all $i \leq k$, $\mathit{path}_\alpha(k) \wedge \mathit{target}^k$ is unsatisfiable and $\mathit{kind}_\alpha(k)$ is unsatisfiable as well, then *target* is not reachable in the abstraction (and therefore neither in the concrete state space).

3 SMT-based Verification of Hybrid Automata Networks

3.1 Hybrid Automata Networks (HAN)

A *Hybrid Automaton* (HA) [20] 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 $\dot{X}$ represents the derivative of the 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 condition of a LHA is a conjunction of inequalities.

A *network* $\mathcal{N}$ of HAs is the parallel composition of two or more HAs. In the following, we consider a network $\mathcal{N} = H_1 || \dots || H_n$ 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). Also, we denote with τ_i the set of local events of the i-th component, i.e., $\tau_i = A_i \setminus \bigcup_{j \neq i} A_j$.

3.2 FOTS-based Semantics of HAN

We consider the *local-time semantics* of [6], where time progresses in each component with a local scale by enriching all shared events with time-stamps and synchronizing the components on shared events forcing the time-stamps to be equal.

We define the semantics of a network of LHAs in terms of the FOTS $S_{\textsc{LocTime}}(\mathcal{N})$. More specifically, we associate to a network $\mathcal{N} = H_1 || \dots || H_n$ a set of FOTSs $S_1, \dots, S_n$, with $S_i = \langle V_i, W_i, I_i, Z_i, T_i \rangle$, and a synchronization constraint $\textsc{Sync}$ over the union of the V_i and W_i. $S_{\textsc{LocTime}}(\mathcal{N}) = \langle V, W, I, Z, T \rangle$ where: $V = \bigcup_i V_i$; $W = \bigcup_i W_i$; $I = \bigwedge_i I_i$; $Z = \bigwedge_i Z_i$; $T = \bigwedge_i T_i \wedge \textsc{Sync}$.

The definition of the FOTS S_i uses the following additional variables: a set of Boolean variables $\{l_{(i,1)}, \dots, l_{(i,\lceil \log(|Q_i|) \rceil)}\}$ that represents the current location of H_i; a set of Boolean variables $\{a_{(i,1)}, \dots, a_{(i,\lceil \log(|A_i \cup \{\textsc{t},\textsc{s}\}|) \rceil)}\}$ that represents the set of events taken by H_i at the current step, with two distinguished values $\textsc{t}$ and $\textsc{s}$, representing a timed transition and stuttering, respectively; an additional local real-valued variable t_i to track the total time elapsed in H_i. An assignment to the variables $\{l_{(i,1)}, \dots, l_{(i,\lceil \log(|Q_i|) \rceil)}\}$ determines the current location q of H_i. For clarity, we will denote this assignment with $loc_i = q$. Similarly, we will write $\varepsilon_i = a$ for the assignment to the variables $\{a_{(i,1)}, \dots, a_{(i,\lceil \log(|A_i \cup \{\textsc{t},\textsc{s}\}|) \rceil)}\}$ used to encode that the current event is a. If E is a set of events, we denote with $\varepsilon_i \in E$ the formula over these variables encoding all the assignments corresponding to the events in E.

The elements of S_i are defined as follows:

- $V_i := \{l_{(i,1)}, \dots, l_{(i,\lceil \log(|Q_i|) \rceil)}\} \cup \{t_i\} \cup X_i$;

- $W_i := \{a_{(i,1)}, \ldots, a_{(i,\lceil \log(|A_i \cup \{\mathrm{T},\mathrm{S}\}|)\rceil)}\}$;
- $I_i := t_i = 0 \wedge \bigwedge_{q \in Q_i}(loc_i = q \rightarrow \iota(q)(X_i))$;
- $Z_i := \bigwedge_{q \in Q_i}(loc_i = q \rightarrow \xi(q)(X_i))$;
- $T_i := \textsc{Stutter}_i \vee \bigwedge_{q \in Q_i}(loc_i = q \rightarrow (\textsc{Timed}_{i,q} \vee \bigvee_{\langle q,a,q'\rangle \in R_i} \textsc{Untimed}_{i,\langle q,a,q'\rangle}))$ with
  - $\textsc{Stutter}_i := \varepsilon_i = \mathrm{S} \wedge t_i = t_i' \wedge loc_i' = loc_i \wedge \bigwedge_{x \in X_i} x' = x$;
  - $\textsc{Timed}_{i,q} := \varepsilon_i = \mathrm{T} \wedge loc_i' = loc_i \wedge t_i' > t_i \wedge \overline{\mu_i(q)}^{(t_i'-t_i)}(X_i, X_i')$
  - $\textsc{Untimed}_{i,\langle q,a,q'\rangle} := \varepsilon_i = a \wedge loc_i' = q' \wedge t_i' = t_i \wedge \theta_i(\langle q,a,q'\rangle)(X_i, X_i')$

  where $\overline{\mu_i(q)}^{(t_i'-t_i)}$ is a formula in LRA over X_i and X_i'. Recall that $\overline{\mu_i(q)}$ is of the form $\sum a \cdot \dot{x} \bowtie b$, for some $a \in \mathbb{R}, x \in X_i, \bowtie \in \{<, \leq, >, \geq, =, \neq\}$. We replace each $\dot{x}$ in $\mu_i(q)$ by $\frac{x_i'-x_i}{t_i'-t_i}$, and we remove the denominator since $t_i' > t_i$, obtaining $\overline{\mu_i(q)}^{(t_i'-t_i)} := \sum a \cdot (x_i' - x_i) \bowtie (t_i' - t_i) \cdot b$.

The formula $\textsc{Sync}$ is defined as follows:

$$\begin{aligned}\textsc{Sync} := \bigwedge_{1 \leq j < h \leq n} & \bigwedge_{a \in A_j \cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \wedge (\varepsilon_j = a \rightarrow t_j = t_h) \\ & \wedge \bigwedge_{a \in (A_j \setminus A_h) \cup \{\mathrm{T}\}} (\varepsilon_j = a \rightarrow \varepsilon_h = \mathrm{S}) \\ & \wedge \bigwedge_{a \in (A_h \setminus A_j) \cup \{\mathrm{T}\}} (\varepsilon_h = a \rightarrow \varepsilon_j = \mathrm{S})\end{aligned}$$

A variant, called *step semantics*, allows to have independent transitions in parallel:

$$\textsc{Sync}_{step} := \bigwedge_{1 \leq j < h \leq n} \bigwedge_{a \in A_j \cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \wedge (\varepsilon_j = a \rightarrow t_j = t_h)$$

We say that a state s_k of $S_{\textsc{LocTime}}(\mathcal{N})$ is *synchronized* iff for $1 \leq i < j \leq n$, $t_i = t_j$, i.e., the local times are equal. We say that a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ of the FOTS $S_{\textsc{LocTime}}(\mathcal{N})$ is *synchronized* if s_k is a synchronized state.

The paths of the network $\mathcal{N}$ are given by the synchronized $\mathcal{T}$-paths of $S_{\textsc{LocTime}}(\mathcal{N})$, where $\mathcal{T}$ is the theory of LRA.

3.3 Specialized BMC

In order to solve the reachability problem in a network of LHAs, we can use BMC and unroll the transition relation for a bounded number of steps. However, this requires each component to perform the same number of steps in order to reach the target. If some components can reach the target in less steps, they are forced to stutter.

In [8], a more *shallow synchronization* is required among the components resulting in local runs free of stuttering. Basically, instead of unrolling the composition of the S_i, the BMC encoding of each component is built with an independent bound. A global (often more complex) synchronization constraint guarantees that the sequence of events and their time-stamps are compliant with the semantics of the composition.

Another interesting optimization is the *alternation of discrete and timed transitions*, first proposed in [4]. The idea consists of using different transition conditions for odd and even steps during the unrolling. In particular, $T_{\mathrm{T},i} := \bigwedge_{q \in Q_i}(loc_i = q \rightarrow (\textsc{Stutter}_i \vee \textsc{Timed}_{i,q}))$ can be used in even steps, while $T_{\mathrm{D},i} := \bigwedge_{q \in Q_i}(loc_i = q \rightarrow (\textsc{Stutter}_i \vee \bigvee_{\langle q,a,q' \rangle \in R_i} \textsc{Untimed}_{i,\langle q,a,q' \rangle}))$ in odd steps.

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.

However, in order to enhance k-induction with alternation, the encoding of stutter transitions, which usually are not allowed when checking the loop-free condition, must be taken into account. $T_{\mathrm{T},i}$ must allow to stutter: if it is not the case, then the alternation imposed in the encoding will not allow to execute two consecutive discrete transitions, where time does not elapse. Thus, the loop-free condition of k-induction must be relaxed in order to allow stutter in $T_{\mathrm{T},i}$. $T_{\mathrm{D},i}$ must not stutter: stutter transitions in the odd steps make the alternation ineffective, since they allow infinite loop-free paths. Avoiding stutter does not prevent to visit all the reachable states: a single timed transition can reach the same states reached by two timed transitions interleaved with a stutter transition, covering a bigger time elapse.

4 Scenario Verification for HAN

In order to support user validation, it is very important to be able to check whether a HAN may exhibit behaviors that satisfy a certain scenario, specifying some desired or undesired interactions among the components. We use the language of Message Sequence Charts (MSCs) [22] and its extensions to express scenarios of such interactions.

An MSC defines a single (partial-order) interaction of the components of a network $\mathcal{N} = H_1 || \dots || H_n$. 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 $A_m \subseteq A_{\mathcal{N}}$, where $A_{\mathcal{N}} = \bigcup_{1 \leq i \leq n} A_i$ is the set of all the events of the network $\mathcal{N}$. The typical implicit assumption is that the set A_m contains all the synchronization events of the network. Since we are dealing with networks of hybrid automata 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 if $\mathcal{N}$ is a network of the hybrid automata $H_1, \dots, H_n$ with alphabet respectively $A_1, \dots A_n$, then $A_m = \bigcup_{1 \leq i < j \leq n} A_i \cap A_j$ and thus the timed event is not part of A_m[1].

The MSC defines a sequence of events for every component H_i of the network, called *instance* of H_i. An instance σ_i of H_i is a sequence $a_1; \dots; a_l \in (A_m \cap A_i)^*$ of events of H_i. We denote the j-th event a_j of the instance σ_i with $\sigma_i[j]$ and the length of σ_i with $|\sigma_i|$. Along each instance line σ_i there is a finite set of *local segments* $\{lsg(\sigma_i[0]), \dots, lsg(\sigma_i[l])\}$ which denote the position between two consecutive events: $lsg(\sigma_i[j])$ is the local segment between the j-th event and the $j+1$-th event of σ_i. The first local segment from the beginning of the instance to the first event is $lsg(\sigma_i[0])$ and the final local segment after the $|\sigma_i|$-th event is $lsg(\sigma_i[l])$.

H_i accepts the instance σ_i with respect to A_m iff the FOTS S_i of H_i *accepts σ_i with respect to A_m* ($S_i \models^m \sigma_i$). $S_i \models^m \sigma_i$ iff there exists a trace w accepted by S_i such that the sub-sequence of events in A_m of w is equal to σ_i (i.e. $w_{|\bigvee_{a \in A_m} \varepsilon = a} = \sigma_i$). In this case we say that w is *compatible* with σ_i. In other words, S_i accepts the instance with respect to

[1] The techniques presented in this paper can be adapted to handle synchronizations wich are not in A_m.

A_m iff there exists a path π of S_i over a trace compatible with σ_i. In such cases, we write $\pi \models^m \sigma_i$.

If $\pi \models^m \sigma$, π must be in the form $s_0; \varepsilon = \tau; \ldots; \varepsilon = \tau; s_{h_1}; \varepsilon = \sigma[1]; s_{(h_1+1)}; \varepsilon = \tau; \ldots; \varepsilon = \tau; s_{h_{|\sigma|}}; \varepsilon = \sigma[|\sigma|] s_{(h_{|\sigma|}+1)}; \varepsilon = \tau; \ldots; \varepsilon = \tau; s_{h_{(|\sigma|+1)}}$, where s_h is a model over the state variables V_i of S_i and τ are local events of H_i. We denote the sub-sequences of the path π in which it is split by σ as follows:

- $pre_j(\pi) = s_{h_j}$ is the source state of the transition labeled with $\varepsilon = \sigma[j]$ in π.
- $post_j(\pi) = s_{h_j+1}$ is the destination state of the transition labeled with $\varepsilon = \sigma[j]$ in π.
- $loc_j(\pi) = s_{h_j+1}; \ldots; s_{h_{j+1}}$ 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 $\sigma_1 || \ldots || \sigma_n$ where σ_i is an instance of H_i. 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 $A = A_i \cap A_j$, $\sigma_{i|A} = \sigma_{j|A}$. Henceforth, we assume that the MSCs are consistent.

The network $\mathcal{N}$ accepts the MSC m iff the FOTS $S_{\text{LocTime}}(\mathcal{N}) = S_1 || \ldots || S_n$ accepts m ($S_{\text{LocTime}}(\mathcal{N}) \models m$). $S_{\text{LocTime}}(\mathcal{N}) \models m$ iff there exists a trace w accepted by $S_{\text{LocTime}}(\mathcal{N})$ such that, for every S_i, the sub-sequence of events in $A_m \cap A_i$ is equal to σ_i ($w_{|(\bigvee_{a \in A_m} \varepsilon = a)} = \sigma_i$). In other words, $S_{\text{LocTime}}(\mathcal{N})$ accepts the MSC m iff there exists a path of $S_{\text{LocTime}}(\mathcal{N})$ over a trace compatible with every instance of the MSC.

$V_m := \bigcup_{1 \le i \le n} V_{\sigma_i}$ is the set of variables of the CMSC m, where for all $1 \le i \le n$, $V_{\sigma_i} := \bigcup_{0 \le j \le (|\sigma_i|+1)} V_i^j$ (e.g. v_i^j represents the value of the variable v_i of the i-th component just before the j-th event $\sigma_i[j]$ of σ_i). We define a *Constrained MSC (CMSC)* as a pair $\langle m, \phi \rangle$ where m is an MSC $\sigma_1 || \ldots || \sigma_n$, $\phi = \phi_0 \wedge \phi_1 \wedge \ldots \wedge \phi_n$, ϕ_0 is a formula over V_m and for all $1 \le i \le n$, ϕ_i is a formula over V_i^j. Given a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ of $\mathcal{N}_{\text{LocTime}}(\mathcal{H})$, the *projection* of π over S_i is the path $prj(\pi, i)$ obtained projecting the states over the S_i-th component and removing all the transitions over events which are not in A_i. $S_{\text{LocTime}}(\mathcal{N}) \models \langle m, \phi \rangle$ iff there exists a path π of $S_{\text{LocTime}}(\mathcal{N})$ such that $\bigcup_{1 \le i \le n, 0 \le j \le |\sigma_i|} pre_j(prj(\pi, i)) \models \phi$.

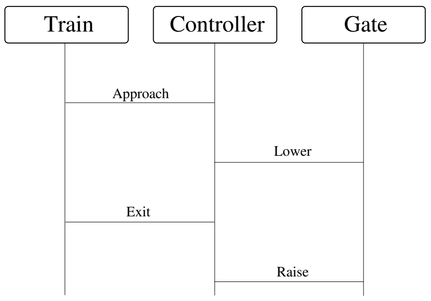


Fig. 1 An MSC for the Train-Gate-Controller [20].

Example 1 Figure 1 shows an MSC for the railroad model from [20]. 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*.

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 that, composed with $S_{\text{LocTime}}(\mathcal{N})$, forces $S_{\text{LocTime}}(\mathcal{N})$ to follow only paths that satisfy the MSC. It is in spirit similar to the automata-approach to LTL model checking [37]. The SMT-based verification techniques are applied off the shelf on the resulting FOTS. The monitor can be one additional component in the network or consist of many components one for each instance of the CMSC. Exploiting local-time semantics, the monitor can also be reduced to follow one interleaving of the partial-order reduction defined by the CMSC. However, the experimental analysis of [11] shows that the best option is to use a monitor per component.

5 Scenario-driven BMC

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. The encoding is conceived in order to maximize the incrementality of the solver, as described in Section 2, along the increase of the number of local steps. The idea is that we keep the encodings of the sequences of local transitions separated from the encoding of the encoding of the next shared event, and we unroll them incrementally, while we add and remove accordingly the equality constraints which glue such sequences. Note that the encoding of a sequence of local transitions does not fix the exact number of local transitions, since we allow the stutter action. This because we do not known a priori what will be the exact length of each local path.

Let us consider a network $\mathcal{N} = H_1 || \dots || H_n$ of LHAs and the correspondent FOTS $S_i = \langle V_i, \mathcal{W}_i, I_i, Z_i, T_i \rangle$, representing 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 $T_{i|\varepsilon \in \tau_i \cup \{\mathrm{S},\mathrm{T}\}}$ with $T_{i|\tau}$. We associate a bound $k_i[j]$ to the j-th segment $\mathit{lsg}(\sigma_i[j])$ of the i-th instance. $k_i[j]$ is used to limit the number of transitions in the local path $\mathit{loc}_j(\pi)$ of a path π satisfying the instance σ_i. We use $\overline{k}_i$ to denote $\langle k_i[0], \dots, k_i[h_{|\sigma_i|}] \rangle$ and $\overline{k}$ to denote $\langle \overline{k}_1, \dots, \overline{k}_n \rangle$. Moreover, for all $1 \leq i \leq n$ and $0 \leq j \leq |\sigma_i|$, we define the index $\mathit{idx}_i[j]$ such that $\mathit{idx}_i[0] = -1$ and if $\langle i, j \rangle \neq \langle i, j' \rangle$ then $\mathit{idx}_i[j] + h \neq \mathit{idx}_i[j'] + h'$ for all h, h' with $0 \leq h \leq k_i[j]$ and $0 \leq h' \leq k_i[j']$ (i.e. the indexes of two different segments do not overlap).

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):

$$
\begin{aligned}
\mathit{enc}(m, \overline{k}) &:= \bigwedge_{1 \leq i \leq n} \mathit{enc}(\sigma_i, \overline{k}_i) \wedge \bigwedge_{1 \leq j < i \leq n} \mathit{sync}(\sigma_j, \sigma_i) \wedge \mathit{finalsync}(m, \overline{k}) \\
\mathit{enc}(\sigma_i, \overline{k}_i) &:= I_i^0 \wedge Z_i^0 \wedge \mathit{enc}(\sigma_i, k_i[0]) \wedge \bigwedge_{1 \leq j \leq |\sigma_i|} (V^{\mathit{idx}_i[j-1] + k_i[j-1] + 1} = V^{\mathit{idx}_i[j]} \wedge \\
&\qquad T_{i|\sigma_i[j]}^{\mathit{idx}_i[j]} \wedge Z_i^{\mathit{idx}_i[j]} \wedge \mathit{enc}(\sigma_i, k_i[j])) \\
\mathit{enc}(\sigma_i, k_i[j]) &:= \bigwedge_{1 \leq h \leq k_i[j]} (T_{i|\tau}^{\mathit{idx}_i[j]+h} \wedge Z_i^{\mathit{idx}_i[j]+h}) \\
\mathit{sync}(\sigma_j, \sigma_i) &:= \bigwedge_{1 \leq z \leq |\sigma_{j|A_{\{i,j\}}}| = |\sigma_{i|A_{\{i,j\}}}|} t_i^{\mathit{idx}_i[f_i^{ij}(z)]} = t_j^{\mathit{idx}_j[f_j^{ij}(z)]} \\
\mathit{finalsync}(m, \overline{k}) &:= \bigwedge_{1 \leq i < n, j = i+1} (t_i^{\mathit{idx}_i[|\sigma_i|] + k_i[|\sigma_i|] + 1} = t_j^{\mathit{idx}_j[|\sigma_j|] + k_j[|\sigma_j|] + 1})
\end{aligned}
$$

where $A_{\{i,j\}} = A_i \cap A_j$ 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 $\sigma_{j|_A} = \sigma_{i|_A} = a_1; \ldots a_l$, then $f_i^{ij}, f_j^{ij} : \mathbb{N} \rightarrow \mathbb{N}$ 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 $\langle m, \phi \rangle$ we have just to conjoin the additional constraints ϕ:

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

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 $\mathcal{N} \models \langle m, \phi \rangle$. Vice versa, if $\mathcal{N} \models \langle m, \phi \rangle$, then there exist integers $\overline{k}$ such that $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable.*

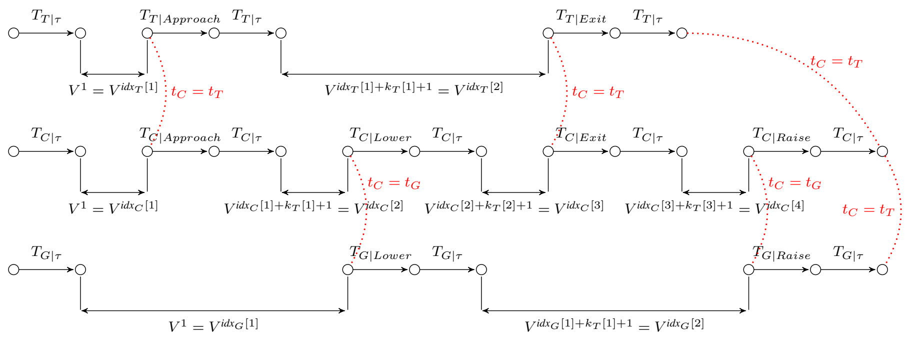


Fig. 2 Encoding for the Train-Gate-Controller MSC fixing to 1 the length of all the local bounds.

Example 2 The Figure 2 shows the encoding of the MSC of Example 1, fixing the length of all the local steps to 1 (we abbreviated the name of the components with their initial letter). Each sequence of states represents the encoding for a single instance of the MSC, the double arrows represent the links of the local segments with the shared events, and the dotted lines represent the equalities over the local time variables.

In the following, we detail how we increase the bound of the local transitions incrementally adding new constraints to the solver. We consider the same number k of local steps for each segment of the CMSC. This simplifies the algorithm because it is not necessary to decide which segment to increment and reduces the number of calls to the SMT solver.

With regard to the formulas introduced in Section 2, we define the partial encoding for an instance σ_i as follows:

$$\gamma_{enc(\sigma_i)}(0) := I_i^0 \wedge Z_i^0 \wedge \bigwedge_{1 \leq j \leq |\sigma_i|} T_{i|\sigma_i[j]}^{idx_i[j]} \wedge Z_i^{idx_i[j]}$$

$$\gamma_{enc(\sigma_i)}(k) := \bigwedge_{0 \leq j \leq |\sigma_i|} T_{i|\tau}^{idx_i[j]+k} Z_i^{idx_i[j]+k}$$

$$\beta_{enc(\sigma_i)}(k) := \bigwedge_{0 \leq j < |\sigma_i|} V^{idx_i[j]+k+1} = V^{idx_i[j+1]}$$

For each instance σ_i we encode the initial condition and all the $|\sigma_i|$ 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:

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

6 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 Section 5. For the inductive step, we apply the simple path condition to each segment of the scenario. The use of different local bounds as presented in Section 5 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.

The goal is to find an inductive condition $kind(\langle m,\phi\rangle,\overline{k})$ such that, if $kind(\langle m,\phi\rangle,\overline{k})$ and $enc(\langle m,\phi\rangle,\overline{k})$ are unsatisfiable for some $\overline{k}$, then $\mathcal{N}\not\models\langle m,\phi\rangle$. It is not possible to apply the simple path condition independently to each segment of the encoding. There exists two main difficulties. The first is that the projection of a simple path on a component may not be a simple path. The second is that 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 $\leq_m$ among the segments of m defined as the transitive closure of the smallest relation such that:

– $lsg(\sigma_i[j]) \leq_m lsg(\sigma_i[j'])$ if $0\leq j\leq j'\leq|\sigma_i|$;
– $lsg(\sigma_i[j]) \leq_m lsg(\sigma_{i'}[j'])$ if $lsg(\sigma_i[j]) = lsg(\sigma_{i'}[j'])$ or there exists a $lsg(\sigma_{i''}[j''])$ such that there is a synchronization between $\sigma_i[j]$ and $\sigma_{i''}[j'']$ and $lsg(\sigma_{i''}[j''])\leq_m lsg(\sigma_{i'}[j'])$.

Given a CMSC $\langle m,\phi\rangle$ and the local segment $lsg(\sigma_i[j])$, $\langle\overline{m}_i[j],\overline{\phi}_i[j]\rangle$ is partial CMSC 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 $lsg(\overline{\sigma}_v[z])\leq_m lsg(\sigma_i[j])$ or $lsg(\overline{\sigma}_v[z]) = lsg(\sigma_i[j])$, while for all $|\overline{\sigma}_v|<z\leq|\sigma_v|$ $lsg(\sigma_v[z])\not\leq_m lsg(\sigma_i[j])$.
– $\overline{\phi}_i[j]$ is the conjunction of all the conjuncts of ϕ which are over variables in $\overline{m}_i[j]$.

We define the local simple path condition as follows:

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

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 $\leq_m$. We incrementally apply the partitioned simple path condition to the local segments of m. The incremental checks exploit the incremental interface of the solver.

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 there exists a loop in the local structure of the automaton (i.e. a loop where all the transitions are labelled with a local event), due to the infinite nature of the state space we may have an infinite path in the system without loops, so that the simple-path condition never holds. In order to prove the unfeasibility of scenarios also in these cases, we combine k-induction with predicate abstraction as described in Section 2.4. 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 abstracts away the continuous components only where necessary.

7 Unfeasibility Explanation

In the case the CMSC $\langle m,\phi\rangle$ is unfeasible in the network $\mathcal{N}$ we provide the user with explanations which help to identify the reasons of the unfeasibility of the CMSC. In this section we first describe what kind of explanations we provide, their formalization and how we compute them. Then, we present three different case studies where we use the unfeasibility explanation to infer the cause of the unfeasibility of the scenario.

We identify the following three types of explanations:

1. an unfeasible prefix of the CMSC, which reduces the number of events and constraints to be inspected by a user to find a bug (either in the scenario or in the network);
2. why the paths of the network consistent with m cannot satisfy ϕ. The explanation is a formula that helps the user to understand the behaviours of $\mathcal{N}$ that are not consistent with ϕ;
3. why the paths of a component consistent with the corresponding instance of m are not consistent with the rest of m: this formula helps the user in detecting if a component is involved in the unfeasibility and, in that case, what synchronization constraints are not consistent with the other components in the network.

A *prefix* of a CMSC $\langle m,\phi\rangle$ is a CMSC $\langle\overline{m},\overline{\phi}\rangle$ such that: $\overline{m}:=\overline{\sigma}_1||\ldots||\overline{\sigma}_n$ and m is consistent; for all $1\leq i\leq n$, $|\overline{\sigma}_i|\leq|\sigma_i|$ and for all $1\leq j\leq|\overline{\sigma_i}|$, $\overline{\sigma}_i[j]=\sigma_i[j]$; $\overline{\phi}$ contains only the conjunct of ϕ over the variables $V_{\overline{m}}$.

Given a CMSC $\langle m,\phi\rangle$ and a network $\mathcal{N}$ such that $\mathcal{N}\not\models\langle m,\phi\rangle$, the unfeasibility explanation of the *first type* is an unfeasible CMSC prefix $\langle\overline{m},\overline{\phi}\rangle$ of $\langle m,\phi\rangle$.

Given a path $\pi_i\models\sigma_i$ and a tuple of bounds $\overline{k}_i:=\langle k_i[0],\ldots,k_i[|\sigma_i|]\rangle$, where $k_i[j]$ is the length of a local segment, we say that π_i is *bounded by* $\overline{k}_i$ if for all $0\leq j\leq|\sigma_i|$ the length of *lsg*$(\sigma_i[j])$ in π_i is $k_i[j]$. Given a path π of $S_{\text{LocTime}}(\mathcal{N})$ compatible with m and $\overline{k}:=\langle\overline{k}_1,\ldots,\overline{k}_n\rangle$, where $\overline{k}_i$ is a tuple of bounds, we say that π is *bounded by* $\overline{k}$ iff for all $1\leq i\leq n$, $prj(\pi,i)$ is bounded by $\overline{k}_i$.

Given a CMSC $\langle m,\phi\rangle$ unfeasible in $\mathcal{N}$ and bounds $\overline{k}$, an explanation of the *second type* is a formula ψ over V_m such that $\mathcal{N}\parallel m\models_{\overline{k}}\psi$ and $\psi\wedge\phi\models\bot$, where $\mathcal{N}\parallel m\models_{\overline{k}}\psi$ iff for all π of $S_{\text{LocTime}}(\mathcal{N})$ bounded by $\overline{k}$, $\bigcup_{1\leq i\leq n,0\leq j\leq|\sigma_i|+1}pre_j(prj(\pi,i))\models\psi$.

Given a CMSC $\langle m, \phi \rangle$ unfeasible in $\mathcal{N}$ and bounds $\overline{k}$, an explanation of the *third type* is a formula ψ_i over V_{σ_i} such that $S_i \parallel \sigma_{i_{|\phi_i}} \models_{\overline{k}} \psi_i$ and $\parallel_{j \neq i} S_j \parallel \sigma_{j_{|\phi_j}} \models_{\overline{k}} \neg\psi_i$. $S_i \parallel \sigma_{i_{|\phi_i}} \models_{\overline{k}} \psi_i$ iff for all paths π_i of S_i bounded by $\overline{k}_i$ $\bigcup_{0 \leq j \leq |\sigma_i|+1} pre_j(\pi_i) \models \psi_i$ and $\parallel_{i \neq j} S_j \parallel \sigma_{j_{|\phi_j}} \models_{\overline{k}} \neg\phi$ iff for $i \neq j$, for all π_j bounded by $\overline{k}_j$, $\bigcup_{i \neq j, 0 \leq v \leq |\sigma_j|+1} pre_v(\pi_j) \models \neg\psi_i$.

We extract the explanations by exploiting both unsatisfiable cores and interpolation. In particular, when we perform the inductive check incrementally on the CMSC $\langle m, \phi \rangle$ we run a satisfiability check after we have encoded all the synchronizations in the same partial order defined by $\leq_m$. If the encoding is unsatisfiable, by Theorem 2 we known that $\langle m, \phi \rangle$ is unfeasible. We extract the unsatisfiable core ξ of the encoding and different interpolants from the same proof of unsatisfiability.

In order to compute an unfeasible CMSC prefix $\langle \overline{m}, \overline{\phi} \rangle$, we use the unsatisfiable core ξ. ξ contains a subset of the encoding of the local paths, events, and constraints, since they are encoded in different conjuncts. Thus, ξ is fine-grained enough to obtain a precise subset X of the elements of the CMSC. For example, if the formula $T^{idx_i[j]}_{i|\sigma_i[j]} \wedge Z^{idx_i[j]}_i$ is in ξ, it means that the encoding of the event $\sigma_i[j]$ is in X. The unfeasible CMSC prefix $\langle \overline{m}, \overline{\phi} \rangle$ is obtained taking all the elements of $\langle m, \phi \rangle$ that are in the relation $\leq_m$ with an element of X.

We compute the explanation of the second type ψ using interpolation. We partition the formulas in the unsatisfiable core ξ in two different formulas, A and B. A is the conjunction of all the formulas of ξ which encodes the parallel composition of the network $\mathcal{N}$ and the MSC m (i.e. $enc(m, \overline{k})$), while B contains the other formulas of ξ (i.e. $\phi[v_i^{idx_i[j]}/v_i[j]]$). ψ is an interpolant of A and B. Note that, if $\psi \models \bot$, we deduce that ϕ is not responsible of the unfeasibility and that the unrolling of the network is inconsistent by itself.

Finally, we compute the third explanation type ψ_i for the i-th component of $\mathcal{N}$. We partition the unsatisfiable core ξ in the formulas A and B. A is the conjunction of all the formulas of ξ which encode the unrolling of the i-th component along its instance σ_i and CMSC constraints (i.e. $enc(\sigma_i, \overline{k}_i)$ and ϕ_i), while B is the conjunctions of all the other formulas of ξ. If $\psi_i \models \top$, the component does not play a role in the unfeasibility. On the contrary, if $\psi_i \models \bot$, the component does not have a path compatible with its instance.

Note that, when the abstraction is used to prove the unfeasibility of the scenario, the three types of explanations are still valid.

Distributed Controller [21]. This 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 3 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$_1$ accesses the processor (event *read*$_1$); 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 3 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.

The scenario is proved unfeasible. In Figure 3 we outline in gray the elements of the scenario, events and constraints, which contribute to the unfeasibility. In particular, the unsatisfiable core contains all the events of the CMSC apart from Ack_1 and Ack_2. Thus, the unfeasible CMSC prefix does not include the last event Ack_1.

We exploit the interpolation techniques to get the constraints $z >= x_1$.All the interpolants are computed by the SMT solver, then we manually simplify them removing redundant constraints. 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$_1$.

Audio Control Protocol [21]. The benchmark models a protocol that 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.

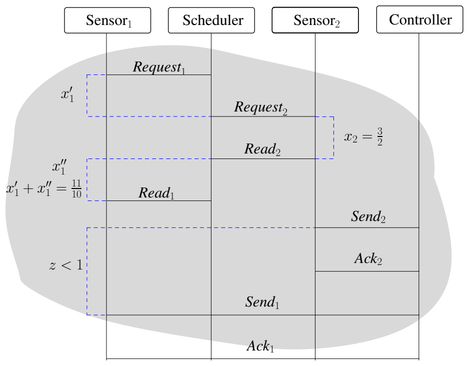


Fig. 3 The MSC for the distributed controller.

The considered scenarios consist of unfeasible timed sequences of up. For example, the sequence $\langle up, 4\rangle$, $\langle up, 8\rangle$, $\langle up, 12\rangle$, $\langle up, 16\rangle$, $\langle up, 19\rangle$, $\langle up, 23\rangle$ does not respect the protocol, since the 4-th and 5-th events must be separated by 3 seconds.

Once scenario-based induction proves that the scenario is unfeasible, the unsatisfiable core contains the encoding of the 4-th and 5-th event, thus the unfeasible CMSC prefix is the CMSC formed by the first 5 events. 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.

Electronic Height Control System [30]. 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 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 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 evolves 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 $\dot{f} \in [a, b]$.

In order to prove the scenario unfeasible, abstraction is required. In fact, in the concrete state space of the environment, the portrait partitioning creates discrete loops that correspond to infinite simple paths, and thus concrete K-induction can not converge. K-induction combined with abstraction 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).

For the environment we used 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 localise the abstraction by using $t \in [i, i+1]$ only in the i-th event and considering the partition consistent with the initial, values. We added a total of 16 predicates to the first two local segments of the environment. This way, the unfeasibility can be proved unrolling the environment for 25 steps in the first segment and 20 steps in the second segment. The details of the abstraction are available at `http://es.fbk.eu/people/mover/tests/FMSD11/`.

We extract an unfeasible CMSC formed by the first 2 events. Non-trivial explanations of the third type are associated with the timer, where the 2-nd event must happen in less then 3 seconds, and with the environment, where the same event to happen not before 3.3 seconds.

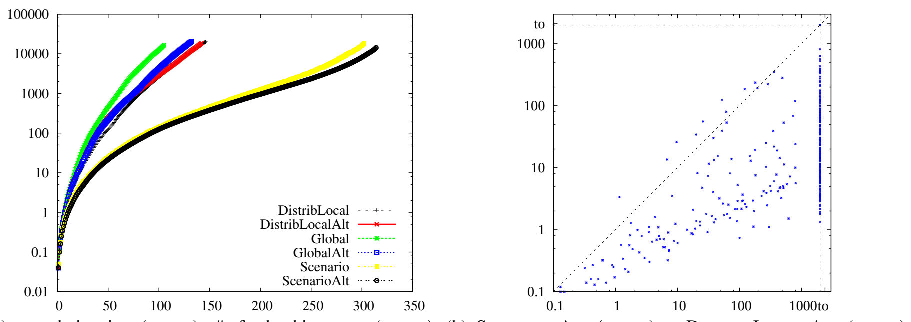
8 Experimental Evaluation

8.1 Settings

The techniques discussed in the previous sections were implemented in an extended version of NuSMV [10], that deals with networks of HAs, formalized in the HYDI language [12]. The extended version of NuSMV 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 BMC, inductive reasoning, and explanation extraction. For the comparison, we implemented the different approaches based on the automata construction [11].

In the experimental evaluation, we used the following benchmarks: *Distributed Controller*, *Electronic Height Control System (EHC)* and *Audio Protocol*, presented in Section 7; *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 section; *Ring-shape Fischer* is a variant of the protocol where the processes are in a ring, and each process shares a lock variable with its neighbors; *Nuclear Reactor* [39] models the control of a nuclear reactor with n rods.

First we compare the scenario-based encoding with the automata-based approach on feasible MSCs. Then we evaluate the scenario-driven induction with the k-induction performed on the system composed with the monitor. The experimental comparison does not take into account the computation of unfeasibility explanations. On the one hand, the extraction of explanation does not appear to be straightforward for the automata-based approach. On the other hand, the overhead largely depends on the fact that the SMT solver must be run with proof logging activated. This can in general lead to non-negligible overheads, but in the benchmarks we analyzed this did not turn out to be the case.



(a) cumulative time (y axes) - # of solved instances (x axes) (b) SCENARIOALT (y axes) vs. DISTRIBLOCALALT (x axes)

Fig. 4 (a) Cactus plot of run times (sec.). (b) Scatter plots of run times (sec.).

The experiments were run on two Linux machine (with an Intel i7 CPU at 2.93 for feasible MSCs and an Intel Core 2 Quad CPU 2.66 for unfeasible MSCs), setting the timeout and the memory out for a single run to 900 seconds and to 4 GB. The test cases, the executable and the results are available at `http://es.fbk.eu/people/mover/tests/FMSD11/`.

8.2 Scenario-driven BMC

We compare the scenario-based approach with the automata-based approach on a set of feasible meaningful scenarios, that describe the interaction of all the automata in the benchmarks, possibly containing parallel event synchronizations. We evaluate the scalability of the proposed approaches with respect to the number of components in the network and to the length of the MSCs. We increase the number of the components for all the benchmarks, except for the *EHC* and the *Audio Protocol* which have a fixed number of processes.

For the automata-based approach we exploit two different construction of the monitor. In the approach called GLOBAL we construct a single monitor which represents one of the possible equivalent partial-orders imposed by the scenario, while in DISTRIBLOCAL we build a distributed monitor, one for each hybrid automaton in the network. In both construction, we used the optimization of step semantics. Then we check the reachability of the target state of the monitor in the model obtained composing the monitor with the original system. The search is performed using an incremental BMC.

We evaluate the scenario approach, called (SCENARIO), where we further optimize the encoding locally simplifying it with respect to the structure of the different components. Additionally, we evaluate for both the automata-based and the scenario-based approach the optimization where we alternate the timed and the discrete transitions in the encoding. We add the suffix ALT to the names of the approaches to denote this variant.

The main findings of the experimental evaluation regard the effectiveness of the scenario-based encoding, which outperforms the optimized automata-based techniques. The Figure 4(a) shows a cactus plot (in logarithmic scale) for all the tested instances of benchmarks and scenarios. The plot shows the cumulative time (on the y axes, in seconds) to solve a given number of instances (on the x axes). From the plot it is clear that the scenario-driven encoding solves more instances than the automata approaches, and is significantly faster. Moreover, we note that the alternation improves the performances for the scenario and for the global automaton, while it is counterproductive for DISTRIBLOCAL. Figure 4(b) shows a scatter plot that compares the run-times of DISTRIBLOCALALT and SCENARIOALT on every instance. A point in the plot represents the time used by DISTRIBLOCALALT (x axes) and by SCENARIOALT (y axes) to solve an instance. SCENARIOALT outperforms DISTRIBLOCALALT in almost all the benchmarks.

8.3 Scenario-driven Induction

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, using the DISTRIBLOCAL construction with step semantics.

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. The scatter plot in Figure 5 shows the execution time for both methods on all the instances. The Scenario-based induction is clearly superior to monolithic k-induction. This because it exploits the structure of the MSC, resulting in localized simple path conditions, that are both simpler, and more effective, so that unsatisfiability is detected with a much shorter unrolling.

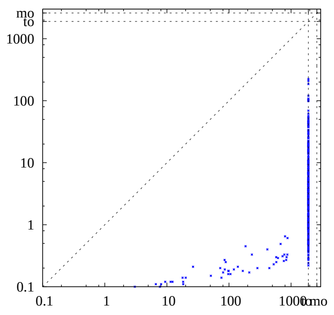


Fig. 5 Run times (sec.): monolithic induction (x axes) vs. scenario-induction (y axes).

9 Related Work

MSCs [22] are a basic building block to describe the interactions among components. Several works, such as High-Level Message Sequence Charts [28] and Live Sequence Charts (LSC) [14], 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 [2,5,9]. We consider a trace-based semantics for the MSC, where the MSC predicates range over the observable events of a system [25, 26]. While several works use MSCs to describe the entire system [3, 31], 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. In [9] the authors consider the feasibility problem for MSCs with timed constraints and a timed message-passing automaton. Their solution consists of a translation of the timed MSC into an automaton, reducing the problem to a reachability analysis for timed systems. They support also “weak” embeddings, where the MSC specification can be partial. *Live Sequence Charts (LSCs)* are translated into timed automata in the UPPAAL model checker [27], while in [24] 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 give 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, 15, 16, 38] can be used to solve the feasibility problem. The use of the step semantics to optimize the BMC encodings for asynchronous systems was investigated in [19,23]. 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 [35]. K-induction is complete for finite state systems, but it was applied also to infinite state systems in [29,32,36]. In [29] 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 in [36]. 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.

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 [34]. Interpolation for temporal properties is proposed in [33] as a theoretical framework for analyzing vacuity for discrete systems; the practical implications are not addressed in depth. In [34], 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.

10 Conclusions and Future Work

In this paper, we described a new SMT-based verification technique tailored to the verification of MSC feasibility in a network of hybrid automata. The encoding of the problem exploits the local time semantics and structures the search according to the local segments of the MSC. The events of the MSC are used to simplify the encoding while the local transitions are incrementally added up to a certain bound. A specialized version of k-induction, based on the localization of simple paths, may prove that the MSC is instead unfeasible. Unsat cores and interpolants are used to explain the reason of such unfeasibility. The experiments show that the proposed method significantly outperforms optimized techniques based on the reduction to reachability, and is able to construct interesting explanations.

In the future, we will address the following research directions. First, will lift the proposed techniques to incompletely specified MSCs. Second, we will experiment with abstraction refinement techniques such as localization reduction based on the structure of the network of components. Finally, we will address hybrid automata with nonlinear dynamics.

References

1. Ábrahám, E., Becker, B., Klaedtke, F., Steffen, M.: Optimizing bounded model checking for linear Hybrid Systems. In: VMCAI, pp. 396–412 (2005)
2. Akshay, S., Bollig, B., Gastin, P.: Automata and logics for timed message sequence charts. In: FSTTCS, pp. 290–302 (2007)
3. Alur, R., Yannakakis, M.: Model Checking of Message Sequence Charts. In: CONCUR, pp. 114–129 (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., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: CONCUR, pp. 485–500 (1998)
7. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT4 smt solver. In: CAV, pp. 299–303. Springer (2008)
8. Bu, L., Cimatti, A., Li, X., Mover, S., Tonetta, S.: Model checking of hybrid systems using shallow synchronization. In: FORTE, pp. 155–169 (2010)
9. Chandrasekaran, P., Mukund, M.: Matching scenarios with timing constraints. In: FORMATS, pp. 98–112 (2006)
10. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: an opensource tool for symbolic model checking. In: CAV, pp. 359–364 (2002)
11. Cimatti, A., Mover, S., Tonetta, S.: Efficient scenario verificastion for Hybrid Automata. In: CAV, pp. 317–332 (2011)
12. Cimatti, A., Mover, S., Tonetta, S.: HYDI: a language for symbolic hybrid systems with discrete interaction. In: EUROMICRO-SEAA, pp. 275–278 (2011)
13. Cimatti, A., Mover, S., Tonetta, S.: Proving and explaining the unfeasibility of Message Sequence Charts for hybrid systems. In: FMCAD (2011)
14. Damm, W., Harel, D.: LSCs: Breathing life into message sequence charts. Formal Methods in System Design **19**(1), 45–80 (2001)
15. Fränzle, M., Herde, C.: Efficient proof engines for bounded model checking of hybrid systems. ENTCS **133**, 119–137 (2005)
16. 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)
17. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Combination methods for satisfiability and model-checking of infinite-state systems. In: CADE, pp. 362–378 (2007)
18. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: CAV, pp. 72–83 (1997)
19. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. Theory and Practice of Logic Programming **3**(4-5), 519–550 (2003)
20. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE CS (1996)
21. Henzinger, T.A., Ho, P.: HyTech: the Cornell HYbrid TECHnology tool. In: Hybrid Systems II, LNCS 999, pp. 265–293 (1995)
22. ITU-T: Recommendation Z.120 - Message Sequence Charts (1996)
23. J., D., Junttila, T., Heljanko, K.: Exploiting step semantics for efficient bounded model checking of asynchronous systems. Sci. Comput. Program. (2011)
24. Klose, J., Wittke, H.: An automata based interpretation of Live Sequence Charts. In: TACAS, pp. 512–527 (2001)
25. Ladkin, P., Leue, S.: On the semantics of message sequence charts. In: FBT, pp. 88–104 (1992)
26. Ladkin, P.B., Leue, S.: Interpreting Message Flow Graphs. Formal Asp. Comput. **7**(5), 473–509 (1995)
27. 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 pp. 200–264 (2010)
28. Mauw, S., Reniers, M.A.: High-level message sequence charts. In: SDL Forum, pp. 291–306 (1997)
29. de Moura, L., Rueß, H., Sorea, M.: Bounded Model Checking and induction: from refutation to verification. In: CAV, pp. 14–26 (2003)
30. Müller, O., Stauner, T.: Modelling and verification using Linear Hybrid Automata - a case study. Mathematical and Computer Modelling of Dynamical Systems **71**, 71–89 (2000)
31. Pan, M., Bu, L., Li, X.: TASS: timing analyzer of scenario-based specifications. In: CAV, pp. 689–695 (2009)
32. Pike, L.: Real-Time System Verification by k-Induction. Tech. Rep. NASA/TM-2005-213751, NASA (2005)
33. Samer, M., Veith, H.: On the notion of vacuous truth. In: LPAR, pp. 2–14 (2007)
34. Schuppan, V.: Towards a notion of unsatisfiable cores for LTL. In: FSEN, pp. 129–145 (2009)
35. Sheeran, M., Singh, S., Stålmarck, G.: checking safety properties using induction and a SAT-solver
36. Tonetta, S.: Abstract model checking without computing the abstraction. In: FM, pp. 89–105 (2009)
37. Vardi, M.: An automata-theoretic approach to Linear Temporal Logic. In: Banff Higher Order Workshop, pp. 238–266 (1995)
38. Walter, D., Little, S., Myers, C.J., Seegmiller, N., Yoneda, T.: Verification of Analog/Mixed-Signal circuits using symbolic methods. IEEE Trans. on CAD of Integrated Circuits and Systems **27**(12), 2223–2235 (2008)
39. Wang, F.: Symbolic parametric safety analysis of linear hybrid systems with BDD-like data structures. IEEE TSE **31**(1), 38–51 (2005)

A Proofs

A.1 Theorem 1

A.1.1 Shallow synchronization

In order to prove Theorem 1, we briefly recap the concept of shallowly synchronized paths from [8].

Definition 1 **(S-trace)** Given a set of events $S \subseteq A$ and a path $\pi = s_0; a_1; s_1; \ldots; a_h; s_h$, the *$S$-trace* $\tau_S(\pi)$ is the sequence of events $\langle t_1, a_1\rangle; \ldots; \langle t_k, a_k\rangle$ where t_j is the time at which the event a_j occurs in π and $a_j \in S$, for $1 \leq j \leq k$.

Definition 2 **(Consistent traces)** Let π_1 and π_2 be two paths over the sets of events A_1 and A_2 respectively, and $S = A_1 \cap A_2$. The pair $\langle \pi_1, \pi_2\rangle$ is *consistent* iff the S-trace of π_1 is equal to the S-trace of π_2 ($\tau_S(\pi_1) = \tau_S(\pi_2)$) and the final time of π_1 is equal to the final time of π_2.

Definition 3 **(Shallowly synchronized path)** A *shallowly synchronized path* of a network $S_{\textsc{LocTime}}(\mathcal{N})$ is a tuple $\pi_{\textsc{shallow}} = \langle \pi_1, \ldots, \pi_n\rangle$ such that π_j is a path of S_i and, for all i, j, $1 \leq i < j \leq n$, π_i and π_j are consistent.

Definition 4 **(Projection)** Given S_i and a path $\pi_{\textsc{LocTime}}$ in $S_{\textsc{LocTime}}(\mathcal{N})$, the projection of $\pi_{\textsc{LocTime}}$ over S_i is the path $prj(\pi_{\textsc{LocTime}}, i)$ obtained projecting the states over the S_i-th component and removing all the transitions over events which are not in $A_i \cup \{\textsc{t}\}$ (in the alphabet of H_i and the timed event). Note that the stutter event ($\textsc{s}$) is projected. Formally, given the component S_i with alphabet A_i and $\pi_{\textsc{LocTime}} := s_0; a_1; s_1; \ldots; a_h; s_h$, the projection of $\pi_{\textsc{LocTime}}$ over S_i is the path $prj(\pi_{\textsc{LocTime}}, i) := s'_0; a'_1; s'_1; \ldots; a'_l; s'_l$ such that:

- $f_{A_i} : \mathcal{N} \times \mathcal{N}$ is a function such that $f_{A_i}(z)$ maps the index in $\pi_{\textsc{LocTime}}$ of the z-th occurrence of an event which belongs to the set A_i.
- l is the number of events in the path $\pi_{\textsc{LocTime}}$ which also belong to A_i.
- $s_{|V}$ is the $\mathcal{T}$-model s restricted to the symbols of the signature V.
- $s'_0 := s_0$.
- for all $1 \leq j \leq l$, $s'_j := s_{f_{A_i}(j)}{}_{|V_i}$.
- for all $1 \leq j \leq l$, $a'_j := a_{f_{A_i}(j)}{}_{|A_i}$.

Theorem 3 *If $\pi_{\textsc{LocTime}} := s_0; a_1; s_1; \ldots; a_h; s_h$ is a path in the local-time semantics then $\pi_{\textsc{shallow}} = \langle prj(\pi_{\textsc{LocTime}}, 1), \ldots, prj(\pi_{\textsc{LocTime}}, n)\rangle$ is a* shallowly synchronized *path.*
Vice versa, given a shallowly synchronized *path $\pi_{\textsc{shallow}}$ there exists a path $\pi_{\textsc{LocTime}}$ in $S_{\textsc{LocTime}}(\mathcal{N})$ such that $\pi_{\textsc{shallow}} = \langle prj(\pi_{\textsc{LocTime}}, 1), \ldots, prj(\pi_{\textsc{LocTime}}, n)\rangle$.*

Proof $\pi_{\textsc{shallow}}$ is a *shallowly synchronized path* since:

- for all $1 \leq i \leq n$, $prj(\pi_{\textsc{LocTime}}, i) \models S_i$: By construction, the projection of $prj(\pi_{\textsc{LocTime}}, i)$ removes from the network path $\pi_{\textsc{LocTime}}$ all the transitions which are not over $A_i \cup \textsc{t}$. $prj(\pi_{\textsc{LocTime}}, i) \models S_i$, since in the transitions removed from $\pi_{\textsc{LocTime}}$ S_i stutters, thus it does not change the value of its local states.
- for all $1 \leq i < j \leq n$, $prj(\pi_{\textsc{LocTime}}, i)$ and $prj(\pi_{\textsc{LocTime}}, j)$ are consistent: by construction, the projection does not change the order of states and events of $\pi_{\textsc{LocTime}}$, and restricts each projection to a given alphabet. $prj(\pi_{\textsc{LocTime}}, i)$ is restricted to all the events in A_i while $prj(\pi_{\textsc{LocTime}}, j)$ is restricted to all the events in A_j. $\tau_{A_i \cap A_j}(prj(\pi_{\textsc{LocTime}}, i))$ and $\tau_{A_i \cap A_j}(prj(\pi_{\textsc{LocTime}}, j))$ restrict both sequences to events in $A_i \cap A_j$. The two sequences contain the same events and have the same order and, therefore, they are equal. Moreover, since $\pi_{\textsc{LocTime}}$ is in $S_{\textsc{LocTime}}(\mathcal{N})$, the assignment to the variables t_i, t_j in the last state of $\pi_{\textsc{LocTime}}$ is the same. Thus, also the assignments to t_i and t_j in the last state of $prj(\pi_{\textsc{LocTime}}, i)$ and $prj(\pi_{\textsc{LocTime}}, j)$ respectively must be the same.

If $\pi_{\textsc{shallow}}$ is a *shallowly synchronized* path, then there exists a path $\pi_{\textsc{LocTime}}$ in $S_{\textsc{LocTime}}(\mathcal{N})$ such that $\pi_{\textsc{shallow}} = \langle prj(\pi_{\textsc{LocTime}}, 1), \ldots, prj(\pi_{\textsc{LocTime}}, n)\rangle$. $\pi_i := s^i_0; a^i_1; s^i_1; \ldots; a^i_{l^i}; s^i_{l^i}$. We recursively define the function γ which maps $\pi_{\textsc{shallow}}$ to a path $\pi_{\textsc{LocTime}} \in S_{\textsc{LocTime}}(\mathcal{N})$ ($\pi_{\textsc{LocTime}} := \gamma(\pi_{\textsc{shallow}})$):

- if $\pi_i = s^i_0$ for all $1 \leq i \leq n$, then $\gamma(\pi_{\textsc{shallow}}) := s^1_0 \cup \ldots \cup s^n_0$ (i.e. all the local paths have a single state).

– if there exists an i in $1 \leq i \leq n$ such that $\pi_i := s^i_0; a^i_0; \ldots; s^i_{li-1}; a^i_{li}; s^i_{li}$ and a^i_0 is a local event of S_i, then $\gamma(\pi_{\text{SHALLOW}}) := s^1_0 \cup \ldots \cup s^i_0 \cup \ldots \cup s^n_0; a^1 \cup \ldots a^n; \gamma(\langle \pi_1, \ldots, \pi'_i, \ldots, \pi_n \rangle)$, where

$$a^j = \begin{cases} a^j_0 & \text{if } i = i \\ a^j \text{ s.t. } a^j \models \varepsilon_j = \text{S} & \text{otherwise} \end{cases}$$

and $\pi'_i := s^i_1; a^i_1; \ldots; s^i_{li-1}; a^i_{li}; s^i_{li}$ (i.e. when a process can move on a local event, all the other processes stutter).

– otherwise, since π_{SHALLOW} is a *shallowly synchronized path*, there exists a set J of indexes and and event $a \in \bigcap_{i \in J} A_i$ such that $\bigcup_{i \in J} a_i \models \bigwedge_{i \in J} \varepsilon_i = a$ and $\bigcup_{i \in J} s_i \models \bigwedge_{i,j \in J, i \neq j} t_i = t_j$ (i.e. all the processes with index in J synchronize on the event a). In this case $\gamma(\pi_{\text{SHALLOW}}) := s^1_0 \cup \ldots \cup s^i_0 \cup \ldots \cup s^n_0; a^1 \cup \ldots a^n; \gamma(\langle \pi'_1, \ldots, \pi'_n \rangle)$, where:

$$a^i = \begin{cases} a^i_0 & \text{if } i \in J \\ a^i \text{ s.t. } a^i \models \varepsilon_i = \text{S} & \text{otherwise} \end{cases}$$

and

$$\pi'_i = \begin{cases} s^i_1; a^i_1; \ldots; s^i_{li-1}; a^i_{li}; s^i_{li} & \text{if } i \in J \\ \pi_i & \text{otherwise} \end{cases}$$

We can prove by induction that $\pi_{\text{LocTime}} \in S_{\text{LocTime}}(\mathcal{N})$. Moreover, since the last time of all the components in $\gamma(\pi_{\text{SHALLOW}})$ are equal, then $\langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, m) \rangle$ is a *shallowly synchronized path.*

A.1.2 Proof of Theorem 1

Proof Let us consider a model μ of the formula $enc(\langle m, \phi \rangle, \overline{k})$. μ is a model over the variables of the network $S_{\text{LocTime}}(\mathcal{N})$. For all $1 \leq i \leq n$, consider the projection π_i of μ over the symbols defined in the Σ-structure V_i and $\mathcal{W}_i$ of S_i. By construction π_i is a path of S_i (i.e. $\pi_i \models S_i$), $\pi_i \models^m \sigma_i$ and $\bigcup_{1 \leq i \leq n, 0 \leq j \leq |\sigma_i|} pre_j(\pi_i) \models \phi$. Moreover, for all $1 \leq i < j \leq j$, π_i and π_j are consistent, since the MSC is consistent and the events happen at the same time due to *sync*. Thus, $\langle \pi_1, \ldots, \pi_n \rangle$ is a shallowly synchronized path. By Theorem 3, there exists a path π_{LocTime} in $S_{\text{LocTime}}(\mathcal{N})$ such that $\pi_{\text{SHALLOW}} := \langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, n) \rangle$. By the definition in Section 4, it means that $\mathcal{N} \models \langle m, \phi \rangle$.

If $\mathcal{N} \models \langle m, \phi \rangle$, then there is a path π_{LocTime} such that $\pi_{\text{LocTime}} \models S_{\text{LocTime}}(\mathcal{N})$. This means that $prj(\pi_{\text{LocTime}}, i) \models^m \sigma^i$, $prj(\pi_{\text{LocTime}}, i) \models S_i$ and $\bigcup_{1 \leq i \leq n, 0 \leq j \leq |\sigma_i|} pre_j(prj(\pi_{\text{LocTime}}, i)) \models \phi$. By Theorem 3, $\langle prj(\pi_{\text{LocTime}}, 1); \ldots; prj(\pi_{\text{LocTime}}, n) \rangle$ forms a shallowly synchronized path. Let us consider a $\overline{k}$ such that, for all $1 \leq i \leq n$, for all $0 \leq j \leq |\sigma_i|$, $k_i[j]$ is equal to the length of the local segment *lsg*$(\sigma_i[j])$ in $prj(\pi_{\text{LocTime}}, i)$. $prj(\pi_{\text{LocTime}}, 1) \cup \ldots \cup prj(\pi_{\text{LocTime}}, n) \models enc(\langle m, \phi \rangle, \overline{k})$, since $prj(\pi_{\text{LocTime}}, i) \models enc(\sigma_i, \overline{k}_i)$ and $\bigcup_{1 \leq i \leq n, 0 \leq j \leq |\sigma_i|} pre_j(prj(\pi_{\text{LocTime}}, i)) \models \phi$ by definition, and since $\langle prj(\pi_{\text{LocTime}}, 1); \ldots; prj(\pi_{\text{LocTime}}, n) \rangle$ is a shallowly synchronized path then the *sync* constraints holds in $enc(\langle m, \phi \rangle, \overline{k})$ (i.e. the synchronization happen at the same time and the time at the end of all the paths is the same).

Remark 2 Here we do not prove that the *global time semantics*, which is more common in the definition of Hybrid Automata [20], and the *local time semantics* [6] are equivalent for our purposes (i.e. for the scenario verification problem).

However, a proof of Theorem 1 which considers the global time semantics is available at `https://es.fbk.eu/people/mover/paper/scenario/main.pdf`.

A.2 Theorem 2

A.2.1 Additional lemmas

In the following we provide two additional lemmas which are used to prove Theorem 2.

We introduce the following notation shorthands. Given a tuple of bounds $\overline{k}$ we write $\overline{k}' \geq \overline{k}$ iff for all $1 \leq i \leq n$, for all $0 \leq j \leq |\sigma_i|$, $k_i'[j] \geq k_i[j]$. Given a path π such that $\pi \models^m \sigma_i$, $|loc_j(\pi)|$ is the length of the sequence of states $loc_j(\pi)$ of π.

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

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

Note that $<_m$ is defined in a similarly to $\leq_m$.

Lemma 1 *Given a CMSC $\langle m, \phi \rangle$ and a tuple of bounds $\overline{k}$, if $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable then for all $\overline{k}' \geq \overline{k}$, $enc(\langle m, \phi \rangle, \overline{k}')$ is satisfiable.*

Proof If $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable then there exists a path π consistent with $\langle m, \phi \rangle$ such that for all $1 \leq i \leq n$, for all $1 \leq j|\sigma_i|$, $|loc_j(prj(i, \pi))|$ is $k_i[j]$.

π can be extended to a path π' as follows. For all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$:

- $pre_j(prj(\pi, i)) = pre_j(prj(\pi', i))$,
- $post_j(prj(\pi, i)) = post_j(prj(\pi', i))$,
- $loc_j(prj(\pi', i)) := loc_j(prj(\pi, i)); \varepsilon = \mathrm{S}; s_1; \ldots; \varepsilon = \mathrm{S}; s_{k_i'[j]-k_i[j]}$, where for $1 \leq z \leq k_i'[j] - k_i[j]$, s_z is equal to the last state of $loc_j(prj(\pi, i))$. $loc_j(prj(\pi', i))$ is the concatenation of the j-th local sequence of π ($loc_j(prj(\pi, i))$) with a sequence of stutter actions (i.e. $\varepsilon = \mathrm{S}$).

Thus, π' extends all the local sequences of π by stuttering actions S.

Since stutter does not change the state reached by the system, $\pi' \models enc(\langle m, \phi \rangle, \overline{k}')$.

Lemma 1 states that if $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable, then all the encodings which consider a $\overline{k}' >= \overline{k}$ are satisfiable as well, due to the insertion of stuttering action.

Lemma 2 *For all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$, if $kind_i[j]$ is unsatisfiable for bounds $\overline{k}$, then for all paths π such that $\pi \models enc(\langle \overline{m}_i[j], \overline{\phi}_i[j] \rangle, \overline{k})$ and $|loc_j(prj(i, \pi))| > k_i[j]$, $loc_j(prj(i, \pi))$ is not simple.*

Proof For all $1 \leq i \leq n$ and $j = 0$ if $kind_i[j]$ is unsatisfiable, $\pi \models enc(\langle \overline{m}_i[0], \overline{\phi}_i[0] \rangle, \overline{k})$ and $|loc_0(prj(i, \pi))| > k_i[j]$. In this case $enc(\langle \overline{m}_i[0], \overline{\phi}_i[0] \rangle, \overline{k})$ encodes the local segment $lsg(\sigma_i[0])$ and $kind_i[0]$ is unsatisfiable. Thus, it does not exist a simple path $loc_0(prj(i, \pi))$ longer than $lsg(\sigma_i[0])$.

Consider the local segment $lsg(\sigma_i[j])$ and suppose that the lemma holds for all l, h such that $lsg(\sigma_l[h]) <_m lsg(\sigma_i[j])$ (i.e. the theorem holds for all the local segment which are found "before" in the partial order defined by $<_m$). Suppose that the bounds $\overline{k}'$ are the same as $\overline{k}'$, except for $k_i'[j]$ which is $k_i[j] + 1$. For every path π such that $\pi \models enc(\langle \overline{m}_i[j], \overline{\phi}_i[j] \rangle, \overline{k}')$, we have that $loc_j(prj(i, \pi))$ is not a simple path, since $kind_i[j]$ is unsatisfiable.

By induction, the lemma holds for all $loc_j(prj(i, \pi))$.

A.2.2 Proof of Theorem 2

Proof Suppose that $\mathcal{N} \models m$.

By Theorem 1 there exist bounds $\overline{k}'$ and a path π' such that:

- $\pi' \models enc(\langle m, \phi \rangle, \overline{k}')$;
- for all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$, $loc_j(prj(\pi', i))$ is of length $k_i'[j]$.

We consider the bounds $\overline{k}'$ such that $\overline{k}' \geq \overline{k}$. If we show that $enc(\langle m, \phi \rangle, \overline{k}')$ is unsatisfiable, then by Lemma 1 we known that for all the $k'' \leq k$ $enc(\langle m, \phi \rangle, \overline{k}'')$ is unsatisfiable.

For all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$, by Lemma 2 it does not exist a π'' consistent with $\langle m, \phi \rangle$ such that its local segment $loc_j(prj(\pi'', i))$ is simple and longer than $k_i[j]$.

Thus, there exists a path π such that:

- for all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$:
  - $pre_j(prj(\pi, i)) = pre_j(prj(\pi', i))$;
  - $post_j(prj(\pi, i)) = post_j(prj(\pi', i))$;
  - the length of loc_j is $k_i[j]$.
- $\pi \models enc(\langle m, \phi \rangle, \overline{k})$.

This contraddicts the fact that by hypothesis $enc(\langle m, \phi \rangle)$ is unsatisfiable.