Model Checking of Hybrid Systems using Shallow Synchronization*

Lei Bu[1], Alessandro Cimatti[2], Xuandong Li[1], Sergio Mover[2], and Stefano Tonetta[2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University
[2] Fondazione Bruno Kessler - IRST

Abstract. Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables. The traditional semantics of a network of automata is based on interleaving, and requires the construction of a monolithic hybrid automaton based on the composition of the automata. This destroys the structure of the network and results in a loss of efficiency, especially using bounded model checking techniques. An alternative compositional semantics, called "shallow synchronization", exploits the locality of transitions and relaxes time synchronization. The semantics is obtained by composing traces of the local automata, and superimposing compatibility constraints resulting from synchronization.
In this paper, we investigate the different symbolic encodings of the reachability problem of a network of hybrid automata. We propose a novel encoding based on the shallow synchronization semantics, which allows different strategies for searching local paths that can be synchronized. We implemented a bounded reachability search based on the use of an incremental Satisfiability-Modulo-Theory solver. The experimental results confirm that the new encoding often performs better than the one based on interleaving.

1 Introduction

Hybrid automata ([13]) are increasingly recognized as a clean modeling framework for systems with discrete and continuous variables. Many systems are structured into components, and can often be naturally modeled as networks of communicating hybrid automata: local activities of each component 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 traditional asynchronous semantics is based on interleaving, and requires the construction of a monolithic hybrid automaton based on the composition of the automata in the network. Intuitively, this means that a path in the automaton is the result of the composition of interleaving paths. However, the monolithic automaton resulting from the composition can be seen

* The authors at Nanjing University are supported by the National Natural Science Foundation of China (No.90818022 and No.60721002) and the National 863 High-Tech Programme of China (No.2009AA01Z148). A. Cimatti is supported by the European Commission (FP7-2007-IST-1-217069 COCONUT and ACP7-GA-2008-212088 MISSA) S. Tonetta is supported by the Provincia Autonoma di Trento (project ANACONDA).

as the result of a "strict synchronization", and the analysis has to deal with an overly large number of paths, since the structure and the locality of the network are not taken into account.

An alternative semantics [5] for networks of automata exploits the fact that automata can be "shallowly synchronized". The intuition is that each automaton can proceed based on its individual "local time scale", unless they perform a synchronizing transition, in which case they must realign their absolute time. 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.

In this paper, we provide a fully symbolic account for bounded reachability under "shallow synchronization", and we explore various search strategies. We implement the approach in the sub-case of linear hybrid automata and we use a Satisfiability-Modulo-Theory (SMT) [15] solver to check the satisfiability of the formulas encoding the reachability problem. The main advantage is that the transition relation of each automata is unrolled only for the steps necessary to reach locally the target (regardless the length of the interleaving with the other automata). Typically, local paths are much shorter because they do not need to stutter allowing other processes to perform local or non-shared events. The disadvantage is that we may use additional variables and constraints. We experimentally investigate this trade-off and the results show that the new encoding often performs better than the one based on interleaving. In particular, the improvement increases at the growth of the difference between the length of the local traces and the length of the interleaving trace.

The paper is structured as follows. In Section 2 we present some background on hybrid automata and their composition through interleaving. In Section 3 we present the shallow synchronization semantics revising the concepts described in [5] and defining explicit mappings from the semantics with strict synchronization to the shallowly synchronized one, and vice versa. In Section 4 we show several ways to symbolically encode the bounded model checking problem for shallow synchronization semantics. In Section 5 we discuss related work. In Section 6 we experimentally evaluate our approach. In Section 7 we draw some conclusions.

2 Background

Notation Given a set V of real-valued variables, we denote with $\mathcal{LB}(V)$ the set of Boolean combinations of linear equalities and inequalities over V. We denote with V' the set of "next" variables and with $\dot{V}$ the set of first derivative of the variables in V over time. We write $V' = V$ as an abbreviation for $\bigwedge_{v \in V} v' = v$.

If f is a collection of real functions $\{f^v\}_{v \in V}$, we denote with f^V the composed function $f^V(t) = \Pi_{v \in V} f^v(t)$.

Given a formula ϕ in $\mathcal{LB}(V)$ and $\overline{V}$ a set of copies of the variables in V, we denote with $\phi(\overline{V})$ the formula obtained by substituting each $v \in V$ with its copy $\overline{v} \in \overline{V}$. Given a formula ϕ in $\mathcal{LB}(\dot{V})$, two copies $\overline{V}_1$ and $\overline{V}_2$ of V, and ψ a linear term, we denote with $\phi(\frac{\overline{V}_2 - \overline{V}_1}{\psi})$ the formula obtained by substituting each $\dot{v} \in \dot{V}$ with $\frac{\overline{v}_2 - \overline{v}_1}{\psi}$ and then multiplying by ψ (thus $\phi(\frac{\overline{V}_2 - \overline{V}_1}{\psi})$ is a Boolean combination of linear constraints).

2.1 Hybrid automata

Due to lack of space, but without loss of generality, we restrict the presentation to the framework of Linear Hybrid Automata (LHA). The results presented in the rest of this paper however apply to the general case of Hybrid Automata as defined in [13][3].

Definition 1 ([13]). *A* LHA *is a tuple* $\langle Q, E, X, F, I, Z, J, U, L\rangle$ *where*

- Q *is the set of locations,*
- $E \subseteq Q \times Q$ *is the set of edges,*
- X *is the set of continuous variables,*
- *for each* $q \in Q$, $F(q) \in \mathcal{LB}(\dot{X})$ *is the flow condition (denoted also as* F_q*),*
- *for each* $q \in Q$, $I(q) \in \mathcal{LB}(X)$ *is the initial condition (denoted also as* I_q*),*
- *for each* $q \in Q$, $Z(q) \in \mathcal{LB}(X)$ *is the invariant condition (denoted also as* Z_q*),*
- *for each* $e \in E$, $J(e) \in \mathcal{LB}(X \cup X')$ *is the jump condition (denoted also as* J_e*),*
- U *is the set of labels,*
- *for each* $e \in E$, $L(e) \in U$ *is the label of the edge (denoted also as* L_e*).*

Definition 2. *A* run *of a LHA* H *is a sequence* $\langle q_0, s_0\rangle \xrightarrow{a_1} \langle q_1, s_1\rangle \ldots \langle q_{n-1}, s_{n-1}\rangle \xrightarrow{a_n} \langle q_n, s_n\rangle$ *such that:*

- *for all* i, $0 \leq i \leq n$, $q_i \in Q$ *and* s_i *is an assignment to the variables of* X*;*
- *for all* i, $1 \leq i \leq n$, $a_i \in U \cup \mathbb{R}^{\geq 0}$*; hereafter* $t_i = \sum_{1 \leq j \leq i, a_j \in \mathbb{R}^{\geq 0}} a_j$ *and* $t_0 = 0$*; we call* t_n *the final time of the run; we call the pair* $\langle a_i, t_i\rangle$ *an* event*;*
- *for all* i, $1 \leq i \leq n$, *if* $a_i \in \mathbb{R}^{\geq 0}$*, then* $q_{i-1} = q_i$ *and there exists a collection of real functions* $\{f_i^x\}_{x \in X}$ *such that* f_i^x *is differentiable over* $[t_{i-1}, t_i]$ *and* $f_i^X(t_{i-1}) = s_{i-1}$ *and* $f_i^X(t_i) = s_i$*;*
- *for all* i, $1 \leq i \leq n$, *if* $a_i \in U$ *then* $\langle q_{i-1}, q_i\rangle \in E$ *and* $a_i = L(\langle q_{i-1}, q_i\rangle)$*;*
- *for all* i, $1 \leq i \leq n$, *if* $a_i \in \mathbb{R}^{\geq 0}$*, then for all* $t \in [t_{i-1}, t_i]$*, then* $\dot{f}_i^X(t) \models F_{q_i}$*;*
- $s_0 \models I_{q_0}$ *and for all* i, $0 \leq i \leq n$, $s_i \models Z_{q_i}$*;*
- *for all* i, $1 \leq i \leq n$, *if* $a_i \in \mathbb{R}^{\geq 0}$*, then for all* $t \in [t_{i-1}, t_i]$, $f_i^X(t) \models Z_{q_i}$*;*
- *for all* i, $1 \leq i \leq n$, *if* $a_i \in U$ *then* $s_{i-1}, s_i \models J_{\langle q_{i-1}, q_i\rangle}$.

A run σ_1 is a refinement of another run σ_2 iff σ_1 is obtained by σ_2 by splitting some timed transition $\langle q_i, s_{i-1}\rangle \xrightarrow{a_i} \langle q_i, s_i\rangle$, $a_i \in \mathbb{R}^{\geq 0}$ into two or more timed transitions $\langle q_i, s_{i-1}\rangle \xrightarrow{a_{i_1}} \langle q_i, s_{i_1}\rangle \ldots \langle q_i, s_{i_{h-1}}\rangle \xrightarrow{a_{i_h}} \langle q_i, s_i\rangle$ such that $a_{i_j} \in \mathbb{R}^{\geq 0}$, $1 \leq j \leq h$, and, $\sum_{1 \leq j \leq h} a_{i_j} = a_i$. A timed transition $\langle q_i, s_{i-1}\rangle \xrightarrow{a_i} \langle q_i, s_i\rangle$ with $a_i = 0 \in \mathbb{R}^{\geq 0}$ is called a stuttering transition.

2.2 Network of hybrid automata

The definition of network of hybrid automata is based on the definition in [13], which means components communicate with each other by shared labels.

[3] As far as the solutions of the flow conditions can be represented in the logic handled by the SMT solver

Definition 3. *Given two LHAs* $H_1 = \langle Q_1, E_1, X_1, F_1, I_1, Z_1, J_1, U_1, L_1\rangle$ *and* $H_2 = \langle Q_2, E_2, X_2, F_2, I_2, Z_2, J_2, U_2, L_2\rangle$ *with* $Q_1 \cap Q_2 = X_1 \cap X_2 = \emptyset$*, the* composition $H_1 \times H_2$ *is the LHA* $\langle Q_P, E_P, X_P, F_P, I_P, Z_P, J_P, U_P, L_P\rangle$ *where*

- $Q_P = Q_1 \times Q_2$,
- $E_P = \{\langle q_1 \times q_2, q_1' \times q_2'\rangle \mid$ *either* $\langle q_1, q_1'\rangle \in E_1, q_2 = q_2', L_1(\langle q_1, q_1'\rangle) \notin U_2$, *or* $\langle q_2, q_2'\rangle \in E_2, q_1 = q_1', L_2(\langle q_2, q_2'\rangle) \notin U_1$, *or* $\langle q_1, q_1'\rangle \in E_1, \langle q_2, q_2'\rangle \in E_2, L_1(\langle q_1, q_1'\rangle) = L_2(\langle q_2, q_2'\rangle)\}$,
- $X_P = X_1 \cup X_2$,
- $F_P(q_1 \times q_2) = F_1(q_1) \wedge F_2(q_2)$,
- $I_P(q_1 \times q_2) = I_1(q_1) \wedge I_2(q_2)$,
- $Z_P(q_1 \times q_2) = Z_1(q_1) \wedge Z_2(q_2)$,
- $U_P = U_1 \cup U_2$,
- $J_P(\langle q_1 \times q_2, q_1' \times q_2'\rangle) = \begin{cases} J(\langle q_1, q_1'\rangle) \wedge X_2' = X_2 & \text{if } q_2 = q_2', L_1(\langle q_1, q_1'\rangle) \notin U_2 \\ J(\langle q_2, q_2'\rangle) \wedge X_1' = X_1 & \text{if } q_1 = q_1', L_2(\langle q_2, q_2'\rangle) \notin U_1 \\ J(\langle q_1, q_1'\rangle) \wedge J(\langle q_2, q_2'\rangle) & \text{if } L_1(\langle q_1, q_1'\rangle) = L_2(\langle q_2, q_2'\rangle), \end{cases}$
- $L_P(\langle q_1 \times q_2, q_1' \times q_2'\rangle) = \begin{cases} L(\langle q_1, q_1'\rangle) & \text{if } q_2 = q_2', L_1(\langle q_1, q_1'\rangle) \notin U_2 \\ L(\langle q_2, q_2'\rangle) & \text{if } q_1 = q_1', L_2(\langle q_2, q_2'\rangle) \notin U_1 \\ L(\langle q_1, q_1'\rangle) & \text{if } L_1(\langle q_1, q_1'\rangle) = L_2(\langle q_2, q_2'\rangle). \end{cases}$

Definition 4. *A* network $\mathcal{H}$ *of LHAs is a tuple of LHAs.*

The semantics of a network of automata is given by the composition of the automata.

Definition 5. *A* synchronized run *of a network* $\mathcal{H} = \langle H_1, \ldots, H_n\rangle$ *is a run of the composition* $H_1 \times \ldots \times H_n$.

In the following we refer to a run of a single automaton in a network as "local", to distinguish it from a run of the composition automaton.

Reachability problem Given a network of automata $\mathcal{H} = \langle H_1, H_2, \ldots, H_n\rangle$, and a target set $T = \langle q_1, q_2, \ldots, q_n\rangle$, the reachability problem for $\mathcal{H}$ and T is to verify whether $q_1 \times q_2 \times \cdots \times q_n$ can be reached in the composition $\mathcal{H}$. Thus, we consider only finite runs, although the approach can be extended to infinite runs which can be represented by lasso-shape paths.

3 Shallow Synchronization Semantics

While in strict synchronization the behavior of a network is basically obtained by interleaving, in shallow synchronization a run of the network is the result of "composition" of runs local to each automaton in the network. The intuition is demonstrated in Figure 1. In the upper part, we see three traces of three automata in a network. Each automaton H_i has a local label τ; the ij labels are shared between processes H_i and H_j; δ denotes local time elapse. We notice that the synchronization over the ij labels happens exactly at the same time, e.g., 12 takes place at absolute time 5, although the number of transitions required by H_1 and H_2 is different. In the lower part of the figure,

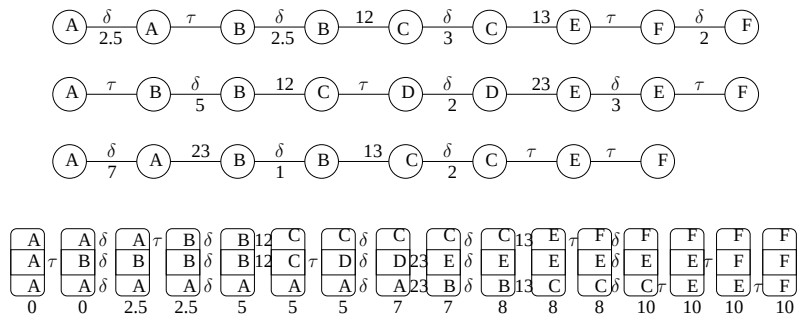


Fig. 1. Three local traces (above), and the corresponding interleaving (below).

we report the corresponding trace based on interleaving (where each box contains the state of each of the three processes). Stuttering (e.g. of process 1 and 3 in the first step) is modeled by the fact that a process does not have any label on its side.

We also define a mapping of a set of shallowly synchronized runs of the automata into a run in the composition of the automata. Intuitively, the mapping induces an equivalence relation among the runs of the composition automaton which are obtained by composing the same set of local runs with different interleaving. The shallow synchronization is defined according to the trace of a run i.e., the list of events occurring in the run. An S-trace, with $S \subseteq U$, is a trace restricted to the labels in S.

Definition 6. *Given a set of labels $S \subseteq U$ and a run $\sigma = \langle q_0, s_0\rangle \xrightarrow{a_1} \langle q_1, s_1\rangle \ldots \xrightarrow{a_n} \langle q_n, s_n\rangle$, the* S-trace *$\tau_S(\sigma)$ is the sequence of events $\langle a_1, t_1\rangle, \langle a_2, t_2\rangle, \ldots, \langle a_k, t_k\rangle$ where t_i is the time at which the event a_i occurs in σ.*

Definition 7. *Given two LHAs H_1 and H_2 with sets of labels U_1 and U_2 resp., let σ_1 be a run of H_1 and σ_2 a run of H_2. Let S be the intersection of U_1 and U_2 ($S = U_1 \cap U_2$). The pair $\langle \sigma_1, \sigma_2\rangle$ is* consistent *iff the S-trace of σ_1 is equal to the S-trace of σ_2 ($\tau_S(\sigma_1) = \tau_S(\sigma_2)$) and the final time of σ_1 is equal to the final time of σ_2.*

The last constraint on the final time is necessary because otherwise the two runs may terminate with a series of local steps with different timings.

Definition 8. *A* shallowly synchronized run *of a network of LHAs is a tuple $\theta = \langle \sigma_1, \ldots, \sigma_n\rangle$ such that σ_j is the run of H_j and, for all j, h, $1 \leq j < h \leq n$, σ_j and σ_h are consistent.*

If θ is a shallowly synchronized run, we denote with θ_j the j-th component of θ.

Remark 1. In general, two different events can occur at the same time in the same run, because discrete transitions are not forced to be interleaved with timed transitions. Moreover, simultaneous events may be interleaved with different orders.

However, in many cases, we can assume that whenever two events occur simultaneously, they have a fixed order. Then, the pair $\langle \sigma_1, \sigma_2\rangle$ is consistent simply iff for all

$a \in U_1 \cap U_2$ and $t \in \mathbb{R}$, $\langle a, t\rangle$ occurs in σ_1 iff $\langle a, t\rangle$ occurs in σ_2. I.e., having the events at the same time guarantees that the traces are the same. The definitions and theorems in [5] have this assumption, while in this section we consider the most general case.

Projection of a synchronized run of the composition automaton on one component is the corresponding run local to that component automaton. Intuitively, the set of projections of a synchronized run form a shallowly synchronized run. The projection induces an equivalence relation over strictly synchronized traces, namely the equivalence of runs that are the same modulo a reordering of the interleaved labels.

Definition 9. *Given a network $\mathcal{H}$ and an LHA $H \in \mathcal{H}$, the* projection π_H *of a synchronized run σ of $\mathcal{H}$ over H is obtained by projecting the states and the assignments occurring in σ on the H component and substituting transitions labeled with events not accepted by H with stuttering transitions*[4].

The following theorem states the relationship between the two semantics[5].

Theorem 1. *Given a synchronized run σ, the tuple of projections $\langle \pi_{H_1}(\sigma), \ldots, \pi_{H_n}(\sigma)\rangle$ on the different components is a shallowly synchronized run. Vice versa, given a shallowly synchronized run θ, there exists a synchronized run σ such that $\langle \pi_{H_1}(\sigma), \ldots, \pi_{H_n}(\sigma)\rangle$ is a refinement of θ.*

As corollary, there exists a strictly synchronized run reaching $q_{H_1} \times \cdots \times q_{H_n}$ iff there exists a shallowly synchronized run θ such that for all i, $1 \leq i \leq n$, θ_{H_i} reaches q_{H_i}.

4 Symbolic Encoding

In this section, first, we recall how linear hybrid automata and their reachability problem can be encoded symbolically; second, we show how we can encode symbolically the problem for a network with strict and shallow synchronization.

4.1 Symbolic encoding for single automaton

In the following, in order to encode the flow condition into a quantifier-free formula, we assume the convexity of the invariant conditions. The symbolic encoding of a single LHA consists of three formulas representing respectively the initial, the transition, and the invariant condition. The encoding uses the following additional variables: a discrete variable loc that represents the current location; a real-valued variable δ that represents the time elapsed at the current step; a discrete variable l that represents the label taken at the current step; and two distinguished values T and S, representing a timed transition and stuttering, respectively.

[4] The projection is well defined because if $\langle q_{i-1}, s_{i-1}\rangle \xrightarrow{a_i} \langle q_i, s_i\rangle$ occurs in σ and a_i is not a label of H, then the H components of q_{i-1} and s_{i-1} are equal to the H components of q_i and s_i respectively. Thus, the transition can be locally substituted with a stuttering transition.

[5] An extended version with proofs can be find at `http://es.fbk.eu/people/tonetta/papers/forte10/`

The encoding consists of the following formulas:

$$\begin{aligned}
\textsc{Init} &:= \bigwedge_{q \in Q} (loc = q \rightarrow I_q(X)) \\
\textsc{Invar} &:= \bigwedge_{q \in Q} (loc = q \rightarrow Z_q(X)) \\
\textsc{Trans} &:= \bigwedge_{q \in Q} (loc = q \rightarrow (\textsc{Stutter} \vee \textsc{Timed}_q \vee \bigvee_{(q,p) \in E} \textsc{Untimed}_{q,p})) \\
\textsc{Stutter} &:= l = \textsc{s} \wedge \delta = 0 \wedge loc' = loc \wedge X' = X \\
\textsc{Timed}_q &:= l = \textsc{t} \wedge \delta > 0 \wedge loc' = loc \wedge F_q(\frac{X' - X}{\delta}) \\
\textsc{Untimed}_{q,p} &:= l = L_{q,p} \wedge \delta = 0 \wedge loc' = p \wedge J_{q,p}(X, X')
\end{aligned}$$

Given a reachability problem and a bound k on the length of the runs, we can encode the bounded reachability problem into a formula which is satisfiable iff there exists a run reaching the target condition. We assume to have a formula $\textsc{Target}$ encoding the target condition. For example, if we want to check the reachability of the location q, we can set $\textsc{Target} := loc = q$.

As usual in BMC, we introduce $k+1$ copies of every variable in the encoding of the automata. Then, the reachability problem can be encoded into the following formula:

$$\textsc{Bmc}^k := \textsc{Init}^0 \wedge \textsc{Invar}^0 \wedge \bigwedge_{0 \leq i < k} \left(\textsc{Trans}^i \wedge \textsc{Invar}^{i+1}\right) \wedge \textsc{Target}^k$$

where ϕ^i means that the current and next variables of ϕ have been substituted with their i-th and $(i+1)$-th copy, respectively.

When we consider a network, we use $\textsc{Bmc}^k_H$ to refer to the encoding of the problem for the automaton H.

4.2 Symbolic encoding based on interleaving

In principle, it would be possible to generate the automaton corresponding to the composition of two or more LHAs, and use the above encoding. A more reasonable encoding for a network is based on the encoding of each LHA in the network. The idea is to simply conjunct the encodings forcing the shared event variables to be true exactly at the same steps, and forcing the processes to "stutter" when they are not activated. We assume that the variable δ is shared among the encodings of the different automata.

The reachability problem with a bound k can be encoded as

$$\textsc{BmcInt}^k_{\mathcal{H}} := \textstyle\bigwedge_{1 \leq j \leq n} \textsc{Bmc}^k_{H_j} \wedge \textsc{StrictSync}^k_{\mathcal{H}}$$

where $\textsc{StrictSync}$ guarantees that for every pair of processes j and h, every shared event and the timed event occur at the same step in the two processes, and while a

non-shared event occurs in one process, the other process must stutter[6]:

$$\begin{aligned}\textsc{StrictSync}^{k}_{\mathcal{H}} := \textstyle\bigwedge_{1\le j<h\le n}\bigwedge_{0\le i<k} & \bigwedge_{a\in U_j\cap U_h}(l^i_j = a \leftrightarrow l^i_h = a)\\ & \wedge \bigwedge_{a\in U_j\setminus U_h}(l^i_j = a \rightarrow l^i_h = \textsc{s})\\ & \wedge \bigwedge_{a\in U_h\setminus U_j}(l^i_h = a \rightarrow l^i_j = \textsc{s})\\ & \wedge\ (l^i_h = \textsc{t} \leftrightarrow l^i_j = \textsc{t})\end{aligned}$$

The encoding is compositional in the sense that each automaton is individually encoded. However, the necessity of stuttering on non-shared events and of performing shared events in the same steps may cause complex runs (as shown in Fig. 1).

We also consider a variant of the above encoding where we allow discrete transitions in different automata to occur at the same step of the encoding. Basically, with this variant, we do not force a process to stutter when other processes perform either a local event or an event which is not shared by the process. In this cases, we omit the constraints which force to stutter. This encoding corresponds to the *step semantics* used in [12] for encoding the bounded model checking problem of asynchronous systems.

4.3 Symbolic encoding based on shallow synchronization

In this section, we propose an encoding based on shallow synchronization. We let each automaton keep its own copy of the bound k and the elapsed time δ; we do not force processes to stutter and we let shared events occur at different (local) steps. This means that each of the local encodings is able to construct a local trace.

The reachability problem with bounds $\overline{k} = \langle k_1, k_2, \ldots, k_n\rangle$ can be encoded as

$$\textsc{BmcSS}^{\overline{k}}_{\mathcal{H}} := \bigwedge_{1\le j\le n}\textsc{Bmc}^{k_j}_{H_j} \wedge \textsc{ShallowSync}$$

where ShallowSync encodes the constraints enforcing that all the paths must be consistent according to Definition 7. In the following, we present different ways to encode ShallowSync. (We assume to be in the case described in Remark 1, but all the encodings that we are showing can be lifted to the general case.)

Encoding based on enumeration The first way to encode ShallowSync is by enumerating all possible combinations of steps on which the synchronization occurs. For example, processes P1 and P2 may synchronize over event a, but a may occur in step 2 for P1, and in step 4 for P2. ShallowSync guarantees that, for all pairs of processes, (i) if a shared event occurs in the first process, then the event must occur also in the

[6] Note that it is not necessary to force at least one process not to stutter.

second process at the same time (possibly in different steps), and (ii) the final time of the two processes is the same:

$$\textsc{ShallowSync} := \bigwedge_{1\le j<h\le n}\bigwedge_{a\in U_j\cap U_h} \bigwedge_{1\le i_j\le k_j}(l_j^{i_j}=a \leftrightarrow \bigvee_{1\le i_h\le k_h} l_h^{i_h}=a \wedge t_j^{i_j}=t_h^{i_h}) \wedge \bigwedge_{1\le i_h\le k_h}(l_h^{i_h}=a \leftrightarrow \bigvee_{1\le i_j\le k_j} l_j^{i_j}=a \wedge t_j^{i_j}=t_h^{i_h}) \wedge \bigwedge_{1<j\le n} t_j^{k_j}=t_1^{k_1}$$

Local reasoning We propose a variant of the previous encoding which can be split into constraints local to each automaton, and one for each step. The encoding uses the following additional variables:

- for each automaton H_j, for each shared label l, a variable $count_{l,j}^i$ to represent how many times l has occurred in H_j before step i;
- for each shared label l, a group of variables $\mathit{occ\_time}_{i,l}$ to represent the time at which the i-th occurrence of l is fired;
- for each shared label l, a variable l_{last} to record how many times l has been fired in the whole run;
- c_{last} to record the time at which the system reaches the target.

Note that the variables without superscript are untimed, in the sense that they do not depend on any temporal step.

The shallow synchronization can be encoded as:

$$\textsc{ShallowSync} := \bigwedge_{1\le j\le n}\bigwedge_{0\le i<k_j} \textsc{ShallowStep}_j^i \wedge \textsc{CounterInit}_j \wedge \left(\bigwedge_{0\le i<k_j} \textsc{CounterStep}_j^i\right) \wedge \textsc{FinalShallow}_j$$

where $\textsc{ShallowStep}_j^i$ states that if in the i-th step, an event l occurs in the j-th process for the g-th time, then the local time of the process must be $\mathit{occ\_time}_{g,l}$:

$$\textsc{ShallowStep}_j^i := \bigwedge_{l\in U_j}(l_j^i=l) \rightarrow \bigwedge_{1\le g\le i}((count_{l,j}^i=g) \rightarrow t_j^i=\mathit{occ\_time}_{g,l})$$

$\textsc{CounterInit}$ and $\textsc{CounterStep}$ encode how the counters evolve:

$$\textsc{CounterStep}_j^i := \bigwedge_{l\in U_j}(l_j^i=l) \rightarrow (count_{l,j}^{i+1}=count_{l,j}^i+1)$$

$$\textsc{CounterInit}_j := (count_{l,j}^0=0)$$

while $\textsc{FinalShallow}$ states that the final values of the counters and the local time must be the same:

$$\textsc{FinalShallow}_j := (\bigwedge_{l\in U_j} count_{l,j}^{k_j}=l_{last}) \wedge (t_j^{k_j+1}=c_{last}))$$

Exploiting richer theories It is possible to represent the above encoding with richer theories introducing uninterpreted functions symbols. In particular we represent the time of the i-th occurrence of a label l as a function $\mathit{occ\_time}_l$ from integers to reals. This way we can rewrite SHALLOWSTEP into

$$\textsc{ShallowStep}^i_j := \bigwedge_{l \in U_j} (l^i_j = l) \rightarrow (t^i_j = \mathit{occ\_time}_l(count^i_{l,j}))$$

5 Related Work

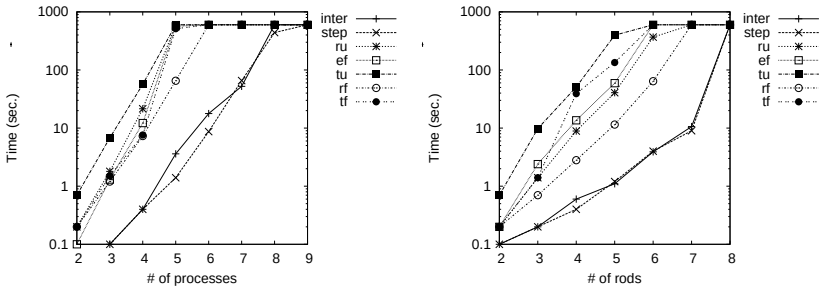
The shallow semantics (defined in [5] and adopted in this paper) bears many similarities with the "local-time" semantics defined in [3] for networks of timed systems and can in fact be seen as a generalization to the hybrid case of [3]. Indeed, neither requires the synchronization of timed transitions of different components; they both use local clocks that are re-synchronized upon shared events. The two semantics differ in the types of runs used to solve the reachability problem: the shallow semantics consists of sets of local runs, while the local-time semantics consists of runs in the interleaving composition. With a mapping similar to the one defined in Section 3, it can be shown that the two semantics are equivalent. As far as we know, this is the first attempt to exploit the shallow/local-time semantics to improve BMC.

Partial-Order Reduction (POR) [11] is one of the most known and used technique to tackle the state-space explosion problem due to interleaving of concurrent systems. The idea is to identify cases when the order of transitions is not relevant in order to prune the search space. The application of POR techniques is difficult in the context of timed and hybrid systems because the timed transitions are global actions which typically interleave the local transition, and thus forbid the pruning performed by POR. The local-time semantics was proposed in [3] to enable POR by removing the synchronization on timed transitions. Other works as in [17] propose symbolic versions of POR and combine them with bounded model checking and SMT. The main difference between POR and the techniques presented in this paper is that while POR tackles the interleaving explosion problem by fixing the order of independent transitions, we allow them to be executed in parallel.

Also related is the "step" semantics, used in [12] for an efficient encoding of the reachability problem in a network of asynchronous systems. The work in [12] is limited to the case of discrete transitions. The idea presented in this paper can be seen as a generalization of the step semantics to the case of timed transitions.

The work described in [16] proposes an event-order abstraction to verify timed automata. The idea is to analyze the discrete and continuous aspects separately by first finding a discrete path causing an error and then computing a set of timing constraints that make the path realistic. Similarly, CEGAR-based approaches such as [1, 14] perform a search on a purely discrete abstraction of the hybrid automaton, and check if the obtained paths are compliant with the original constraints.

The first approach that adopts a shallowly synchronized semantics is presented in [5] for path-oriented bounded reachability analysis of a network of LHAs. In the approach, one path is selected for each component and all selected paths compose a path set for



(a : Star-shape Fischer) (b : Nuclear Reactor)

Fig. 2. Results where the length of a local run depends on the number of processes.

reachability analysis. Each path is independently encoded to a set of constraints while synchronization controls are encoded according to the position of shared labels. By merging all the constraints, the path-oriented reachability problem can be transformed to the feasibility problem of the resulting linear constraint set, which can be solved by linear programming efficiently. This approach has been extended in BACH [6] into a general bounded reachability analysis technique. Different from the approach presented in this paper, this technique traverses the structure of a network of automata using depth-first search and checks the abstract path set one by one.

In the approaches mentioned above, the search is carried out in two stages: in the first, a discrete abstraction of the problem is constructed, while in the second the candidate paths found in the abstract state are checked for consistency in the concrete space. In our approach, the SMT solver carries out the refinement automatically during the search, on demand. With respect to explicit-state search, the symbolic representation is less sensitive to the state-space explosion problem. With respect to abstraction-based techniques, the BMC technique is more tailored to find error paths.

Bounded model checking for hybrid systems using SMT solvers has been investigated in [2, 10, 8, 9]. The characterizing feature of our work is the attempt to leverage the structure induced by the synchronization of a network of hybrid automata.

6 Experimental Evaluation

6.1 Implementation

We implemented the encodings presented in Section 4 within the setting of NUSMT, a model checker that extends NuSMV2 [7] with SMT techniques. The solver used to check the satisfiability of the formulas was MathSat [4], which provides an incremental interface. Thus, the search interacts with the solver to analyze problems of increasing depth. As standard in bounded model checking, we exploit the fact that subproblems at increasing depth share large parts of the encoding: the solver is able to retain information discovered during the previous searches to solve next subproblems more efficiently. We use the following notation to refer to the options: we use `e` for using the enumerative encoding, `r` for using local reasoning, `t` for using local reasoning with uninterpreted

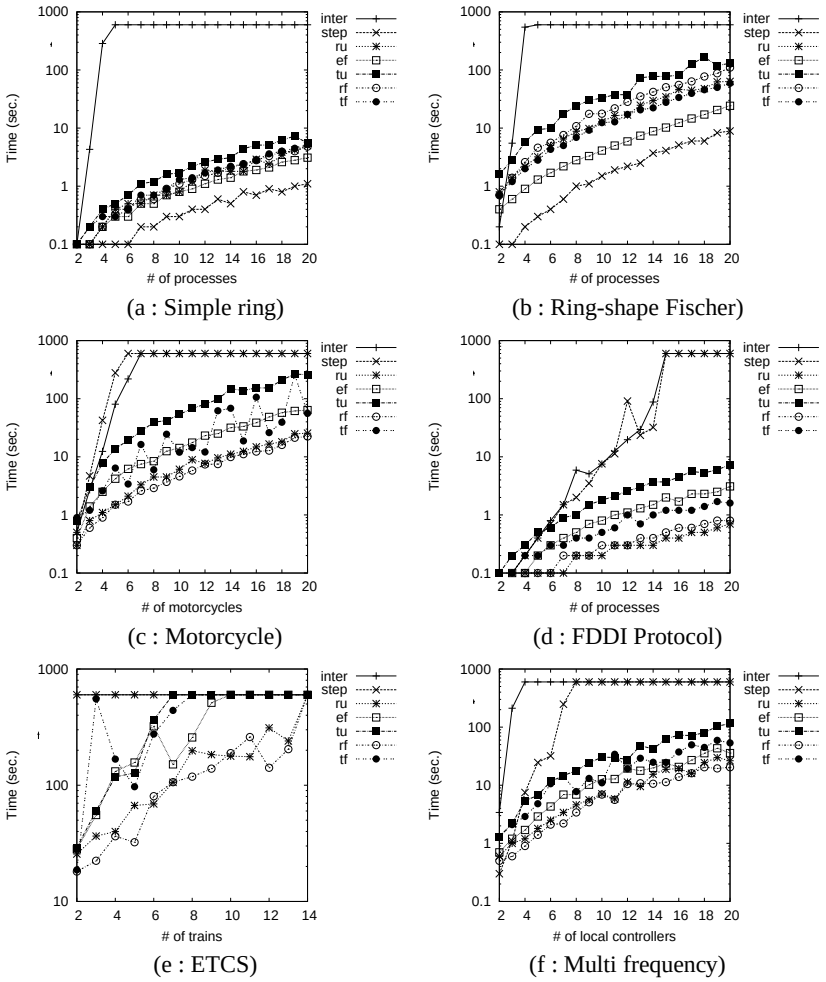


Fig. 3. Results where the length of the local runs does not depend on the number of processes.

functions; with regards to the incrementality, when we use local reasoning, we can add the synchronization constraints during the unrolling (denoted with `u`) or add them after the unrolling (denoted with `f`). Overall, the options are `ru`, `rf`, `ef`, `tu`, `tf` (e.g., `ru` means using local encoding with the constraints added during the unrolling).

6.2 Benchmarks

We test the performance of the shallow synchronization on the following benchmarks:

– *Simple ring*: this example is a simple ring of processes where each process only communicates with its left and right neighbors; it is a proof of concept to show how the shallow synchronization can perform exponentially better than the interleaving.

– *Star-shape Fischer*: this is the hybrid fischer algorithm for the mutual exclusion protocol that uses a shared variable to control the access to a critical session.
– *Ring-shape Fischer*: this variant contains a ring of processes where each process shares a variable with its left and right neighbor; the variables are used to access critical sections in mutual exclusion with the neighbors.
– *ETCS*: this example is inspired by the European Train Control System (ETCS) specification which controls the movement of trains on a track divided into sections. The accelerated motion of the trains is approximated with linear constraints.
– *Motorcycle*: this example is inspired by the automated highway system from [14]. This system models a sequence of n motorcycles. Each motorcycle i needs to wait the signal from the previous one to move, and it needs to keep the sequence during the parade by synchronizing shared labels with neighbors.
– *FDDI Protocol*: this example is a ring topology model based on the system in [19]. It is a set of standards for data transmission on fiber optic lines in a LAN. Each component in the system waits for the signal of previous one to transmit data.
– *Nuclear Reactor*: this example from [18]. The system controls a nuclear reactor with n rods, and uses these rods to absorb neutrons one by one. Each rod that has just been moved out must stay out of the water and cool for several time units.
– *Multi-Frequency*: this example models a global controller that periodically reads the value of a variable from n local controllers, which synchronizes with an high frequency with its environment, and a lower frequency with the global controller.

6.3 Results

We check reachability problems comparing the encodings based on interleaving, step semantics, and shallow synchronization. We compared the results only on reachable instances. For unreachable cases, since we are using a BMC approach, the results strongly depend on the fixed bound, but the meaning of the bound depends on the semantics: for the interleaving, it represents the total number of local and global steps; for the shallow synchronization, it represents the maximum bound of a local run. Thus, any bound would be unfair for either semantics. Nevertheless, note that all algorithms check the unreachability of the target for path lengths smaller than the final one. So, the performance does not depend on the chance of finding the right path. We ran the experiments on a Red Hat 4.1.2 machine, with Intel(R) Core(TM)2 Quad CPU 2.66*4, and 4GB of RAM with a time out of 600 seconds.

The results of the comparison are shown in Figures 2 and 3, where the time to solve the reachability problem is plotted in log scale against the number of automata in the network. Each line corresponds to a particular option. Table 1 shows some of the features of the benchmarks, such as the length of the paths found by reachability analysis as a function of n (the number of processes in the benchmark family). Results are reported for interleaving, step semantic and shallow synchronization.

The main finding of the experimental results is that the efficiency of the bounded model checker depends on necessary depth of the search regardless the adopted semantics. The interleaving performs better than shallow synchronization in the cases where the depth of the search is the same for the different semantics (because one process interacts with all the others and its local run of one process interleaves the synchronization

Benchmark	Path length			Hardest instance attempted		
	Inter	Step	Shallow	Inter	Step	Shallow
Simple Ring	$5n$	6	6	5[TO]	20[1.1]	20 [3.1] - 20 [5.5]
Ring-shape Fischer	$7n$	7	7	5[TO]	20[8.9]	20 [24.2] - 20 [130.2]
Star-shape Fischer	$3n$	$3n$	$3n$	8[TO]	9[TO]	5 [TO] - 6 [TO]
FDDI Protocol	$2n+1$	5	3..5	15[TO]	15[TO]	20 [0.7] - 20 [7.3]
Nuclear Reactor	$4n$	$4n$	$4n$	8[TO]	8[TO]	6 [TO] - 7 [TO]
Motorcycle	$4n+3$	$4n+3$	7..9	7[TO]	6[TO]	20 [22.4] - 20 [259.5]
ETCS	NA	NA	17	2[TO]	2[TO]	7 [TO] - 14 [TO]
Multi-Frequency	NA	$3(n-1)..3n$	9	4[TO]	8[TO]	20 [20.4] - 20 [115.6]

Table 1. Columns 2, 3 and 4 report the length of the path found with the different semantics in function of the number of processes n. Columns 5, 6, 7 report the size of the hardest instance attempted, and, in square brackets, the corresponding time, or "TO" in case of timeout. For Shallow, we report the best and worst result over the different options.

with all other processes): in these cases, the shallow synchronization is penalized by the overhead of the synchronizing constraints. Nevertheless, in many cases (see Fig. 3), the length of local runs do not depend on the number of processes. Thus, using the shallow semantics, we reach the target at same depth. In these cases, the encoding based on shallow synchronization scales exponentially better than the one based on interleaving. The shorter depth of the encoding pays off the overhead due to the more complex synchronizing constraints. The same happens for the step semantics, which is the winner when it is possible to parallelize independent transitions. Among the different options of the shallow synchronization encodings, there is no winner, but using the local encoding added after reaching the target seems to win in most of cases.

We also compared our implementation with BACH, which results to be faster on many examples, while on others it does not terminate with few processes. The comparison does not help in understanding which encoding is more efficient, but rather it confirms that explicit-state search is faster on automata with a small graph, while does not compete on automata with complex graph structure. Finally, we played with different search strategies but they do not modify the outcome of the presented results. All results, together with the binaries and test cases necessary to reproduce them, are available at `http://es.fbk.eu/people/tonetta/tests/forte10/`.

7 Conclusions and Future Work

In this paper we have introduced a novel approach to symbolic reachability in networks of hybrid automata. The approach relies on the shallow synchronization semantics, that preserves the locality of reasoning within each automaton, and forces synchronization between them only when necessary. We discussed how to exploit the features of shallow synchronization in the setting of symbolic bounded model checking, exploiting some advanced features of modern SMT solvers. An experimental evaluation in the setting of linear hybrid automata shows that the proposed encodings are often more scalable than the traditional encodings based on interleaving.

In the future, we will investigate the impact of shallow synchronization to the general case of non-linear hybrid systems. Since automata synchronize only by way of discrete messages, it should be possible to integrate different reasoning engines, with different expressive power, within the same framework. The idea is to selectively apply engines to automata, and to control the search based on the computation cost associated to each tool. Furthermore, we will investigate the application of shallow synchronization in the discrete setting, and its combination with partial order reduction techniques.

References

1. R. Alur, T. Dang, and F. Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst.*, 5(1):152–199, 2006.
2. G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. *Electr. Notes Theor. Comput. Sci.*, 119(2):17–32, 2005.
3. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial Order Reductions for Timed Systems. In *CONCUR*, volume 1466 of *LNCS*, pages 485–500. Springer, 1998.
4. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *CAV*, volume 5123 of *LNCS*, pages 299–303. Springer, 2008.
5. L. Bu and X. Li. Path-Oriented Bounded Reachability Analysis of Compositional Linear Hybrid Systems, manuscript submitted, 2008.
6. L. Bu, Y. Li, L. Wang, X. Chen, and X. Li. BACH2: Bounded reachAbility CHecker for Compositional Linear Hybrid Systems. In *DATE*, pages 1512–1517. EDAA, 2010.
7. A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *CAV*, volume 2102 of *LNCS*, pages 359–364. Springer, 2002.
8. M. Fränzle and C. Herde. Efficient Proof Engines for Bounded Model Checking of Hybrid Systems. *Electr. Notes Theor. Comput. Sci.*, 133:119–137, 2005.
9. M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.
10. N. Giorgetti, G.J. Pappas, and A. Bemporad. Bounded model checking for hybrid dynamical systems. In *DAC*, pages 672–677. IEEE, 2005.
11. P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer, 1996.
12. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.
13. T.A. Henzinger. The Theory of Hybrid Automata. In *LICS*, pages 278–292. IEEE Computer Society, 1996.
14. S. Jha, B. Krogh, J. Weimer, and E. Clarke. Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction. In *HSCC*, volume 4416 of *LNCS*, pages 287–300. Springer, 2007.
15. Roberto Sebastiani. Lazy satisability modulo theories. *JSAT*, 3(3-4):141–224, 2007.
16. U. Shinya. Event order abstraction for parametric real-time system verification. In *EMSOFT*, pages 1–10. ACM, 2008.
17. C. Wang, Z. Yang, V. Kahlon, and A. Gupta. Peephole Partial Order Reduction. In *TACAS*, volume 4963 of *LNCS*, pages 382–396. Springer, 2008.
18. F. Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data structures. *IEEE Trans. Soft. Eng.*, 31(1):38–51, 2005.
19. J. Zhao, X. Li, T. Zheng, and G. Zheng. Removing Irrelevant Atomic Formulas for Checking Timed Automata Efficiently. In *FORMATS*, volume 2791 of *LNCS*, pages 34–45. Springer, 2003.