# Verification of Hybrid Dynamic Systems Using Linear Hybrid Automata

Akshay Rajhans
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
arajhans@ece.cmu.edu

***Abstract*— This paper concerns the use of linear hybrid automata (LHA) to verify properties of hybrid dynamic systems based on the concept of simulation relations. Following a review of basic concepts and a description of the LHA analysis tool PHAVer, assume-guarantee reasoning is described as a method for compositional verification. The results from the literature are summarized with an example to illustrate the concepts. Finally, the paper outlines some research directions for making this approach more useful.**

## I. Introduction

Dynamic models of computer-controlled physical systems typically require both discrete and continuous state variables to represent the interaction of discrete control logic with physical systems. Hybrid automata have been a widely used formalism for modeling such hybrid dynamic systems. Verifying properties of hybrid automata with arbitrary continuous dynamics is a hard problem, but effective analysis algorithms and tools have been developed for restricted classes of hybrid systems. In this paper, we focus our attention to *linear hybrid automata* (LHA) and discuss the problem of *checking conformance* between a system and its specification. Conformance checking concerns guaranteeing that all behaviors of the system need to be *contained* in the set of behaviors allowed by the specification. The notion of simulation relations, originally defined to study equivalence between two programs, can be used to verify that a given LHA conforms to a specification that is also modeled using another LHA. This paper reviews the concept of simulation relations for LHA where the behaviors of LHA are defined in terms of *timed* traces, i.e. sequences of time elapses and discrete transitions. Because LHAs have a continuum of traces, symbolic algorithms and tools for checking simulation relations between them are needed. Moreover, for complex systems made up of subsystems, we need compositional approaches that check simulation relations on subsystems, instead of on the complete system.

The next section presents the preliminaries. Sec. III defines the formal notion of simulation relations and discusses its applicability to the LHA domain. PHAVer, a tool which implements simulation relation check symbolically on LHA, is presented in Sec. IV. Sec. V presents assume-guarantee (AG) reasoning as one of the ways to approach the problem of checking simulation relations compositionally. Throughout the paper, the theoretical concepts are illustrated using a temperature control system example. Finally, Sec. VI assesses some limitations of the LHA-based simulation relation checking approach and identifies opportunities for further research.

## II. Preliminaries

A *hybrid automaton* (HA) is a tuple ($Loc$, $Var$, $Lab$, $Tran$, $Act$, $Inv$, $Init$), where $Loc$, $Var$, $Lab$ are finite sets of locations, variables and synchronization labels respectively. $Tran$ is a finite set of discrete transitions, each of which is a 4-tuple $(l,a,\mu,l')$ often written as $(l \xrightarrow{a,\mu} l')$. In each transition tuple, $a$ is a label and $\mu$ is the continuous transition relation that relates the values of $Var$ before and after the transition. Let $V(Var)$ denote all possible values of the variables $Var$. An activity is a function $f:\mathbb{R}^+ \to V(Var)$, used to capture the differential dynamics. If the set of all possible activities on $Var$ is denoted as $acti(Var)$, then $Act : Loc \to 2^{acti(Var)}$ is a mapping between a location and its allowed set of activities. The invariant set $Inv : Loc \to 2^{V(Var)}$ is a mapping from locations to their corresponding sets of allowed valuations. The states of HA are pairs $(l,v)$, with $l \in Loc$ and $v \in V(Var)$. A non-empty set $Init \subseteq Loc \times V(Var)$ is the set of initial states such that $(l,v) \in Init \Rightarrow v \in Inv(l)$.

Hybrid automata are a useful framework for modeling computer-controlled physical systems because they can model both the continuous dynamics as well as the discrete control logic using transitions. They are more than just switching systems in that they also have labels on the transitions as synchronizing events. Therefore, the notion of formal verification from the labeled transition systems (LTS) domain can readily be applied and extended in case of HA.

*Linear hybrid automata* (LHA) are HA in which, the activities are constrained by linear formulas over the time-derivatives of the variables, and the continuous components of invariants, transition relation and initial states are described by linear formulas over the variables. The attraction of LHA is that the reachable sets can be exactly computed using polyhedra. Henzinger et al. showed that any complex hybrid dynamics can be approximated arbitrarily well by LHA [8]. This result, coupled with the availability of efficient and exact algorithms for polyhedra makes LHA a potentially useful model for analyzing hybrid systems.

Fig. 1 shows a temperature control system using a thermostat modeled as an LHA. The allowed time derivatives of the room temperature $t$ are defined by $t' \in [0.5, 0.7]$ in the heating mode and $t' \in [-0.4, -0.2]$ in the cooling mode.

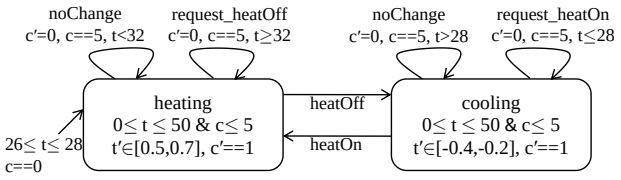


Fig. 1. LHA $H_1$: Thermostat system. $t$ represents the temperature, $c$ represents the clock for sampling. Primed variables inside locations represent the time derivatives. Primed variables on discrete transitions represent the instantaneous change in variables.

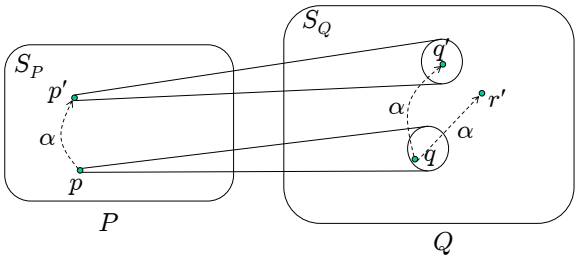


Fig. 2. Pictorial representation of simulation relation.

The allowed time derivative of the clock variable $c$ is $c' = 1$. The thermostat samples the temperature every 5 time units and gives out a `request_heatOn`, `request_heatOff`, or `noChange`, based on where the temperature is with respect to a dead zone ($\pm 2$ degrees) about the temperature set-point (30 degrees). On `heatOn` and `heatOff` the room temperature dynamics switch between heating and cooling.

## III. Simulation Relations

Simulation relations have been used in the transition systems literature for over 20 years as a way to analyze the similarity between LTSs [9]. They are defined as follows.

*Definition 1 (Simulation Relations for LTS):* Given LTSs $P = (S_P, \Sigma_P = \Sigma, \rightarrow, S_{P_0})$ and $Q = (S_Q, \Sigma_Q = \Sigma, \rightarrow, S_{Q_0})$, a relation $\preceq$, which is a subset of $S_P \times S_Q$, is a simulation relation iff $\forall (p,q) \in \preceq$[1], $\forall \alpha \in \Sigma$: $p \xrightarrow{\alpha} p' \Rightarrow \exists q' \in S_Q$ such that $(q \xrightarrow{\alpha} q' \wedge (p', q') \in \preceq)$.

Fig. 2 illustrates this concept pictorially, where the circles in $S_Q$ represent the set of states in $Q$ that simulate the states $p$ and $p'$ in $P$. Every behavior of automaton $P$, signified by a transition on the label $\alpha$, has at least one corresponding matching behavior in $Q$. Frehse generalized this definition to allow cases where $\Sigma_P \neq \Sigma_Q$, i.e. cases where there are internal events in $P$ that are not visible in $Q$ and vice versa [4]. The LTS $Q$ simulates the LTS $P$, written as $P \preceq Q$, if all the initial states are in a simulation relation. Using the definition of simulation relations for LTSs, Frehse defined the simulation relations for LHAs based on their timed transition semantics.

For conformance checking, we want to verify whether the external behavior, i.e. (timed) *traces* starting from some given initial states of an LHA, is contained within those allowed by some specification. To prove this trace inclusion, it suffices to prove whether the *language*, i.e. the set of traces from all possible initial states, is contained in that of the specification. Both trace containment checking and language containment checking are hard, since they involve the entire traces of the sytem. Simulation relations serve as a sufficient condition for language (and therefore trace) inclusion. Furthermore, simulation relations are a local property on the states, which makes them relatively simpler to compute. This property makes them useful for conformance checking.

For example, suppose we wanted to check whether the temperature in the thermostat system LHA $H_1$ from Fig. 1 always remains between 20 and 40 degrees. We could model this specification in terms of another LHA $H_Q$ with a single location whose *Inv* is the desired temperature range $t \in [20, 40]$. Checking whether $t$ always remains in the interval $[20, 40]$ corresponds to checking whether $H_1 \preceq H_Q$.

## IV. Polyhedral Hybrid Automaton Verifier

Polyhedral Hybrid Automaton Verifier (PHAVer) [5] is a tool that implements a symbolic algorithm for finding a simulation relation between two LHAs. In order to perform polyhedral computations efficiently, PHAVer uses the Parma Polyhedra Library (PPL) [1]. PPL supports various set-based operations on polyhedra, such as intersection, union, set difference, embedding into a bigger space, and projection onto a smaller subspace, all of which are useful for analysis of LHA. Moreover, both PPL and PHAVer use exact integer arithmetic with arbitrary precision, which makes the polyhedral computations free from arithmetic errors.

PHAVer has a simple and intuitive textual interface. The textual representation of the LHA from Fig. 1 in PHAVer syntax is:

```
automaton thermostat
state_var: t,c;
synclabs: noChange, request_heatOn, request_heatOff,
heatOn, heatOff;
loc heating: while 0 <= t & t <= 50 & c <= 5
     wait {0.5 <= t' & t' <= 0.7 & c'==1};
when c == 5 & t>=35 sync request_heatOff
     do{c'==0 & t'==t} goto heating;
when c == 5 & t<35 sync noChange
     do{c'==0 & t'==t} goto heating;
when True sync heatOff do {t'==t & c'==c} goto cooling;
loc cooling: while 0 <= t & t <= 50 & c <= 5
     wait {-0.4 <= t' & t' <= -0.2 & c'==1};
when c == 5 & 25<t sync noChange
     do {c'==0 & t'==t} goto cooling;
when c == 5 & t<=25 sync request_heatOn
     do {c'==0 & t'==t} goto cooling;
when True sync heatOn do {t'==t & c'==c} goto heating;

initially: heating & 26 <= t & t <= 28 & c == 0;
end
```

The specification $H_Q$ from the previous section can be written in PHAVer syntax as

```
automaton spec
state_var: t;
loc always: while 20 <= t & t <= 40 wait {True};

initially: always & 20 <=t & t<= 40;
end
```

To construct a simulation relation between two automata, PHAVer uses an algorithm that works with polyhedral sets of

[1] $(p,q) \in \preceq$ is often compactly written as $p \preceq q$.

states at a time, subtracting the set of bad states (states for which, the simulation relation cannot be satisfied). This is done iteratively until a fixed-point is obtained. Given LHA $P$ and $Q$ with states $(k,u)$ and $(l,v)$, $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^m$, let $B^{tr}(k,l)$ be the set of states in $\mathbb{R}^{m+n}$, such that the states from $P$ have no matching *discrete* transitions in $Q$. Similarly, let $B^{te}(k,l)$ be the set of states in $\mathbb{R}^{m+n}$, such that the states from $P$ have no matching *timed* transitions in $Q$. $B^{tr}(k,l)$ and $B^{te}(k,l)$ can be computed using the symbolic operations of intersection, set-difference, projection, embedding and reordering of variables. The simulation relation $R$ is the largest fixed-point of the operation

$$R(k,l) := R(k,l) \cap \neg B^{tr}(k,l) \cap \neg B^{te}(k,l). \qquad (1)$$

In the thermostat example, the question whether `thermostat` $\preceq$ `spec` can be posed to PHAVer using the command `is_sim(thermostat,spec)`, to which PHAVer will return a `Yes` or `No` answer. It turns out, for this example, that the simulation relation fails because the `heatOn` and `heatOff` transitions can happen at any time. Thus, for some evolutions of the thermostat system, it may take arbitrarily long before these transitions occur, and the room may keep cooling or heating, driving the temperatures below or above the specified limits.

## V. Assume-guarantee reasoning

Complex systems are often modeled modularly as a parallel composition of subsystems. In such cases, compositional approaches that analyze only the subsystems are desirable, since the system-wide model and analysis may become too expensive. If all the subsystems satisfy the part of the specification concerning them (which may involve breaking down the overall specification into modular specification on the subsystems), the overall system will also satisfy its specification. However, in case of interconnected systems, subsystems often cannot satisfy their specifications by themselves without making *assumptions* about their environment, i.e. the other subsystems. For example, in the earlier section, the thermostat cannot satisfy the specification by itself, because it can only `request_heatOn` and `request_heatOff`. The events `heatOn` and `heatOff` that decide when the temperature dynamics will actually switch between heating and cooling depend on the environment of the thermostat, and so are beyond the scope of the thermostat subsystem.

In case of systems made up of two subsystems, a non-circular structure arises if one subsystem can make some assumption about the behavior of the other subsystem in order to guarantee the specification, and if the other subsystem guarantees this assumption without needing to make any further assumptions about the first one. This structure is called (non-circular) assume-guarantee (AG) reasoning. In such a case, using AG reasoning, it can be deduced that the overall system will satisfy its specification [4]. This type of reasoning is particularly interesting if the assumption is significantly simpler than the actual subsystem, resulting in an overall reduction in the computational complexity of verification.

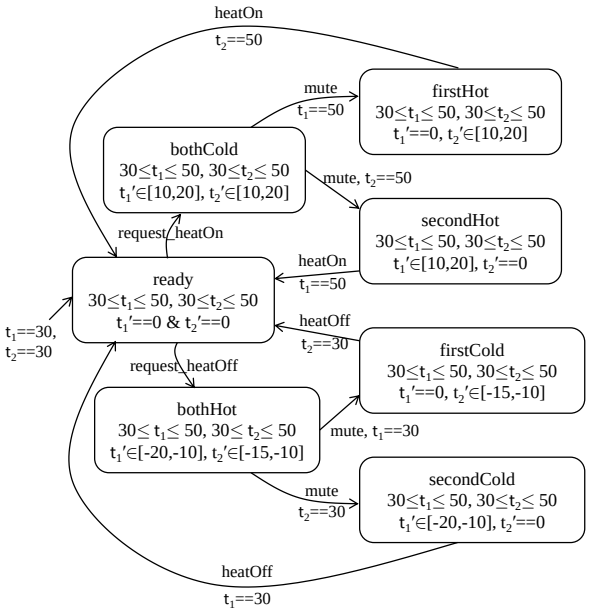


Fig. 3. LHA $H_2$: The furnace. $t_1$ and $t_2$ are the temperature readings at two different points on the furnace heat exchanger. 'mute' is used to label transitions internal to the furnace, that are invisible to the outside world.

Formally, this notion of triangular assume-guarantee (AG) reasoning is summarized from [4] as follows:

$$\frac{\begin{array}{rcl} P_1||A & \preceq & Q \\ P_2 & \preceq & A \end{array}}{P_1||P_2 \preceq Q}$$

In our example, consider that there is another subsystem $H_2$ in the temperature control system. Consider that this second subsystem is a forced-air furnace, with a gas burner which heats up a panel in the heat exchanger and forced air blown through this panel is circulated through the room. Whenever the thermostat executes the command `request_heatOn`, the burners get turned on and the heat exchanger warms up, albeit unevenly. Because of this uneven warming up, there are $n$ temperature sensors deployed at different locations on the heat exchanger. All the sensors have to show that a specified temperature has been reached before the blower can be turned on to start supplying the heat to the room. After the `request_heatOff` event, the burners are switched off immediately, but the blowers are kept on for some time to let the heat exchanger cool. This furnace behavior can be modeled by LHAs, like the one shown in Fig. 3 for $n = 2$.

The thermostat needs to make some assumptions about the furnace behavior before it can satisfy the temperature specification. For $H_Q$ that concerns only the temperature of the room, the internal temperatures inside the heat exchangers are irrelevant. A possible assumption about the furnace, modeled by an LHA $H_A$ as shown Fig. 4, might abstract away these temperature dynamics of the furnace and instead, only make an assumption about the worst case time the furnace will take to get ready.

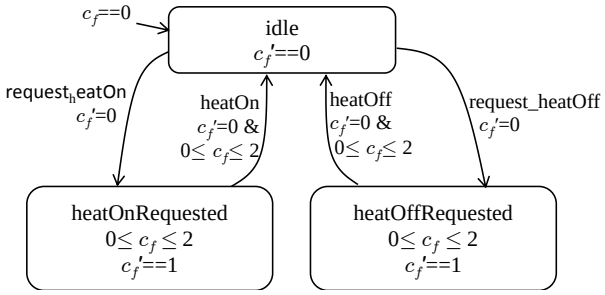


Fig. 4. LHA $H_A$: A candidate furnace assumption. $c_f$ represents a furnace clock to measure the furnace delay.

| | Standard Algoritm | | Only Reachable States | |
|---|---|---|---|---|
| Furnace Model | non-AG | AG | non-AG | AG |
| One sensor | 2.10 | 1.10 | 0.10 | 0.20 |
| Two sensors | 121.45 | 2.80 | 0.40 | 0.30 |
| Three sensors | ∞ | 23.51 | 1.40 | 2.80 |
| Four sensors | ∞ | 272.67 | 23.71 | 96.02 |

TABLE I
EXPERIMENTAL RESULTS SHOWING COMPUTATION TIMES IN SECONDS

We check in PHAVer whether $H_{SIMPLE} = H_1||H_A \preceq H_Q$ and $H_2 \preceq H_A$, and PHAVer returns a `Yes` for both of these confirming that our triangular AG reasoning worked.

Varying the number of sensors $n$ used in the furnace model, experiments were run to compare the time elapsed in verifying the specification without using AG reasoning approach ($H_{SYS} \preceq H_Q$) and using AG reasoning approach ($H_1||H_A \preceq H_Q$ and $H_2 \preceq H_A$). Table I summarizes the results. $\infty$ means that the experiment did not terminate in as long as one day. The first two columns depict the results for the computation of simulation relations using the standard fixed-point algorithm. The results show that the computational complexity increases exponentially with an increasing number of state variables. Notice that as the problem size grows, the AG reasoning approach seems to be clearly more beneficial than the non-AG approach. In order to simplify the simulation relation computation, PHAVer also allows a heuristic for speed-up, which looks only at the states reachable from the given initial conditions. The results with this heuristic are depicted in the last two columns. The heuristic works quite well, but it is not a complete analysis, since it only looks at the behaviors of the system starting from given initial conditions. For this example, the heuristic seems to simplify the simulation relation computation enough to make the two computations in AG slower than the one in non-AG. We conjecture that with increasing size, even in this case AG will eventually outweigh non-AG.

## VI. DISCUSSION

Simulation relation checking of LHA provides a way to carry out verification of hybrid systems with AG reasoning to do it compositionally. However, this approach has some shortcomings and further research is needed to address these shortcomings.

One trouble in computing simulation relations using a fixed-point algorithm is that it is pretty expensive, since all possible states need to be looked at. An alternative to computing simulation relations explicitly as fixed-points would be to develop analytical conditions that will guarantee their existence. For example, there has been some work about the approximate (bi-)simulation for transition systems with observations (outputs) in metric spaces [7]. In this work, the simulation relations become simulation functions, whose level sets give the approximation metrics for simulation relations.

Another issue in AG reasoning is that coming up with appropriate assumption LHAs requires a significant amount of human effort, expertise and intuition. The assumption needs to be both strong enough to satisfy the system specification and weak enough so that it can be satisfied by the component. Tools need to be developed to assist the system designer in developing these assumptions. For purely discrete systems, there has been some work done to automatically generate assumptions by learning the language of the assumption automaton [11], [2], [10]. However, learning the entire language might turn out to be too expensive, as reported in [3]. For LHA, it would be worth investigating whether a user-guided approach can be developed to determine the assumption LHA. One promising approach towards doing this might be to use the procedure in [6] for finding sets of feasible parameters for LHA models, getting the discrete structure as a starting point from the user.

## REFERENCES

[1] The Parma Polyhedra Library. URL: http://www.cs.unipr.it/ppl/.
[2] S. Chaki, E. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In *In Proceedings of CAV05*, volume 3576 of *LNCS*, pages 534–547. Springer-Verlag, 2005.
[3] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Transactions on Software Engineering and Methodology*, 17(2):782–798, April 2008.
[4] G. Frehse. Compositional verification of hybrid systems with discrete interaction using simulation relations. In *IEEE Conference on Computer Aided Control Systems Design*, Taipei, Taiwan, September 2004.
[5] G. Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10(3):263279, July 2008.
[6] G. Frehse, S. K. Jha, and B. H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *HSCC '08: Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control*, pages 187–200, Berlin, Heidelberg, 2008. Springer-Verlag.
[7] A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, May 2007.
[8] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, April 1998.
[9] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
[10] W. Nam, P. Madhusudan, and R. Alur. Automatic symbolic compositional verification by learning assumptions. *Form. Methods Syst. Des.*, 32(3):207–234, 2008.
[11] C. S. Pasareanu, D. Giannakopoulou, M. Gheorghiu Bobaru, J. M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, June 2008.