Compositional Verification of Hybrid Systems with Discrete Interaction using Simulation Relations

Goran Frehse

***Abstract*— Simulation relations can be used to verify refinement between a system and its specification, or between models of different complexity. It is known that for the verification of safety properties, simulation between hybrid systems can be defined based on their labeled transition system semantics. We show that for hybrid systems without shared variables, which therefore only interact at discrete events, this simulation preorder is compositional, and present assume-guarantee rules that help to counter the state explosion problem. Some experimental results for simulation checking of linear hybrid automata are provided using a prototype tool with exact arithmetic and unlimited digits.**

I. Introduction

The complexity of verifying hybrid systems increases exponentially with the number of system components and, particularly, with the number of continuous variables involved, which requires abstraction and divide-and-conquer, i.e., compositional, strategies. Hybrid automata [1] have been successfully used to model and verify hybrid systems. To be able to apply compositional reasoning, we restrict ourselves to hybrid automata with disjunct variables, so that they interact only via synchronization on discrete events. We compare hybrid automata by computing a simulation relation between their states. A state of an automaton simulates that of another if it can exhibit the same discrete and timed behavior. Other than trace or language containment, simulation also captures the branching behavior. In a hybrid setting, it provides a compact and intuitive way to specify desired behavior and can be applied to verify refinement and abstraction, if systems exhibit a continuous behavior too complex to be analyzed or even modeled accurately.

We present an extension of simulation to compare automata of arbitrary alphabets that allows more compact models and proofs, and show that it supports compositional reasoning for hybrid systems with no shared variables. In particular, we use simulation to establish non-circular and circular assume-guarantee rules that do not require non-blocking or receptiveness. Finally, we provide some experimental data obtained with a tool prototype for linear hybrid automata.

In our definition of *simulation* for hybrid automata we follow the approach of [2], which is based on labeled transition system semantics and takes into account a given equivalence relation between states. Simulation without state equivalence was presented for modal hybrid systems in [3]. A rule for *assumme-guarantee reasoning* for Moore machines based on simulation relation has been presented in [4]. It requires non-blocking and is a special case of the assume-guarantee rule in Sect. IV-C. In [5], assume-guarantee reasoning for refinement based on trace inclusion is shown to be sound for receptive timed and hybrid modules. In contrast, we do not require receptiveness, abstract from continuous flows with labeled transition system (LTS) semantics and retain the branching structure of simulation.

The next section defines hybrid automata and their LTS-semantics. Then Sect. III defines simulation relations for hybrid automata with equivalence between states, for which compositional proof rules are given in Sect. IV. Finally, some implementation results are presented in Sect. V.

II. Hybrid Systems with Discrete Interaction

This section briefly recalls basic definitions, which slightly differ from those in [1], [2] in that automata interact only on discrete transitions and have no shared variables.

A. Hybrid Automata

A variable is an identifier that is associated with a real number. This mapping is called a *valuation*. The continuous change of a variable over time is defined by an *activity*:

Definition 2.1 (Valuation, Activity): [1] Given a set Var of variables, a valuation is function $v : Var \to \mathbb{R}$. Let $V(Var)$ denote the set of valuations over Var. An *activity* is a function $f : \mathbb{R}^{\geq 0} \to V(Var)$ in C^∞. Let $act(Var)$ denote the set of activities over the variables in Var. Let $f + t$ be defined for $t \geq 0$ by $(f + t)(d) = f(d + t), d \in \mathbb{R}^{\geq 0}$. A set S of activities is *time-invariant* if for all $f \in S, t \in \mathbb{R}^{\geq 0} : f + t \in S$. Given a set of variables $Var' \subseteq Var$, let the *projection* $v' = v{\downarrow}_{Var'}$ be the valuation over Var' defined by $v'(x) = v(x)$ for all $x \in Var'$. The extension to activities is straightforward.

Hybrid automata are state-transition systems that have variables that can change continuously over time or at discrete events via a discrete transition:

Definition 2.2 (Hybrid Automaton): [1] A *hybrid automaton* (HA) $H = (Loc, Var, Lab, \to, Act, Inv, Init)$ has the following components:

- A finite set Loc of locations.
- A finite set Var of variables. A pair (l, v) of a location and a valuation of the variables is a *state* of the automaton. The state space is $S_H = Loc \times V(Var)$.
- A finite set Lab of synchronization labels,

This research was supported in part by US Army Research Office (ARO) contract no. DAAD19-01-1-0485, and the US National Science Foundation (NSF) contract no. CCR-0121547, and by the German Research Foundation (DFG) grants KO 1430/6-1 and EN 152/29-1.

Goran Frehse is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA gfrehse@ece.cmu.edu

- A finite set of discrete transitions $\rightarrow \subseteq Loc \times Lab \times 2^{V(Var) \times V(Var)} \times Loc$. A transition $(l, a, \mu, l') \in \rightarrow$ is also written as $l \xrightarrow{a,\mu}_H l'$.
- A mapping $Act : Loc \rightarrow 2^{act(Var)}$ from locations to time-invariant sets of activities.
- A mapping $Inv : Loc \rightarrow 2^{V(Var)}$ from locations to sets of valuations.
- A non-empty set $Init \subseteq Loc \times V(Var)$ of initial states such that $(l, v) \in Init \Rightarrow v \in Inv(l)$.

Of particular interest are classes of hybrid automata that can be modeled and analyzed using polyhedra, since efficient algorithms exist for those computations. We will refer to them as *polyhedral hybrid automata* and describe polyhedra with *linear constraints* of the form $\sum_i a_i v_i + b \bowtie 0$, where the a_i and b are integers, the v_i are variables and the sign $\bowtie$ is $<$ or $\leq$. A *linear formula* is a boolean combination of linear constraints and defines a, possibly non-convex, polyhedron. A prominent case of polyhedral hybrid automata are *linear hybrid automata* (LHA) [1], in which the activities have a time-derivative constrained by a linear formula over the time-derivatives of the variables, and the continuous components of invariants, transition relation and initial states can be described by linear formulas.

B. Labeled Transition System Semantics

The safe semantics of a hybrid automaton can defined using an infinite *labeled transition system* (LTS) [2]. The advantage is that, instead of examining the hybrid automaton, we can analyze the LTS. It is substantially simpler because it abstracts from the continuous activities with timed transitions. We define LTSs and then attribute to each hybrid automaton a LTS called its *timed transition system*.

Definition 2.3 (Labeled Transition System): [6] A *labeled transition system* (LTS) $P = (S_P, \Sigma_P, \rightarrow, S_{P0})$ consists of a set S_P of states, a set of labels Σ_P, a transition relation $\rightarrow \subseteq S_P \times \Sigma_P \times S_P$ and a set of initial states $S_{P0} \subseteq S_P$.

Definition 2.4 (Timed Transition System): [2] The *timed transition system* (TTS) of a hybrid automaton H is the LTS $[\![H]\!] = (S_H, Lab \cup \mathbb{R}^{\geq 0}, \rightarrow_{[\![H]\!]}, Init)$ where

- $(l, v) \xrightarrow{\alpha}_{[\![H]\!]} (l', v')$ if and only if $l \xrightarrow{\alpha,\mu}_H l', (v, v') \in \mu, v \in Inv(l), v' \in Inv(l')$,
- $(l, v) \xrightarrow{t}_{[\![H]\!]} (l, v')$ if and only if there exists $f \in Act(l), f(0) = v, f(t) = v'$, and $\forall t', 0 \leq t' \leq t : f(t') \in Inv(l)$.

C. Discrete Interaction

Often a system can divided into several components, each of which is then modeled by a separate automaton. In this paper we restrict ourselves to systems that have no shared variables and therefore only interact at discrete events by synchronizing on transitions with common labels. The interaction is formally defined with a composition operator:

Definition 2.5 (Parallel Composition): [1], [7] Given hybrid automata $H_i = (Loc_i, Var_i, Lab_i, \rightarrow_i, Act_i, Inv_i, Init_i), i = 1, 2$ with disjunct variables, their *parallel composition* $H_1 || H_2$ is the hybrid automaton $H = (Loc_1 \times Loc_2, Var_1 \cup Var_1, Lab_1 \cup Lab_2, \rightarrow_H, Act, Inv, Init_1 \times Init_2)$ with

- $f \in Act(l_1, l_2)$ iff $f{\downarrow}_{Var_i} \in Act_i(l_i)$, $i = 1, 2$,
- $v \in Inv(l_1, l_2)$ iff $v{\downarrow}_{Var_i} \in Inv_i(l_i)$, $i = 1, 2$, and
- a transition $(l_1, l_2) \xrightarrow{a,\mu}_H (l'_1, l'_2)$ exists with $\mu = \{(v, v') | (v{\downarrow}_{Var_i}, v'{\downarrow}_{Var_i}) \in \mu_i\}$ iff for $i = 1, 2$ holds
  - $a \in Lab_i$: $l_i \xrightarrow{a,\mu_i}_i l'_i$.
  - $a \notin Lab_i$: $l_i = l'_i$, $\mu_i = \{(v, v') | v{\downarrow}_{Var_i} = v'{\downarrow}_{Var_i}\}$.
- $((l_1, l_2), v) \in Init$ iff $(l_i, v{\downarrow}_{Var_i}) \in Init_i$, $i = 1, 2$.

For the compositional analysis it will be essential to perform the same operation on the LTS level:

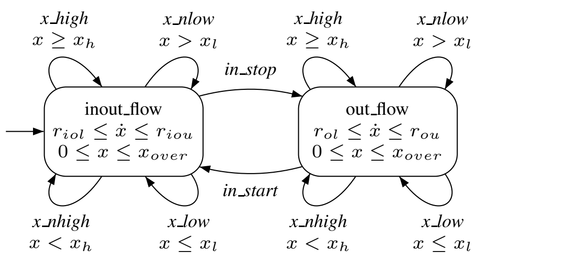
Definition 2.6 (Parallel Composition of LTS): [6] Given labeled transition systems $P_i = (S_{P_i}, \Sigma_{P_i}, \rightarrow_{P_i}, S_{Pi0})$, i=1,2, their *parallel composition* is the LTS $P_1 || P_2 = (S_{P_1} \times S_{P_2}, \Sigma_{P_1} \cup \Sigma_{P_2}, \rightarrow, S_{P_10} \times S_{P_20})$ with a transition $(p_1, p_2) \xrightarrow{\alpha} (p'_1, p'_2)$ iff for i=1,2 holds

- $\alpha \in Lab_i$: $p_i \xrightarrow{a}_{P_i} p'_i$.
- $\alpha \notin Lab_i$: $p_i = p'_i$.

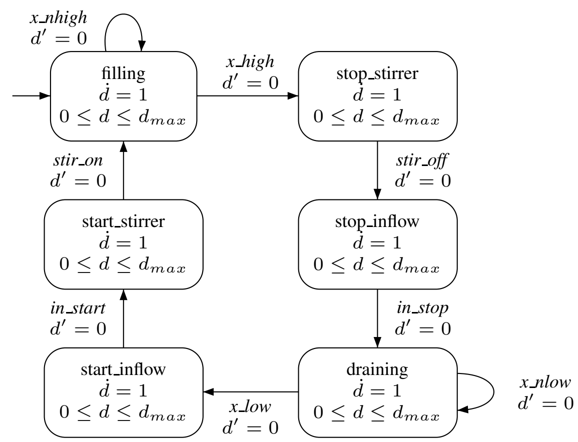
Example 2.1: Consider a chemical reactor with a continuous outflow, a stirrer and a level monitor controller, for which LHA-models are shown in Fig. 1. The controller switches a discrete inlet valve on and off, modelled by labels in_start and in_stop, and is supposed to prevent overflow in the reactor and to operate the stirrer only when there is an inflow. It operates at a maximum sampling time of d_{max} and checks for the level x in the reactor via two discrete sensors at positions x_l and x_h. The sensors are modelled as part of the reactor and trigger events via the labels x_high, x_nhigh, x_low and x_nlow. If the inlet valve is open, the reactor drains at a net rate between r_{iol} and r_{iou}, and fills at rate between r_{ol}, r_{ou} if it is closed.

III. Simulation Relations for Hybrid Automata

In order to be able to compare two automata P and Q, we define a preorder $\preceq$ such that $P \preceq Q$ if any behavior of P finds a match in Q, formally captured by the existence of a *simulation relation* between their states. A state q *simulates* a state p if the system Q shows the same behavior starting from state q as P does starting from state p. In such a comparison, P could be, e.g., an implementation and Q a specification, or P a refined model and Q a more abstract model. Since for safety properties of a hybrid automaton it is sufficient to examine the behavior of its associated LTS, we also define simulation based on the LTSs, following the approach in [2]. For a state q to simulate a state p, an outgoing transition in p must be matched by a transition in q with an identical label. From the TTS semantics it follows that any time elapse should be matched by an identical time elapse. Depending on the application and the meaning that is attributed to the variables in the process of modeling or when designing the specification, it might be desirable to consider certain variables in the system and specification equivalent, which will be illustrated by Example 3.1. This is imposed by requiring that states in the simulation relation are also in a given equivalence relation [2].



(a) Reactor with level sensors R



(b) Level monitoring controller C

Fig. 1. Linear hybrid automaton models

A. *Simulation for arbitrary alphabets*

The classic notion of simulation from [8] requires both automata P and Q to have the same alphabet $\Sigma_P = \Sigma_Q$. We introduce simulation between automata of arbitrary alphabets because it enables smaller models and allows simpler proofs[1]. As will be shown in Sect. IV, it is necessary for compositional proofs that simulation is *invariant under composition*, i.e., that $P \preceq Q$ implies $P||S \preceq Q||S$ for any automaton S. This necessitates two additional conditions for simulation over arbitrary alphabets. We call the classic simulation from [8] into effect as condition (i), for transitions with label $\alpha \in \Sigma_P \cap \Sigma_Q$, (so that if $\Sigma_P = \Sigma_Q$, both notions of simulation are identical) and consider the remaining cases:

(ii) $\alpha \in \Sigma_Q \backslash \Sigma_P$: In parallel composition with another automaton, P cannot block any transitions with label α. Since Q is supposed to be a conservative over-approximation, it shouldn't block either, and therefore must an outgoing transitions with label α in all states.

(iii) $\alpha \in \Sigma_P \backslash \Sigma_Q$: Transitions with label α are allowed as long as they don't eventually lead to states that violate simulation. Therefore the target states of such transitions must themselves be in the relation.

After giving formal definition of simulation for LTS, we will use it to define simulation for hybrid automata:

[1]As far as we know, this extension is a novel approach.

Definition 3.1 (Simulation for LTS): Given an alphabet Σ and LTSs P and Q, $R \subseteq S_P \times S_Q$ is a *simulation relation* iff for all $(p, q) \in R, \alpha \in \Sigma_P \cup \Sigma_Q$ holds either:

(i) $\alpha \in \Sigma_P \cap \Sigma_Q$ and $p \xrightarrow{\alpha} p' \;\Rightarrow\; \exists q' \in S_Q : (q \xrightarrow{\alpha} q' \wedge (p', q') \in R)$

(ii) $\alpha \in \Sigma_Q \backslash \Sigma_P$ and $\exists q' \in S_Q : (q \xrightarrow{\alpha} q' \wedge (p, q') \in R)$

(iii) $\alpha \in \Sigma_P \backslash \Sigma_Q$ and $p \xrightarrow{\alpha} p' \;\Rightarrow\; (p', q) \in R$.

Let $\preceq$ be the largest such relation. A state q *simulates* a state p if $p \preceq q$. Q *simulates* P, denoted as $P \preceq Q$, if for every state $p_0 \in S_{P0}$ there exists a state $q_0 \in S_{Q0}$ such that $p_0 \preceq q_0$, i.e., if for any simulation relation R holds $S_{P0} \subseteq R^{-1}(S_{Q0})$. Such a relation R is then said to *witness* $P \preceq Q$.

Definition 3.2 ($\approx$-Simulation): [2] Given LTS P, Q and an equivalence relation $\approx \subseteq (S_P \cup S_Q)^2$, Q *$\approx$-simulates* P, written as $P \preceq_{\approx} Q$, iff there exists a simulation relation $R \subseteq \approx$ that witnesses $P \preceq Q$. Given HA H_1, H_2, and $\approx$, $H_1 \preceq_{\approx} H_2$, iff $[\![H_1]\!] \preceq_{\approx} [\![H_2]\!]$.

The equivalence relation $\approx$ is usually defined implicitly, e.g., by demanding that certain variables are identical in P and Q. For HA with disjoint variables, the TTS and parallel composition operator are commutative (up to structural isomorphism) if the equivalence relation is given in compliance with the following proposition. This is necessary to carry over the compositional proofs of Sect. IV from LTS to HA:

Proposition 3.1: Given HA H_1 and H_2 with disjoint variables and an equivalence relation $\approx$ such that $(l_1, v_1) \approx (l_2, v_2), v\downarrow_{Var_{H_i}} = v_i$ for $i = 1, 2$ implies $((l_1, l_2), v) \approx ((l_1, v_1), (l_2, v_2))$, it holds that

$$[\![H_1||H_2]\!] \preceq_{\approx} [\![H_1]\!]||[\![H_2]\!] \preceq_{\approx} [\![H_1||H_2]\!]. \qquad (1)$$

B. *Specifying Properties with Simulation Relations*

Frequently, the goal of verifiaction is to establish invariance, sequencing or timing properties. Simulation relations allow to describe all three in an intuitive fashion. The equivalence relation $\approx$ is used to associate variables in the specification with those of the model. In the following example, the tank level in the model is identical to the level variable in the specification, while the timer variables in the specification and the controllers are not related.

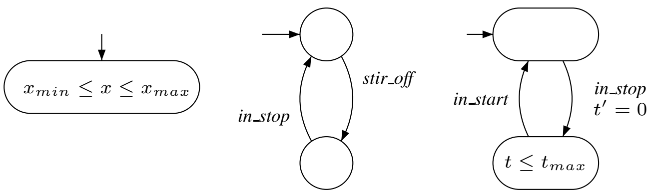
Example 3.1: Consider the tank level monitoring system from Ex. 2.1. Figure 2 shows specification automata for the following properties:

(a) Invariant: The level is always $x_{min} < x < x_{max}$.

(b) Sequencing: A command is sent to turn the stirrer off every time before a command to close the inlet valve.

(c) Timing: The inlet valve is closed for a maximum time of t_{max}.

The specification is fulfilled if $R||C \preceq Q_a||Q_b||Q_v$ holds. Sometimes a specification is expressed easily in terms of a set of forbidden states. A reachability analysis then shows whether the forbidden states can be reached from the initial states of the system. The check for reachability of a set of forbidden states F can be easily combined with checking for simulation by a specification Q. One way is to set $R :=$



(a) Level invariant Q_a (b) Stirrer off before inlet valve closed Q_b (c) Inlet valve closed at most t_{max}, Q_c

Fig. 2. Specification models

$R \backslash (F \times S_Q)$ at the initialization stage of the simulation relation. Alternatively, a label *error* can be introduced with self-loops at the forbidden states. Adding *error* to the alphabet of Q forbids the occurence of those transitions. An advantage of modeling reachability properties with error transitions is that the property can easily be joined with a sequence or other property, e.g. that a fail state can only be reached after a failure prediction system has given alarm.

C. Computing Simulation Relations

A simulation relation can be obtained with a fixed-point computation that removes all states that violate the conditions (i)-(iii) of Def. 3.1. A simple semi-algorithm for computing a simulation relation R is shown in Fig. 3, the reader is referred to [9] for a detailed discussion and more efficient algorithms. While in general reachability and simulation are undecidable for hybrid automata, analysis algorithms terminate for many practical examples, in particular for some polyhedral HA, and techniques are available to force convergence by over-approximation [10], [2]. Before the fixed-point computation of $P \preceq Q$, R can be initialized with the reachable states of $P||Q$ [3], written as $reach_{P||Q}$, or even an over-approximation of those. In many cases, this yields a tremendous speed-up, but sometimes has the reverse effect, see Sect. V.

IV. Compositional Proofs

Most systems of practical interest can be divided into a set of subsystems. A compositional approach to modeling and analysis of such a system is based only on the descriptions of subsystems, without further information about the composed system. Consider the system modeled as $P = P_1|| \ldots ||P_n$ and the specification given by $Q = Q_1|| \ldots ||Q_n$, where the Q_i are considered to be less complex than the P_i. The goal of the compositional proofs is to show $P_1|| \ldots ||P_n \preceq Q_1|| \ldots ||Q_n$ in several steps, which each require the composition of fewer automata and are so less computationally expensive. To that effect, assume-guarantee reasoning deduces the behavior of a composed system from analyses of parts of the system that were made under assumptions about the rest of the system. While the following rules are given for the case $n = 2$, the generalization to arbitrary n is straightforward.

A. Decomposition of the Specification

The first compositional proof is to decompose the specification and verify that $P_1|| \ldots ||P_n \preceq Q_i$ for $i = 1, \ldots, n$. In order to show this, the following lemmas are needed, whose proof is straightforward and omitted due to lack of space:

Lemma 4.1: $P \preceq Q$ if and only if $P \preceq P||Q$.

Lemma 4.2: For any P and Q holds $P||Q \preceq P$.

Theorem 4.1 (Decomposition of Specification): $P \preceq Q_1||Q_2$ if and only if $P \preceq Q_1$ and $P \preceq Q_2$.

Proof: Assume that $P \preceq Q_1$ and $P \preceq Q_2$. With Lemma 4.1, it holds that $P \preceq P||Q_1$. From $P \preceq Q_2$ and invariance under composition follows that $P||Q_1 \preceq Q_2||Q_1$. Transitivity of simulation and commutativity of parallel composition yield $P \preceq Q_1||Q_2$, which proves the sufficient condition. Assume that $P \preceq Q_1||Q_2$. According to Lemma 4.2 it holds that $Q_1||Q_2 \preceq Q_1$ and $Q_1||Q_2 \preceq Q_2$. The conclusion follows directly from the transitivity of simulation. ■

B. Compositionality

A preorder $\preceq$ is called *compositional* if the following rule always holds:[2]

$$\frac{\begin{array}{c} P_1 \preceq Q_1 \\ P_2 \preceq Q_2 \end{array}}{P_1||P_2 \preceq Q_1||Q_2}. \tag{2}$$

Given that the composition operator $||$ is commutative, it is easy to see that a preorder $\preceq$ is compositional iff it is *invariant under composition*, i.e., a *precongruence*. This is the case for $\approx$-simulation and an appropriate equivalence relation:

Proposition 4.1: $\approx$-simulation is a precongruence if the equivalence relation $\approx$ between states is invariant under composition.

Example 4.1: For Ex. 2.1 it holds that $C \preceq Q_b$. By compositionality it follows that $R||C \preceq R||Q_b$, and by decomposing the specification follows $R||C \preceq Q_b$.

procedure *GetSimRel*
Input: labeled transition systems P, Q,
optionally: relation $R_0 \subseteq S_P \times S_Q$
Output: simulation relation R

if R_0 **defined then** $R := R_0$ **else**
optionally $R := S_P \times S_Q$ **or** $R := reach_{P||Q}$ **fi;**
$R' := \emptyset$;
while $R \neq R'$ **do**
$R' := R$
$F_i := \{(p,q) | \exists \alpha \in \Sigma_P \cap \Sigma_Q, p' \in S_P :$
$p \xrightarrow{\alpha} p' \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p',q') \in R\}$;
$F_{ii} := \{(p,q) | \exists \alpha \in \Sigma_Q \backslash \Sigma_P \wedge \nexists q' : q \xrightarrow{\alpha} q' \wedge (p,q') \in R\}$;
$F_{iii} := \{(p,q) | \exists \alpha \in \Sigma_P \backslash \Sigma_Q, p' \in S_P : p \xrightarrow{\alpha} p' \wedge (p',q) \notin R\}$;
$R := R \backslash (F_i \cup F_{ii} \cup F_{iii})$
od.

Fig. 3. Semi-Algorithm for computing a simulation relation

[2]This property is also referred to as modularity. For a detailed discussion and a distinction between compositionality and modularity, see [11].

Often, rule (2) does not allow the Q_i to be much simpler than the P_i, since they must simulate for every possible interaction with the other automata, i.e., without any assumptions about the composed behavior. This motivates assume-guarantee reasoning, of which there are two variants, non-circular and circular.

C. Non-circular Assume-Guarantee Reasoning

In *non-circular assume-guarantee reasoning*, the specification of an automaton serves as the guarantee to the others in the form of a chain rule:

$$\frac{\begin{array}{c} P_1 \preceq Q_1 \\ Q_1||P_2 \preceq Q_2 \end{array}}{P_1||P_2 \preceq Q_1||Q_2}. \tag{3}$$

The proof is straightforward: $P_1 \preceq Q_1 \Rightarrow P_1||P_2 \preceq Q_1||P_2$ due to invariance under composition. According to Theorem 4.1, $P_1||P_2 \preceq Q_1||P_2$ implies $P_1||P_2 \preceq Q_1$. By transitivity follows from $Q_1||P_2 \preceq Q_2$ that $P_1||P_2 \preceq Q_2$. With Theorem 4.1 follows that $P_1||P_2 \preceq Q_1||Q_2$.

Example 4.2: Consider the automaton R as an abstraction of a reactor model $\hat{R}$ with non-linear dynamics, and $\hat{R} \preceq R$ as established manually. Then $R||C \preceq Q_a||Q_c$ can be verified algorithmically, and with (3) and by decomposition of the specification follows $\hat{R}||C \preceq Q_a||Q_c$.

D. Circular Assume-Guarantee Reasoning

In *circular assume-guarantee reasoning*, Q_2 is taken as an assumption about the behavior of P_2 and composed with P_1, and symmetrically Q_1 is composed with P_2. This proof is only sound if additional conditions ensure that $Q_1||Q_2$ does not block transitions that are enabled in $P_1||P_2$.

$$\frac{\begin{array}{c} P_1||Q_2 \preceq Q_1 \\ Q_1||P_2 \preceq Q_2 \\ \textit{A/G conditions} \end{array}}{P_1||P_2 \preceq Q_1||Q_2}. \tag{4}$$

Theorem 4.2 (A/G-simulation): Given that some simulation relation R_1 witnesses $P_1||Q_2 \preceq Q_1$ and some R_2 witnesses $Q_1||P_2 \preceq Q_2$, the relation

$$R = \{\ ((p_1,p_2),(q_1,q_2))\ |\ ((p_1,q_2),q_1) \in R_1 \wedge ((q_1,p_2),q_2) \in R_2\ \} \tag{5}$$

is a simulation relation for $P_1||P_2 \preceq Q_1||Q_2$ if for all $((p_1,p_2),(q_1,q_2)) \in R$ and $\alpha \in \Sigma_{Q_1} \cap \Sigma_{Q_2}$ there exists some q_1' with $q_1 \xrightarrow{\alpha} q_1'$ or some q_2' with $q_2 \xrightarrow{\alpha} q_2'$ whenever

(i) $\alpha \in \Sigma_{P_1} \backslash \Sigma_{P_2}$ and $p_1 \xrightarrow{\alpha} p_1'$,
(ii) $\alpha \in \Sigma_{P_2} \backslash \Sigma_{P_1}$ and $p_2 \xrightarrow{\alpha} p_2'$, or
(iii) $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ and $p_1 \xrightarrow{\alpha} p_1'$ and $p_2 \xrightarrow{\alpha} p_2'$, or
(iv) $\alpha \notin \Sigma_{P_1} \cup \Sigma_{P_2}$.

We refer to the above criteria as the *A/G-conditions*.
Note that R doesn't necessarily contain the initial states. Theorem 4.2 implies that A/G-reasoning is sound if the automata are non-blocking on their common labels. An automaton P is *non-blocking* for a label α if for all states p there exists an outgoing transition with label α.

procedure *CheckAGSimulation*
  Input: labeled transition systems P_1, P_2, Q_1, Q_2
  Output: A/G-simulation relations R_1, R_2

  for (i,j)=(1,2),(2,1) **do**
    optionally $R_i := S_{P_i} \times S_{Q_j} \times S_{Q_i}$ **or** $R_i := reach_{P_i||Q_j||Q_i}$;
    $R_i := R_i \setminus \{\ (p_i, q_j, q_i)|$
      • $\exists \alpha \in \Sigma_{P_i} \cap \Sigma_{Q_i} \cap \Sigma_{Q_j} \backslash \Sigma_{P_j}$:
        $p_i \xrightarrow{\alpha} p_i' \wedge \nexists q_j' : (q_j \xrightarrow{\alpha} q_j') \wedge \nexists q_i' : (q_i \xrightarrow{\alpha} q_i')$ or
      • $\exists \alpha \in (\Sigma_{Q_i} \cap \Sigma_{Q_j}) \backslash (\Sigma_{P_i} \cup \Sigma_{P_j})$:
        $\nexists q_j' : q_j \xrightarrow{\alpha} q_j'$ };
    $R_i :=$ *GetSimRel*$_{P_i||Q_j,Q_i}(R_i)$;
    $D_{P_i} = \{(q_1,q_2,\alpha)|\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2} \cap \Sigma_{Q_1} \cap \Sigma_{Q_2} \wedge \exists p_i :$
      $(p_i,q_j,q_i) \in R_i \wedge p_i \xrightarrow{\alpha} p_i' \wedge \nexists q_i' : q_1 \xrightarrow{\alpha} q_1' \wedge \nexists q_2' : q_2 \xrightarrow{\alpha} q_2'\}$
    od;
  if $D_{P_1} \cap D_{P_2} \neq \emptyset$
    for i=1,2 **do**
      $R_i := R_i \backslash \{(p_i,q_j,q_i)|\exists p_i' : p_i \xrightarrow{\alpha} p_i' \wedge (q_1,q_2,\alpha) \in D_{P_j}\}$;
      $R_i :=$ *GetSimRel*$_{P_i||Q_j,Q_i}(R_i)$
      od;
    fi.

Fig. 4. Algorithm for checking assume-guarantee simulation

Corollary 4.1: Circular A/G-reasoning is sound if Q_1 is non-blocking over Σ_1 and Q_2 is non-blocking over Σ_2 with $\Sigma_1 \cup \Sigma_2 = \Sigma_{Q_1} \cap \Sigma_{Q_2}$.

Checking for A/G-simulation involves the construction of simulation relations R_1 and R_2, and either explicitly constructing R or ensuring that the states in R_1 and R_2 that constitute R fulfill the A/G-conditions. While conditions (i), (ii) and (iv) can be decided strictly from R_1, respectively R_2, (iii) involves both relations. The algorithm shown in Fig. 4 avoids to construct R explicitly by trimming states from R_1 and R_2 that could potentially violate condition (iii). A state p_i in P_i is *potentially violating* if for some $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ there is a transition $p_1 \xrightarrow{\alpha} p_1'$, but no corresponding transition in Q_1 or Q_2. Let the set of *dangerous states and labels* be D_{P_i} with $(q_1,q_2,\alpha) \in D_{P_i}$ if there exists a p_i in P_i that is potentially violating for α and for which $(p_i,q_j,q_i) \in R_i$. The A/G-conditions are fulfilled if for all states and labels for which P_1 has potentially violating states it holds that P_2 does not, i.e., $D_{P_1} \cap D_{P_2} = \emptyset$. The algorithm in Fig. 4 first removes states that violate conditions (i), (ii) and (iv) from R_1 and R_2, then computes dangerous states and labels and removes states from R_i that are dangerous in D_{P_j}. After another fixed-point computation the R_i are simulation relations that fulfill the A/G-conditions. Note that the outcome may depend on whether states are first removed from R_1 or from R_2.

To finalize the A/G-proof, it must be shown that for all $(p_1,p_2) \in P_{01} \times P_{02}$ there exist $(q_1,q_2) \in Init_{Q_1} \times Init_{Q_2}$ such that $(p_i,q_j,q_i) \in R_i$ for $(i,j) \in \{(1,2),(2,1)\}$. It follows from $P_i||Q_j \preceq Q_1||Q_2$ that for any p_i there exists some pair $(q_{1i},q_{2i}) \in Init_{Q_1} \times Init_{Q_2}$, but this must be the *same* pair for both p_1 and p_2. [3] A sufficient condition for the containment is that for all $(p_1,q_2) \in Init_{P_1} \times Init_{Q_2}$

[3]There are cases in which R_1 and R_2 exist, but no simulation relation can be constructed from R_1 and R_2 that contains the initial states appropriately, even though some R' witnesses $P_1||P_2 \preceq Q_1||Q_2$.

and $q_1 \in Init_{Q_1}$ holds $(p_1, q_2, q_1) \in R_1$. Alternatively, a symmetric argument is valid for R_2'.

V. Experimental Results

Algorithms for checking simulation and reachability analysis were implemented in C++ in a prototype tool called PHAVer (Polyhedral Hybrid Automaton Verifier). For computations with convex polyhedra it uses the *Parma Polyhedra Library* (PPL) [12], which provides support for closed and non-closed convex polyhedra and employs exact arithmetic with unlimited digits. The following results were obtained on a 1.9GHz Pentium 4m with 768MB RAM.

A. Performance of Reachability Analysis

The performance of the reachability analysis was compared to HyTech, a powerful model checker for LHA [13]. To ensure balanced comparison, both tools were set to explore the entire reachable state space and check afterwards for intersection with a set of forbidden states. The analysis of Fischer's Mutual Exclusion Protocol from [10], with 5 processes, exact clocks and parameters $t_R = 1$ (waiting time before reserving) and $t_E = 1$ (before entering the critical section) took HyTech 25.8 s (48 MB RAM) and PHAVer 26.2 s (128 MB). For parameters $t_R = 1, t_E = 0$ HyTech took 106.9 s (164 MB), and PHAVer 48.4 s (341 MB).

B. Performance of Simulation Checking

For comparing reachability analysis against simulation, Fischer's protocol was analyzed with for clocks with varying min. and max. speed m, respectively M, and the results are shown in Table I. If the parameters fulfill the specification, the reachability analysis using convex hull (3.) is the fastest. The simulation is comparatively close if the relation is initialized with the convex hull of the reachable state space (6.). Note that here the analysis is slower if the relation is initialized with the exact reachable state space (5.) than if it is not (4.). If the parameters lead to a violation of the specification, the reachability analysis is significantly accelerated by checking at each iteration if forbidden states were encountered (2).

TABLE I

Fischer's mutual exclusion protocol for 4 processes

Algorithm	Time	Memory
(a) $m = 0.99, M = 1.01,\ t_R = 0.99, t_E = 1.01$ (spec. fulfilled)		
1. PHAVer reach.	49.28 s	106 MB
3. PHAVer reach. conv. hull	8.99 s	62 MB
4. PHAVer sim. w/o reach. init.	161.59 s	62 MB
5. PHAVer sim. w. reach. init.	2573.04 s	179 MB
6. PHAVer sim. w. conv. hull reach. init.	15.61 s	63 MB
(b) $m = 0.99, M = 1.01,\ t_R = 1, t_E = 1$ (spec. failed)		
1. PHAVer reach. (full space)	109.40 s	244 MB
2. PHAVer reach. w. stop at forb. states	5.86 s	62 MB
3. PHAVer reach. conv. hull (not sound)	13.76 s	62 MB
4. PHAVer sim. w/o reach. init.	115.54 s	62 MB
5. PHAVer sim. w. reach. init.	$> 10,000$ s	> 300 MB
6. PHAVer sim. w. conv. hull reach. init.	78.25 s	63 MB

VI. Conclusions

The state explosion problem is particularly drastic for hybrid systems because of the complexity arising through continuous variables. We have shown that the established notion of simulation, based on labeled transition system semantics, is compositional for hybrid systems without shared variables. We defined simulation between hybrid automata of arbitrary alphabets, and presented a constructive assume-guarantee rule and an algorithm to ensure soundness without requiring receptiveness. Experimental results using a prototype tool indicate that simulation checking is drastically more expensive than verifying the same property using reachability. However, the compositional application is expected to make up for this deficiency, and we are currently working on an implementation and case studies. Future work includes the extension of the framework to hybrid automata with shared variables by abstracting from the continuous interaction.

References

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, ser. LNCS, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., vol. 736. Springer, 1993, pp. 209–229.

[2] T. A. Henzinger, "The theory of hybrid automata," in *Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996*. IEEE Computer Society Press, 1996, pp. 278–292.

[3] C. Weise and D. Lenzkes, "Weak refinement for modal hybrid systems," in *HART*, ser. LNCS, O. Maler, Ed., vol. 1201. Springer, 1997, pp. 316–330.

[4] T. A. Henzinger, S. Qadeer, S. K. Rajamani, and S. Tasiran, "An assume-guarantee rule for checking simulation," *ACM TOPLAS*, vol. 24, no. 1, pp. 51–64, 2002.

[5] R. Alur and T. A. Henzinger, "Reactive modules," *Formal Methods in System Design*, vol. 15, no. 1, pp. 7–48, July 1999.

[6] R. M. Keller, "Formal verification of parallel programs," *Communications of the ACM*, vol. 19, no. 7, pp. 371–384, July 1976.

[7] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, 1995.

[8] R. Milner, "An algebraic definition of simulation between programs," in *Proc. of the 2nd Int. Joint Conference on Artificial Intelligence. London, UK, September 1971*, D. C. Cooper, Ed. William Kaufmann, British Computer Society, 1971, pp. 481–489.

[9] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke, "Computing simulations on finite and infinite graphs," in *Proc. 36th Ann. Symp. Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 Oct .'95*. IEEE Computer Society Press, 1995, pp. 453–462.

[10] N. Halbwachs, Y.-E. Proy, and P. Raymond, "Verification of linear hybrid systems by means of convex approximations," in *SAS*, ser. LNCS, B. L. Charlier, Ed., vol. 864. Springer, 1994, pp. 223–237.

[11] W.-P. d. Roever, "The need for compositional proof systems: A survey," in *COMPOS*, ser. LNCS, W. P. d. Roever, H. Langmaack, and A. Pnueli, Eds., vol. 1536. Springer, 1998, pp. 1–22.

[12] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill, "Possibly not closed convex polyhedra and the Parma Polyhedra Library," in *Static Analysis: Proc. of the 9th Int. Symposium*, ser. LNCS, M. V. Hermenegildo and G. Puebla, Eds., vol. 2477. Madrid, Spain: Springer, 2002, pp. 213–229.

[13] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: A model checker for hybrid systems," *Int. Journal on Software Tools for Technology Transfer*, vol. 1, no. 1–2, pp. 110–122, Dec. 1997.