Project Report - HSATP

Akshay Rajhans

Spring 2009

1 Introduction

Hybrid systems are systems that have both the continuous-time and discrete-time dynamics. The continuous-time dynamics are introdcuded by the physics of physical (most naturally occuring) systems. On the other hand, the discrete-time dynamics are introduced by either the discontinuities of some of the naturally occuring systems or these days by the intelligence embedded into them by added computational elements such as computers or microcontrollers. Analysis of hybrid systems has remained an important research direction and is a key part of what this course focuses on, as the course title might suggest. Of the kind of analyses one could do on hybrid systems, formal verification is one that has been extensively studied by the research community. It has remained a pressing research problem because it is inherently complex, grows with the size of the systems and quickly becomes intractable even on today's computing machinery.

Compositional reasoning is a clever direction that lets one reason about a complex but modular system indirectly by analyzing its parts (modules) without actually having to analyze the whole system. As systems become more and more complex, they will probably be designed in a modular fashion. Today's complex software systems are already being designed and built in a modular way. All this prompts the need to study compositional analysis techniques. Assume-Guarantee (AG) reasoning is a framework that tries to formalize the notions of compositional reasoning. Consequently, if one wishes to apply compositional reasoning in a formal analysis environment, AG reasoning is an important technique that needs to be studied.

Hybrid systems are modeled mathematically by hybrid automata whose semantics are rich enough to capture both the continuous-time and the discrete-time dynamics. Linear hybrid automata (LHA) are an important subclass of hybrid automata for which the formal analysis techniques and tools are known to exist. Moreover, almost all kinds of hybrid systems can be approximated upto arbitrary precision by LHA. Many computational tools have been developed by the research community to analyze linear hybrid automata. Among them is a tool called Polyhedral Hybrid Automata Verifyer (PHAVer) [4], which supports AG reasoning.

One more approach that can be followed while modeling hybrid systems is that of hybrid programs. This proof-theoretic approach presented in class looks at the verification of hybrid systems from a theorem proving perspective. In order to garantee 'good' behavior of hybrid systems, the class introduced two logics, namely *differential dynamic logic* [5] and *differential-algebraic dynamic logic* [6] which form the basis of hybrid systems theorem proving.

In this report, we present the research effort dedicated during the Spring 2009 semester towards the analysis of hybrid systems. In particular, we present a research dig on compositional verification and the illustration of hybrid systems verification on an example from building control domain using PHAVer. Section 2 presents the literature review on compositional verification; while section 3 presents the hybrid systems verification example.

2 Literature Review on Compositional Verification

In this section, we review the literature on compositional verification, particularly of hybrid systems. A good starting point for the research dig in this direction was Goran Frehse's PhD dissertation [1]. The main contributions of his dissertation are (i) developing compositional verification rules for (hybrid) automata for arbitrary alphabets [2] and (ii) AG reasoning for Hybrid I/O Automata (HIOA) by over-approximation of continuous interaction [3] in addition to development of PHAVer. In this section we will review the contributions of these two papers in detail.

2.1 Compositional verification of hybrid automata with no continuous interaction

Let's now look at the paper [2] in detail. It starts with the definition of hybrid automata.

Definition of hybrid automata According to their definition, a hybrid automaton is a tuple $H = (Loc, Var, Lab, \rightarrow, Act, Inv, Init)$, where

- A finite set of locations, Loc
- A finite set of variables, Var
- A finite set of synchronization labels, Lab
- A finite set of discrete transitions $\rightarrow \subseteq Loc \times Lab \times 2^{V(Var) \times V(Var)} \times Loc$, where $V(Var)$ is the set of all possible valuations of Var
- A mapping $Act : Loc \rightarrow 2^{act(Var)}$ from locations to time-invariant activites
- 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 $(l, v) \in Init \Rightarrow v \in Inv(l)$

Because there are no continuous interactions, the paper makes use of the *labeled Transition System (LTS)* semantics. *Timed Transition System (TTS)* is an LTS that is used to approximate the semantics of HA into a tractable domain. The TTS of a HA H is $[[H]]$ defined as

$[[H]] = (S_H, Lab \cup \mathcal{R}^{\geq 0}, \rightarrow_{[[H]]}, Init)$ where

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

Simulation Relation The simulation relation between two automata P and Q is a preorder $\preceq$ such that $P \preceq Q$ if any behavior of P finds a match in Q. A state q simulates the state p if the system Q shows the same behavior starting in state q as does system P starting in state p. Here P could be an implementation and Q a specification, or P a more refined model and Q a more abstract model.

Simulation Relation for Arbitrary Alphabets Simulation relations according to classical notions are defined only for automata with identical alphabets. This paper relaxes that reqiurement, by allowing arbitrary alphabtes and defines the notion of simulation relation for these more general cases.

In the realm of compositional verification, one key requirement from simulation relations is that they should be invariant under composition. i.e. $P \preceq Q \Rightarrow P||S \preceq Q||S$ for any automaton S. To facilitate this, one needs to put some restrictions on the behaviors of the more general lables from the two automata. These restrictions have been captured as follows in the paper:

Given $(p, q) \in R$ where R is the simulation relation $R \subseteq S_P \times S_Q$, where S_P and S_Q the sets of states of P and Q,

- $\alpha \in \Sigma_P \cap \Sigma_Q$ For all labels that belong to both the alphabets, the notion of simulation relation is the same as the one defined in the classical sense. i.e. $p \rightarrow^{\alpha} p' \Rightarrow \exists q' \in S_Q : (q \rightarrow^{\alpha} q' \wedge (p', q') \in R$
- $\alpha \in \Sigma_Q \backslash \Sigma_P$ Here $\alpha \notin \Sigma_P$. So P cannot block any other automaton S on the label α. Since Q is a conservative overapproximation of P, Q should not be allowed to block S on α. Hence, the restriction imposed is that in Q, there should be an outgoing transition on α in each state, moreover, this new state q' should still be in a simulation relation with state p. i.e. $\exists q' \in S_Q : (q \rightarrow^{\alpha} q' \wedge (p, q') \in R)$
- $\alpha \in \Sigma_P \backslash \Sigma_Q$ Here transitions on label α in P are allowed so long as the original and the target states in P still have a simulation relation of with the same state in Q. i.e. $p \rightarrow p' \Rightarrow (p', q) \in R$

Compositinoal Verification Calculus The circular AG reasoning for compositional verification is given as follows:

Given (i) $P_1||Q_2 \preceq Q_1$ and (ii) $Q_1||P_2 \preceq Q_2$, if certain AG conditions are satisfied, we can conclude that $P_1||P_2 \preceq Q_1||Q_2$.

AG conditions Given that some simulation some simulation relation R_1 witnesses $P_1||Q_2 \preceq Q_1$ and some relation R_2 witnesses $Q_1||P_2 \preceq Q_2$, the relation $R = \{((p_1, p_2), (q_1, q_2))|((p_1, q_2), q_1) \in R_1 \wedge ((q_1, p_2), q_2) \in R_2\}$ 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 $q_1 \rightarrow^{\alpha} q_1'$ or some q_2' with $q_2 \rightarrow^{\alpha} q_2'$ whenever
(i) $\alpha \in \Sigma_{P_1} \backslash \Sigma_{P_2}$ and $p_1 \rightarrow^{\alpha} p_1'$, or
(ii) $\alpha \in \Sigma_{P_2} \backslash \Sigma_{P_1}$ and $p_2 \rightarrow^{\alpha} p_2'$, or
(iii) $\alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$ and $p_1 \rightarrow^{\alpha} p_1'$ and $p_2 \rightarrow^{\alpha} p_2'$, or
(iv) $\alpha \notin \Sigma_{P_1} \cup \Sigma_{P_2}$.

Checking for AG 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 AG conditions. The paper further proposes an algorithm that avoids the constructing R explicitly by trimming states from R_1 and R_2 that could potentially violate condition (iii). Finally, it needs to be shown that there are states from the initial condition sets of the two automata that are contained in the simulation relation R. That concludes the proof.

The paper finally presents some experimental results on PHAVer.

Main contributions of the paper The main contributions of the paper are to come up with a framework that allows having arbitrary alphabets for the automata under composition, and restricting the nature of automata such that the simulation relations will still remain invariant under this new type of composition.

2.2 Compositional verification of hybrid input/output automata

In this subsection, let's look at the contributions of [3]. This paper adds continuous interaction into the mix of compositional reasoning for hybrid systems. To allow for the continuous interactions between two automata, the paper uses the definition of hybrid input-output automata (HIOA). Then the paper gives restrictions on what kind of automata can be composed

Definition of Hybrid IO automata Given a set Var of variables, a valuation $v : Var \rightarrow \mathcal{R}$ maps a real number to each variable. Let $V(Var)$ denote the set of valuations over Var. An activity is a function $f : \mathcal{R}^{\geq 0} \rightarrow V$ in C^{∞} and describes the change of valuations over time. Let $act(Var)$ denote the set of activities over Var. Let $f + t$ be defined for $t \geq 0$ by $(f + t)(d) = f(d + t), d \in \mathcal{R}^{\geq 0}$. A set S of activities is time-invariant if for all $f \in S$, $t \in \mathcal{R}^{\geq 0} : f + t \in S$.

Definition: A hybrid input/output-automaton (HIOA)
$H = (Loc, Var_S, Var_I, Var_O, Lab, \rightarrow, Act, Inv, Init)$ consists of the following:

- A finite set Loc of locations

- Finite and disjoint sets of state and input variables, Var_S and Var_I , and of output variables $Var_O \subseteq Var_S$. Let $Var = Var_S \cup Var_I$. The state space is $S_H = Loc \times V(Var)$, and $(l, v) \in S_H$ a state
- A finite set Lab of labels
- 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 $l \rightarrow_H^{a,\mu} l'$.
- 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 set $Init \subseteq Loc \times V(Var)$ of initial states

Compatibility of HIOA for parallel composition HIOA $H_i = (Loc_i, Var_{S_i}, Var_{I_i}, Var_{O_i}, Lab_i, \rightarrow_i, Act_i, Inv_i, Init_i), i = 1, 2$, are compatible if $Var_{S_1} \cap Var_{S_2} = \emptyset$, and $Var_{I_i} \cap Var_{S_j} \subseteq Var_{O_j}$ for $(i, j) \in (1, 2), (2, 1)$.

Parallel composition of HIOA Given compatible HIOA $H_i = (Loc_i, Var_{S_i}, Var_{I_i}, Var_{O_i}, Lab_i, \rightarrow_i, Act_i, Inv_i, Init_i), i = 1, 2$, their parallel composition $H_1 || H_2$ is the HIOA $H = (Loc_1 \times Loc_2, Var_{S_1} \cup Var_{S_2}, (Var_{I_1} \cup Var_{I_2}) \backslash (Var_{S_1} \cup Var_{S_2}), Var_{O_1} \cup Var_{O_2}, Lab_1 \cup Lab_2, \rightarrow, Act, Inv, Init)$ 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
- $(l1, l2) \rightarrow^{a,\mu} (l_1', l_2')$ with $\mu = \{(v, v') | (v \downarrow_{Var_i}, v' \downarrow_{Var_i}) \in \mu_i, i = 1, 2\}$ iff for $i = 1, 2 : a \in Lab_i \wedge l_i \rightarrow_i^{a,\mu_i} l_i'$, or $a \notin Lab_i \wedge l_i = l - i' \wedge \mu_i = \{(v, v') | v \downarrow_{Vars_i} = v' \downarrow_{Vars_i}\}$
- $((l_1, l_2), v) \in Init$ iff $(l_i, v \downarrow_{Var_i}) \in Init_i, i = 1, 2$

HLTS semantics Like [2], [3] also approximates the semantics of HIOA by Hybrid LTS (HLTS). The behavior of HIOA H is defined by its associated TTS $[[H]]$ which is an HLTS. $[[H]]$ is defined as follows:

The timed transition system (TTS) of a HIOA H is the HLTS $[[H]] = (Loc, Var_S, Var_I, Var_O, \Sigma, \rightarrow_{LH}, Init)$ where $\Sigma = Lab \cup \mathcal{R}^{\geq 0} \cup \varepsilon$ and

- $(l, v) \rightarrow_{LH}^{\alpha} (l', v')$ iff $l \rightarrow_{LH}^{\alpha} l'$, $(v, v') \in \mu$, $v \in Inv(l)$, $v' \in Inv(l')$ (discrete transitions),
- $((l, v) \rightarrow_{LH}^{t} (l', v'))$ iff $l = l'$ and there exists $f \in Act(l)$, $f(0) = v$, $f(t) = v'$ and $\forall t', 0 \leq t' \leq t : f(t') \in Inv(l)$ (timed transitions),
- $(l, v) \rightarrow_{LH}^{\varepsilon} (l', v')$ iff $l = l'$, $v \downarrow_{Var_S} = v' \downarrow_{Var_S}$, $v, v' \in Inv(l)$ (environment transitions).

Due to the loss of information about the continuous activites, the operations [[ ]] and || are not commutative. In this direction, the paper states a lemma that for any HIOA H_1 and H_2, every transition in $[[H_1||H_2]]$ implies a corresponding transition in $[[H_1]]\ ||\ [[H_2]]$.

Simulation of HIOA For deciding the preorder $P \preceq Q$, where P and Q are HLTSs s.t. $P = (Loc_P, Var_{S_P}, Var_{I_P}, VarO_P, \Sigma_P, \rightarrow_P, Init_P)$ and $Q = (Loc_Q, Var_{S_Q}, Var_{I_Q}, Var_{O_Q}, \Sigma_Q, \rightarrow_Q, Init_Q)$.

Simulation relation Given P comparable with Q, a relation $R \subseteq S_P \times S_Q$ is a simulation relation iff $R \subseteq \{(k,u,l,v) | u \downarrow_{Var_{O_Q}} = v \downarrow_{Var_{O_Q}} \wedge u \downarrow_{Var_{I_Q}} = v \downarrow_{Var_{I_Q}}\}$ and $\forall (p,q) \in R$, $\alpha \in \Sigma_P$, $p' \in S_P$, $p \rightarrow^{\alpha} p' \Rightarrow \exists q' \in S_Q : (q \rightarrow^{\alpha} q' \wedge (p',q') \in R)$.

Compositional verification calculus for HIOA Circular AG Reasoning

Consider HIOA P_1, P_2, Q_1, Q_2 (P_i comparable to Q_i).

If there exist simulation relations R_1 for $[[P_1]]\ ||\ [[Q_2]] \preceq [[Q_1||Q_2]]$ and R_2 for $[[P_2]]\ ||\ [[Q_1]] \preceq [[Q_1||Q_2]]$ such that for all $((k_1,k_2,x),(l_1,l_2,z))$ for which there exist $(\hat{l_1}, \hat{z_1})$, $(\hat{l_2}, \hat{z_2})$ with $((k_i,l_j,y_i),(l_i,\hat{l_j},\hat{z_i})) \in R_i$, $(i,j) \in \{(1,2),(2,1)\}$ and $\forall \alpha \in \Sigma_{P_1} \cap \Sigma_{P_2}$, if the following result holds:

$$(k_1,k_2,x) \rightarrow^{\alpha}_{P_1||P_2} (k_1', k2', x') \Rightarrow$$
$$[\exists i, l_i', z' : (l_i, z \downarrow_{Q_i}) \rightarrow^{\alpha}_{Q_i} (l_i', z' \downarrow_{Q_i}) \wedge z' \downarrow_{P_j \cap Q_i} = x' \downarrow_{P_j \cap Q_i}]$$

then a simulation relation for $P_1||P_2 \preceq Q_1||Q_2$ is given by

$$R = \{((k_1,k_2,x),(l_1,l_2,z)) | \exists y_i, \hat{l_j}, \hat{z_j} :$$
$$((k_i,l_j,y_i),(l_i,\hat{l_j},\hat{z_i})) \in R_i, y_i \downarrow_{P_i} = x \downarrow_{P_i}, y_i \downarrow_{Q_j} = z \downarrow_{Q_j}, \hat{z_i} \downarrow_{Q_i} = z \downarrow_{Q_i}\}.$$

The paper then proposes a procedure to come up with R_1 and R_2. To finalize a proof then, it needs to be shown that the initial conditions lie in the simulation relation.

Main contributions of the paper The main contribution of the paper is the simulation relation of HIOA in terms of their corresponding HLTS (TTS) semantics.

2.3 Assumption Learning

In both the papers [2] and[3], the assumptions for the individual components in AG reasoning proofs needed to be known. The papers did not talk about how to find a right assumption for the individual automata (also called 'modules').

Towards this direction, we looked at the paper [7]. This paper makes two contributions - (i) automatic decomposition of a complex system into its modules and (ii) learning of assumptions for individual components by posing it as a DFA learning problem and use L* algorithm for doing this. We are interested in (ii) and will briefly review it here.

The $L*$ algorithm learns an unknown regular language and generates a minimal DFA accepting the language by asking membership and equivalence queries to a teacher. The algorithm infers the structure of the DFA by asking a teacher [1] membership and equivalence queries.

Figure 1 illustrates the $L*$ algorithm. Let U be the unknown regular language and $Sigma$ be its alphabet. At any given time, the $L*$ algorithm has, in order to construct a conjecture DFA, information about a finite collection of strings over Σ, classified either as members or non-members of U. This information is maintained in an observation table (R, E, G) where R is a set of representative strings for states in the conjecture DFA such that each representative string $r_q \in R$ for a state q leads from the initial set (uniquely) to the state q. E is a set of experiment suffixe strings used to distinguish states. G maps strings $\sigma = \sigma.\sigma_2$ (where $\sigma_1 \in R \cup R \cdot \Sigma$ and $\sigma_2 \in E$) to 1 if σ is in U, and to 0 otherwise.

The algorithm is nicely explained with the help of an example in the paper. Suppose the task is to learn a regular language U over the alphabet $\{a, b\}$ with an even number of a's and an even number of b's.

Initially, the algorithm sets R and E to $\{\varepsilon\}$, and asks membership queries for the strings ε, a and b. The initial observation table G_1 is shown in Fig. 2a. This observation table is not closed as there can be no DFA corresponding to this table. So the algorithm adds the string a to the set R and then asks membership queries again for the new strings aa and ab. The resulting observation table G_2 is shown in Fig. 2b, where the new sub-block of rows gets added because of adding a to R. For this table, the algorithm can constructs a conjecture DFA C_1, shown in Fig. 2c. The algorithm then asks the teacher an equivalence query-asking whether C_1 is the correct DFA. The teacher provides a counter-example, say bb; $bb \in U \backslash L(C_1)$.

The function *findSuffix()* takes in the counterexample bb and returns the string b to be added to the experimental string set E. Then, it again updates the observation for b. This is depicted by the addition of a column corresponding to the string b as shown in table G_3 in Fig. 3a. However, G3 is not closed and there can be no DFA corresponding to it. Then the algorithm adds b to the set R and then asks membership queries for the newly formed strings ba and bb. The result observation table is G4 (Fig. 3b). The algorithm constructs a DFA C_2 based on G_4, as shown in Fig. 3c. However, C_2 is again not a correct DFA and the teacher provides a counter-example, say aba; $aba \in L(C2) \backslash U$.

[1] In their impementation, they use a model-checking problem in NuSMV to be their teacher.

```
R := {ε};
E := {ε};
G[ε, ε] := member(ε·ε);
foreach (a ∈ Σ) { G[ε·a, ε] := member(ε·a·ε); }
repeat:
    while ((r_new := closed(R, E, G)) ≠ null) {
        add(R, r_new);
        foreach (a ∈ Σ), (e ∈ E) { G[r_new·a, e] := member(r_new·a·e); }
    }
    C := makeConjectureDFA(R, E, G);
    if ((cex := equivalent(C)) = null) then return C;
    else {
        e_new := findSuffix(cex);
        add(E, e_new);
        foreach (r ∈ R), (a ∈ Σ) {
            G[r, e_new] := member(r·e_new);
            G[r·a, e_new] := member(r·a·e_new);
        }
    }
```

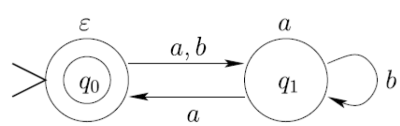
Figure 1: L* algorithm for learning regular languages

G_1		E: ε
R	ε	1
$R\cdot\Sigma$	a	0
	b	0

(a) G_1

G_2		E: ε
R	ε	1
	a	0
$R\cdot\Sigma$	a	0
	b	0
	aa	1
	ab	0

(b) G_2



(c) C_1

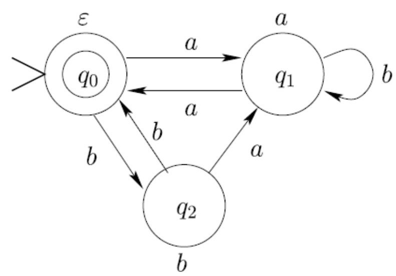
Figure 2: Observation tables and conjecture machines (part 1 of 3)

G_3		E	
		ε	b
R	ε	1	0
	a	0	0
$R \cdot \Sigma$	a	0	0
	b	0	1
	aa	1	0
	ab	0	0

(a) G_3

G_4		E	
		ε	b
R	ε	1	0
	a	0	0
	b	0	1
$R \cdot \Sigma$	a	0	0
	b	0	1
	aa	1	0
	ab	0	0
	ba	0	0
	bb	1	0

(a) G_4



(c) C_2

Figure 3: Observation tables and conjecture machines (part 1 of 3)

The function *findSuffix()* again takes in the counterexample *aba* and returns the string a as a new experiment string. The algorithm adds it to the set E. This is shown by the addition of one more column. (see G_5 in Fig. 4a). G_5 is not closed, so the algorithm adds the string *ab* to the set R and then asks membership queries for the new strings *aba* and *abb*. From the result, the table is updated as G_6 shown in Fig. 4b. This observation table is now closed. The corresponding DFA is C_3, shown in Fig. 4c. This time the teacher answeres the equivalence query with a *yes*, since C_3 is the correct DFA and the algorithm terminates.

2.4 How $L*$ algorithm can be applied to learn assumptions for compositional verification

The paper considers two versions of compositional verification rules, viz. the simple rule $Rule - S$ for two modules, and the general rule $Rule - G$ for n modules. The rule $Rule - S$ is to prove that a composition of two modules, $M1||M2$ satisfies a safety property φ over X^{IO}, which is the union of the finite sets of input and output variables communicated by the two modules M_1 and M_2. The rule states that there exists some module A such that A is safe (i.e. satisfies the property φ) and M_2 refines A then $M_1||M_2$ satisfies φ.

$Rule - S$:

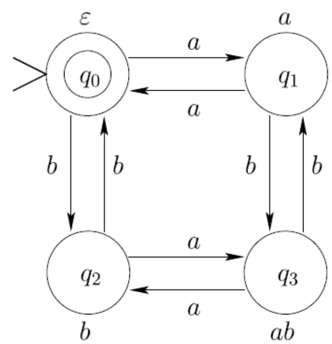
$$\frac{\begin{array}{c}\text{(Pr1-S): } M_1||A \models \varphi \\ \text{(Pr2-S): } M_2 \preceq A\end{array}}{M_1||M_2 \models \varphi.}$$

G_5		ε	b	a
R	ε	1	0	0
	a	0	0	1
	b	0	1	0
$R \cdot \Sigma$	a	0	0	1
	b	0	1	0
	aa	1	0	0
	ab	0	0	0
	ba	0	0	0
	bb	1	0	0

(a) G_5

G_6		ε	b	a
R	ε	1	0	0
	a	0	0	1
	b	0	1	0
	ab	0	0	0
$R \cdot \Sigma$	a	0	0	1
	b	0	1	0
	aa	1	0	0
	ab	0	0	0
	ba	0	0	0
	bb	1	0	0
	aba	0	1	0
	abb	0	0	1

(b) G_6



(c) C_3

Figure 4: Observation tables and conjecture machines (part 1 of 3)

Weakest safe assumption An assumption A is called a safe assumption if the it satisfies the premise Pr1-S (i.e. $M_1||A \models \varphi$), and an assumption A is called an appropriate assumption if the assumption A satisfies both the premises Pr1-S and Pr2-S. The weakest safe assumption W is a module such that it is appropriate (i.e. $M_1||W \models \varphi$) and for all appropriate assumptions A, $L(A) \subseteq L(W)$. For a given module and a safety property, W is guaranteed to exist and is unique.

The paper further mentions two lemmas which state that W is a witness for the truth value of $M_1||M_2 \models \varphi$.

Lemma 1 If $M_1||M_2 \models \varphi$, then the weakest safe assumption W is an appropriate assumption with respect to Rule-S.

Lemma 2 If $M_1||M_2 \not\models \varphi$, then the weakest safe assumption W is not an appropriate assumption with respect to Rule-S.

The weakest safe assumption W can be represented as a DFA with the alphabet Q^{IO} (where Q^{IO} is a set of all states over X^{IO}) because M_1 and M_2 are finite. Therefore, we can learn the weakest safe assumption W which a witness for truth of $M_1||M_2 \models \varphi$ using the $L*$ algorithm for learning regular languages.

The overall scmeme of how this can be done is presented in figure 5, while the algorithm is presented in figure 6.

The paper also provides similar results and the learning algorithm for $Rule-G$. However, for brevity we have not included those here.

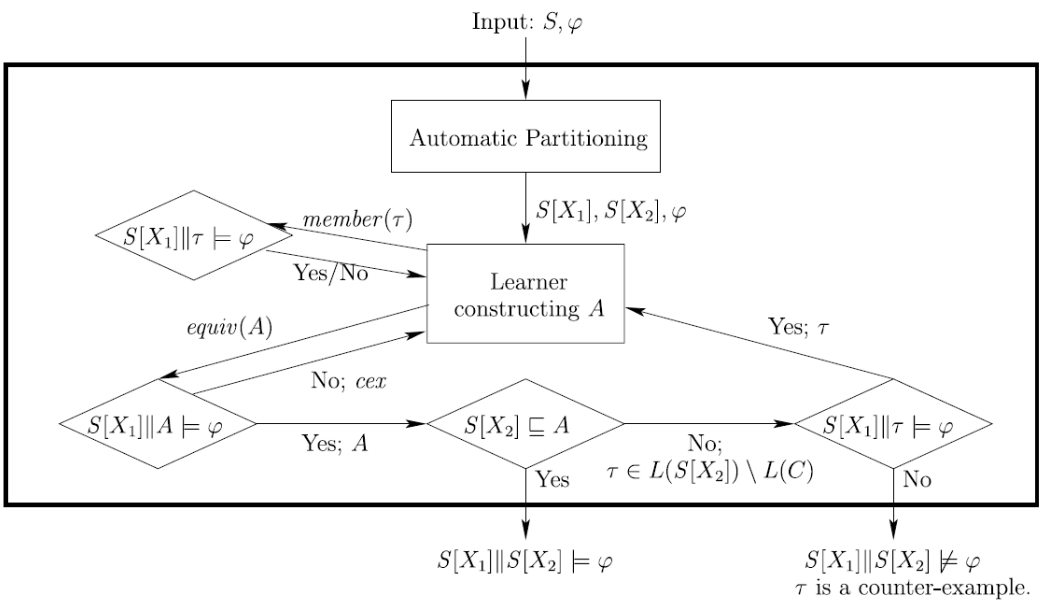


Figure 5: ASCV Algorithm Overview (simple case)

Boolean ASCV_S(S, φ)

$M[\,] :=$ `AutomaticPartitioning`$(S, 2)$;

$A :=$ `InitializeAssumptions`$(M[\,], \varphi)$;

repeat:

    while$((cex :=$ `SafeAssumption`$(M[1], A, \varphi)) \neq null)\{$

        `UpdateAssumption`$(M[1], A, cex)$;

    $\}$

    if$((cex :=$ `Refinement`$(M[2], A)) = null)$ **then** $\{$

        return *true*;

    $\}$ **else** $\{$

        if(`SafeTrace`$(M[1], cex)$) **then** `UpdateAssumption`$(M[1], A, cex)$;

        else return *false*;

    $\}$

Figure 6: ASCV_S Algorithm

3 Hybrid Systems Verification - Experimental Results

In this section we will illustrate a few hybrid systems verification experiments in a building control application.

Consider a temperature control system comprising a space with two zones (rooms), a thermostat and a furnace. The thermostat is physically located in zone 1, so it can only read the temperature in zone 1. The furnace can either be turned on or off manually, and while on, the thermostat can dictate whether or not the furnace should heat the room or not. (The way this is controlled in real systems is by turning a blower forcing hot-air ventilation on or off.) Zone 2 is heated passively through its proximity to zone 1e. Zone 1 and zone 2 both lose heat to the ambient environment. The goal is to maintain the measured temperature of this room close to a specified set point set inside the thermostat.

The furnace in the example does not remember what state it was in when it was turned off, and by default initializes itself in the not-heating mode whenever it is turned on. If the furnace is locally powered off, it forgets what state it was powered off from (heating-the-room or not-heating-the-room) and by default initializes itself into not-heating-the-room when it is powered on. The lack of full information to the thermostat poses a problem in the system behavior. This is because the thermostat issues the 'heatOn' command just once, when it sees that the temperature has dropped below the specified threshold.

One solution to the problem would be to have the thermostat use a time-out to periodically check to see if the temperature of the room temperature is falling when the heat should be on. If the temperature is falling, the thermostat would re-send the heatOn command to the furnace. This solution needs to be verified using model that can determine if the value of the time-out period is appropriate given the heating and cooling dynamics of the room. We use linear hybdir automaton (LHA) model of the system to do analyze this strategy.

The continuous dynamics of the furnace can be modeled to be an ideal heat source while in notHeating and heating modes, i.e. it stays at a constant temperature t_f. The other two possible modes that the furnace can be are - off and warmingUp. While the furnace is warming up, we don't care how the temperature rises, but we do care how long the furnace stays in warmingUp. The ambient temperature is assumed to be constant.

The thermostat periodically checks for the sensed temperature of zone 1, say t_1 to be within the given hysteresis window $t_{set} \pm \delta$ around the temperature set point t_{set}. Once it checks the value of t_1, it signals out commands heatOn or

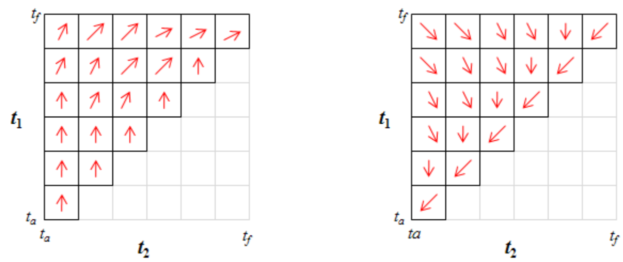


Figure 7: A sample way to approximate the temperature dynamics using LHA

heatOff commands to the furnace, based on whether t_1 respectively falls below or rises above the hysteresis band. This sampling period t_{sample} is an important parameter that ultimately determines the success of the strategy.

The dynamics of the room can be modeled in two differential equations for the temperatures of the two zones as follows.

$$\dot{t_1} = \alpha(t_f - t_1) + \beta(t_2 - t_1) + \gamma_1(t_a - t_1)$$
$$\dot{t_2} = \beta(t_1 - t_2) + \gamma_2(t_a - t_2)$$

Here, α, β, γ_1 and γ_2 are coefficients of diffusion of heat across the furnace, zone 1, zone 2 and the ambient.

Zone 1 gets an input each from the furnace as well as the ambient, while it shares an undirected coupling with zone2. The coupled dynamics of zone1 and zone2 temperatures are affine dynamics and need to be approximated by piecewise constant dynamics of LHA. To do that, the state space can be split into smaller regions, and different values for the different piecewise constant bounds on derivatives can be established for each of the smaller regions.

Figure 7 shows a sample strategy where the state space is split into 6×6 small squares. Since both t_1 and t_2 remain between t_f and t_a, only that part of the state space needs to be considered. Moreover, since zone 2 gets heat from zone 1, its temperature can't be more than that of zone 1. Therefore, the regions where $t_1 < t_2$ can also be ignored.

Depending on whether the furnace is heating zone 1 or not, its dynamics become either positive (t_1 rising primarily due to large α, and smaller β and γ_1 values) or negative (α equals zero, so net drop in t_1). However, in LHA, the dynamics cannot be made dependent on the state. Hence, the states are duplicated into heating or notHeating supermodels as shown in the in the left and right parts of the figure respectively.

Zone 2, on the other hand, is unaware of whether the furnace is in heating or notHeating. It gets its heat from zone 1. Note that the diagram is illustrative only and is intended to give a rough idea about the relative rates of change of temperatures of the two zones. The rate of change of t_1 is different in the two halves, while the rate of change of t_2 stays the same.

Linear hybrid automaton model makes it possible to model the said time-out strategy to resolve the deadlock that might arise due to the lack of full information available to the thermostat. In the LHA model, we could have the thermostat toggle between an 'idle' mode, where it stays until the sampling period and a 'checking mode' where it checks where the temperaure is w.r.t. the setpoint. Based on the temperaure, it makes a decision whether to start heat, stop heat or do nothing and goes back to idle mode, while communicating this decision via labels 'startHeat', 'stopHeat' or 'doNothing' to the furnace. We could add the necessary redundancy based on time-out by adding an extra transition in the thermostat automaton from the 'checking' mode to the 'idle' mode that commands the furnace to 'startHeating'; and the guard of this transition is based on how much time the temperature stays below $t_{set} - \delta$. This is a very simple solution that may resolve the deadlock subject to finding out the correct value of this time window depending on the dynamics.

Following is a PHAVer model of the thermostat automaton that depicts the sampling strategy of the thermostat.

```
automaton thermostat
state_var: c; //c = clock variable
input_var: t1;
synclabs: tick, startHeat, stopHeat, doNothing;

loc idle: while c <= t_sample wait {c'==1};
   when c==t_sample  sync tick do {c'==0}
     goto checking;
loc checking: while c<=1 wait {c'==1};
   when t1<=(t_set - deltaL) sync startHeat
     do {c'==0} goto idle;
   when t1>=(t_set + deltaH) sync stopHeat
     do {c'==0} goto idle;
   when (t_set - deltaL)<=t1&t1<=(t_set + deltaH)
     sync doNothing  do {c'==0} goto idle;

initially: idle & c==0;
end
```

Based on the different parameters of the system, the right value of `t_sample` can be found out, where this sampling strategy works, and the temperature of the room stays within the desired limit.

The appendix A1 presents a the PHAVer codes for the system built in increasing level of detail. Experiments 1 through 4 depict how we incrementally build a fairly detailed model of the said system. Experiment 4 presents the most detailed model featuring a 4×4 decompoition of the temperature state space, similar to the 6×6 shown in the figure 7.

4 Appendix A1: PHAVer codes

Here we attach PHAVer codes of a few experiments we did for modeling the room, thermostat and furnace set-up.

Experiment 1 In this experiment, we model the system as being made up of furnace, thermostat, zone1 and ambient. The LHA models are as follows.

```
//------------------------------------------------------
//     Constants
//------------------------------------------------------

tau_sample := 20;       // sampling time of the controller
tau_spec := 0.0004; // waiting time in the specification
tau_warmup  := 3;

deltaH   := 2;       // delta above t_set for the thermostat to ignore
deltaL      := 2;       // delta below t_set fot the thermostat to ignore

t_set := 14;      // setpoint for the thermostat

t_m         := 15;      // lower bound on the temperature spec
t_M         := 35;      // upper bound on the temperature spec

t_bottom    := 10;

tf_h  := 40; // Hottest furnace temp.
tf_l  := 38; // Hottest furnace temp.

ta_l := 0;
ta_h := 2;

ta_c        := 0;

sqw := (tf_h - ta_l)/4;

v1 := ta_l; // 0
```

```
v2 := v1 + sqw;
v3 := v2 + sqw; // 20
v4 := v3 + sqw;
v5 := v4 + sqw; //40

//--------------------------------------------------
automaton furnace
//--------------------------------------------------
state_var : cf; //tf = temperature of the furnace, cf = clock of the furnace
synclabs: powerOn, powerOff, startHeat, stopHeat, auto;

loc poweredOff: while True wait {cf' == 1};
when cf>=2       sync powerOn  do {cf'==0}  goto warmingUp;
    when True       sync startHeat  do {True}          goto poweredOff;
    when True       sync stopHeat   do {True}          goto poweredOff;

loc warmingUp: while cf <= tau_warmup wait {cf' == 1};
    when cf == tau_warmup   sync auto     do {True}  goto notHeating;
    when True       sync startHeat  do {True}          goto warmingUp;
    when True       sync stopHeat   do {True}          goto warmingUp;

loc notHeating: while True wait {cf' == 1};
    when True       sync startHeat  do {True}  goto heating;
    when True       sync stopHeat  do {True}  goto notHeating;
    when True       sync powerOff do {cf'==0}  goto poweredOff;

loc heating: while True wait {cf' == 1};
    when True       sync startHeat  do {True}  goto heating;
    when True       sync stopHeat   do {True}          goto notHeating;
    when True       sync powerOff do {cf'==0}        goto poweredOff;


initially: notHeating & cf==0;
end

//--------------------------------------------------
automaton thermostat
//--------------------------------------------------

state_var: c; //c = clock variable of the thermostat
input_var: t1;
synclabs: startHeat, stopHeat, doNothing, tick;

loc idle: while c <= tau_sample wait {c' == 1};
    when c==tau_sample  sync tick           do {c'==0}  goto checking;
```

```
loc checking: while c <= 1 wait {c' == 1};
    when t1 <= (t_set - deltaL) sync startHeat do {c'==0} goto idle;
    when (t_set + deltaH) <= t1 sync stopHeat do {c'==0} goto idle;
    when (t_set - deltaL) <= t1 & t1 <= (t_set + deltaH) sync doNothing
     do {c'==0} goto idle;

initially: idle & c==0;
end

//----------------------------------------------------
automaton zone1
//----------------------------------------------------

state_var: t1;
//input_var: t2;
synclabs: startHeat, stopHeat, powerOff, lab1, specTick;

loc atAmbient: while t1 == 0 wait {t1'==0};
    when True          sync stopHeat   do {t1'== t1}  goto atAmbient;
    when True          sync powerOff   do {t1'== t1}  goto atAmbient;
    when True          sync startHeat  do {t1'== t1}       goto h1;

loc h1: while 0 <= t1 & t1 < 40 wait {0.9 <= t1' & t1' <= 1.1};
    when True          sync stopHeat   do {t1'== t1}  goto nh1;
    when True          sync powerOff   do {t1'== t1}  goto nh1;
    when True          sync startHeat  do {t1'== t1}  goto h1;
    when t1 == v5      sync lab1       do {t1'== t1}  goto atHottest;

loc atHottest: while t1 == 40 wait {t1'==0};
    when True          sync stopHeat   do {t1'== t1}  goto nh1;
    when True          sync powerOff   do {t1'== t1}  goto nh1;
    when True          sync startHeat  do {t1'== t1}  goto atHottest;

loc nh1: while 0 < t1 & t1 <= 40 wait {1.2' <= -t1 & -t1' <= 1.3};
    when True          sync stopHeat   do {t1'== t1}  goto nh1;
    when True          sync powerOff   do {t1'== t1}  goto nh1;
    when t1 == v1      sync lab1       do {t1'== t1}  goto atAmbient;

initially: nh1 & t1 == 17;
end


//----------------------------------------------------
automaton ambient
//----------------------------------------------------
```

```
state_var: ta;
//synclabs: specTick;

loc always: while True  wait {ta == 0};

initially: always & ta==0;
end

//------------------------------------------------------
// Composition
//------------------------------------------------------

sys = furnace & thermostat & zone1 & ambient;

//------------------------------------------------------
automaton spec
//------------------------------------------------------

state_var: t1, cSpec;
synclabs: powerOn, powerOff, startHeat, stopHeat,
auto, doNothing, tick, lab1, lab2, specTick;

loc check:

while t_m <= t1 & t1 <= t_M wait {True};
    when True   sync powerOff       do {True}           goto dontCheck;

// self loops
    when True   sync powerOn    do {True}           goto check;
    when True   sync startHeat  do {True}           goto check;
    when True   sync stopHeat   do {True}           goto check;
    when True   sync auto       do {True}           goto check;
    when True   sync doNothing  do {True}           goto check;
    when True   sync tick       do {True}           goto check;
    when True   sync lab1       do {True}           goto check;
    when True   sync lab2       do {True}           goto check;

loc dontCheck: while True wait {True};
    when t_bottom <= t1 sync powerOn do {cSpec'== 0} goto waiting;

// self loops
    when True   sync powerOff   do {True}           goto dontCheck;
    when True   sync startHeat  do {True}           goto dontCheck;
    when True   sync stopHeat   do {True}           goto dontCheck;
    when True   sync auto       do {True}           goto dontCheck;
    when True   sync doNothing  do {True}           goto dontCheck;
```

```
    when True   sync tick       do {True}           goto dontCheck;
    when True   sync lab1       do {True}           goto dontCheck;
    when True   sync lab2       do {True}           goto dontCheck;

loc waiting: while t_bottom <= t1 & cSpec <= tau_spec wait {cSpec' == 1};
    when cSpec==tau_spec  sync specTick     do {True}    goto check;
    when True             sync powerOff     do {True}         goto dontCheck;

// self loops
    when True   sync powerOn    do {True}           goto waiting;
    when True   sync startHeat  do {True}           goto waiting;
    when True   sync stopHeat   do {True}           goto waiting;
    when True   sync auto       do {True}           goto waiting;
    when True   sync doNothing  do {True}           goto waiting;
    when True   sync tick       do {True}           goto waiting;
    when True   sync lab1       do {True}           goto waiting;
    when True   sync lab2       do {True}           goto waiting;

initially: check & t_m <= t1 & t1 <= t_M;
end

//-----------------------------------------------------
// Simulation relation checking
//-----------------------------------------------------

SIM_PRIME_WITH_REACH  = false;
is_sim(sys,spec);
R = get_sim(sys,spec);
R.print;

reach = sys.reachable;
reach.print("test_emsoft",0);
```

(End of experiment 1)

Experiment 2 In this experiment, we model a user who turns the furnace on or off nondeterministically. We assume that the user can't just turn the furnace on or off arbitrarily fast. Therefore, we add some delay in the user actions.

```
//-----------------------------------------------------
//     Constants
//-----------------------------------------------------

tau_sample := 2;       // sampling time of the controller
tau_spec := 4; // waiting time in the specification
```

```
tau_warmup  := 3;

deltaH   := 2;       // delta above t_set for the thermostat to ignore
deltaL      := 2;       // delta below t_set fot the thermostat to ignore

t_set := 30;      // setpoint for the thermostat

t_m         := 25;      // lower bound on the temperature spec
t_M         := 35;      // upper bound on the temperature spec

t_bottom    := 10;

tf_h  := 40; // Hottest furnace temp.
tf_l  := 38; // Hottest furnace temp.

ta_l := 0;
ta_h := 2;

ta_c        := 0;

sqw := (tf_h - ta_l)/4;

v1 := ta_l; // 0
v2 := v1 + sqw;
v3 := v2 + sqw; // 20
v4 := v3 + sqw;
v5 := v4 + sqw; //40

//----------------------------------------------------
automaton furnace
//----------------------------------------------------
state_var : cf; //cf = clock of the furnace
synclabs: powerOn, powerOff, startHeat, stopHeat, auto;

loc off: while True wait {cf' == 1};
when True       sync powerOn  do {cf'==0}  goto warmingUp;
    when True       sync startHeat  do {cf'==cf}        goto off;
    when True       sync stopHeat   do {cf'==cf}        goto off;
when True       sync powerOff  do {cf'==cf}  goto off;

loc warmingUp: while cf <= tau_warmup wait {cf' == 1};
    when cf == tau_warmup   sync auto     do {cf'==cf}  goto notHeating;
    when True       sync startHeat  do {cf'==cf}        goto warmingUp;
    when True       sync stopHeat   do {cf'==cf}        goto warmingUp;
    when True       sync powerOn    do {cf'==cf}        goto warmingUp;
    when True       sync powerOff   do {cf'==cf}        goto off;
```

```
loc notHeating: while True wait {cf' == 1};
    when True       sync startHeat  do {cf'==cf}  goto heating;
    when True       sync stopHeat  do {cf'==cf}  goto notHeating;
    when True       sync powerOff do {cf'==cf}       goto off;
    when True       sync powerOn do {cf'==cf}       goto notHeating;

loc heating: while True wait {cf' == 1};
    when True       sync startHeat  do {cf'==cf}  goto heating;
    when True       sync stopHeat   do {cf'==cf}       goto notHeating;
    when True       sync powerOff do {cf'==cf}       goto off;
    when True       sync powerOn do {cf'==cf}       goto heating;

initially: heating & cf==0;
end

//------------------------------------------------------
automaton thermostat
//------------------------------------------------------

state_var: c; //c = clock variable of the thermostat
input_var: t1;
synclabs: startHeat, stopHeat, doNothing, tick;

loc idle: while c <= tau_sample wait {c'==1};
    when c==tau_sample  sync tick           do {c'==0}  goto checking;

loc checking: while c <= 1 wait {c'==1};
    when t1 <= (t_set - deltaL) sync startHeat do {c'==0} goto idle;
    when (t_set + deltaH) <= t1 sync stopHeat do {c'==0} goto idle;
    when (t_set - deltaL) <= t1 & t1 <= (t_set + deltaH) sync doNothing
do {c'==0} goto idle;

initially: idle & c==0;
end

//------------------------------------------------------
automaton zone1
//------------------------------------------------------

state_var: t1;
synclabs: startHeat, stopHeat, powerOff, lab1, specTick;

loc atAmbient: while t1 == 0 wait {t1'==0};
    when True           sync stopHeat   do {t1'== t1}  goto atAmbient;
    when True           sync powerOff   do {t1'== t1}  goto atAmbient;
```

```
    when True           sync startHeat  do {t1'== t1}       goto h1;

loc h1: while 0 <= t1 & t1 < 40 wait {t1' == 0.5};
    when True           sync stopHeat   do {t1'== t1}  goto nh1;
    when True           sync powerOff   do {t1'== t1}  goto nh1;
    when True           sync startHeat  do {t1'== t1}  goto h1;
    when t1 == v5       sync lab1       do {t1'== t1}  goto atHottest;

loc atHottest: while t1 == 40 wait {t1'==0};
    when True           sync stopHeat   do {t1'== t1}  goto nh1;
    when True           sync powerOff   do {t1'== t1}  goto nh1;
    when True           sync startHeat  do {t1'== t1}  goto atHottest;

loc nh1: while 0 < t1 & t1 <= 40 wait {t1' == -0.3};
    when True           sync stopHeat   do {t1'== t1}  goto nh1;
    when True           sync powerOff   do {t1'== t1}  goto nh1;
    when t1 == v1       sync lab1       do {t1'== t1}  goto atAmbient;

initially: h1 & t1 == 30;
end

//------------------------------------------------------
automaton user
//------------------------------------------------------
state_var: cu;
synclabs: powerOn, powerOff;

loc pause: while True wait {cu'==1};
    when cu>=0.1    sync powerOn    do{cu'==0}    goto pause;
    when cu>=0.1    sync powerOff   do{cu'==0}    goto pause;

initially: pause & cu==0;
end

//------------------------------------------------------
// Composition
//------------------------------------------------------

sys = furnace & thermostat & zone1 & user;


//------------------------------------------------------
automaton spec
//------------------------------------------------------

state_var: t1, cSpec;
```

```
synclabs: powerOn, powerOff, startHeat, stopHeat,
auto, doNothing, tick, lab1, lab2, specTick;

loc check:

while t_m <= t1 & t1 <= t_M wait {cSpec'== 1};
    when True   sync powerOff       do {True}           goto dontCheck;

// self loops
   // when True   sync powerOn    do {True}           goto check;
    when True   sync startHeat  do {True}           goto check;
    when True   sync stopHeat   do {True}           goto check;
    when True   sync auto       do {True}           goto check;
    when True   sync doNothing  do {True}           goto check;
    when True   sync tick       do {True}           goto check;
    when True   sync lab1       do {True}           goto check;
    when True   sync lab2       do {True}           goto check;

loc dontCheck: while True wait {cSpec'== 1};
    when t_bottom <= t1   sync powerOn        do {cSpec'== 0}          goto waiting;
    when t1 <= t_bottom   sync powerOn        do {cSpec'== 0}          goto dontCheck;

// self loops
   // when True   sync powerOff   do {True}           goto dontCheck;
    when True   sync startHeat  do {True}           goto dontCheck;
    when True   sync stopHeat   do {True}           goto dontCheck;
    when True   sync auto       do {True}           goto dontCheck;
    when True   sync doNothing  do {True}           goto dontCheck;
    when True   sync tick       do {True}           goto dontCheck;
   // when True   sync lab1       do {True}           goto dontCheck;
    when True   sync lab2       do {True}           goto dontCheck;

loc waiting: while t_bottom <= t1 & cSpec <= tau_spec wait {cSpec'== 1};
    when cSpec==tau_spec  sync specTick     do {True}    goto check;
    when True             sync powerOff     do {True}         goto dontCheck;

// self loops
  //  when True   sync powerOn    do {True}           goto waiting;
    when True   sync startHeat  do {True}           goto waiting;
    when True   sync stopHeat   do {True}           goto waiting;
    when True   sync auto       do {True}           goto waiting;
    when True   sync doNothing  do {True}           goto waiting;
    when True   sync tick       do {True}           goto waiting;
    when True   sync lab1       do {True}           goto waiting;
    when True   sync lab2       do {True}           goto waiting;
```

```
initially: check & t_m <= t1 & t1 <= t_M;
end

//--------------------------------------------------
// Simulation relation checking
//--------------------------------------------------

SIM_PRIME_WITH_REACH  = false;
//is_sim(sys,spec);
R = get_sim(sys,spec);
R.print;

reach = sys.reachable;
reach.print("test_emsoft",0);
```

(End of experiment 2)

Experiment 3 In this experiment, we include two zones in the system, but model the dynamics of the zones coarsely.

```
//--------------------------------------------------
//     Constants
//--------------------------------------------------

tau_sample := 2;       // sampling time of the controller
tau_spec := 5; // waiting time in the specification
tau_warmup  := 4;

deltaH   := 2;       // delta above t_set for the thermostat to ignore
deltaL      := 2;       // delta below t_set fot the thermostat to ignore

rc_l        := 0.2;     // lower bound on the rate of cooling
rc_h        := 0.4;     // upper bound on the rate of cooling

rh_l        := 0.5;     // lower bound on the rate of heating
rh_h        := 0.7;     // upper bound on the rate of heating

t_set := 17;      // setpoint for the thermostat

t_0l  := 16;      // lower bound for the confidence interval of t_0
t_0h := 18;      // upper bound for the confidence interval of t_0

t_m         := 15;      // lower bound on the temperature spec
t_M         := 25;      // upper bound on the temperature spec
```

```
t_bottom    := 10;

tf_h  := 40; // Hottest furnace temp.
tf_l  := 38; // Hottest furnace temp.

ta_l := 0;
ta_h := 2;

sqw := (tf_h - ta_l)/4;

v1 := ta_l;
v2 := v1 + sqw;
v3 := v2 + sqw;
v4 := v3 + sqw;
v5 := v4 + sqw;

//------------------------------------------------------
automaton furnace
//------------------------------------------------------
state_var : tf,cf; //tf = temperature of the furnace,
// cf = clock of the furnace
synclabs: powerOn, powerOff, startHeat, stopHeat, heatOn, heatOff, auto;

loc poweredOff: while True wait {True};
when True sync powerOn  do {cf'==0} goto warmingUp;
loc warmingUp: while cf <= tau_warmup wait {1 <= cf & cf <= 1};
    when cf == tau_warmup sync auto     do {True}  goto notHeating;
loc notHeating: while True wait {True};
    when True sync startHeat  do {True}  goto startingHeat;
    when True sync powerOff do {True}  goto poweredOff;
loc heating: while True wait {tf' == 0};
    when True sync stopHeat       do {True}  goto stoppingHeat;
    when True sync powerOff do {True}  goto poweredOff;
loc startingHeat: while True wait {True};
    when True sync heatOn  do {tf' == (tf_l+tf_h)/2}  goto heating;
loc stoppingHeat: while True wait {True};
    when True sync heatOff  do {True}  goto notHeating;

initially: notHeating & tf_l<=tf & tf<=tf_h;
end

//------------------------------------------------------
automaton thermostat
//------------------------------------------------------

state_var: c; //c = clock variable of the thermostat
```

```
input_var: t1;
synclabs: startHeat, stopHeat, doNothing, tick;

loc idle: while c <= tau_sample wait {c' == 1};
    when c==tau_sample sync tick do {c'==0} goto checking;
loc checking: while c <= 1 wait {c' == 1};
    when t1 <= (t_set - deltaL) sync startHeat do {c'==0} goto idle;
    when (t_set + deltaH) <= t1 sync stopHeat do {c'==0} goto idle;
    when (t_set - deltaL) <= t1 & t1 <= (t_set + deltaH)  sync doNothing
     do {c'==0} goto idle;

initially: idle & c==0;
end

//----------------------------------------------------
automaton zone1
//----------------------------------------------------

state_var: t1;
input_var: t2, ta, tf;
synclabs: heatOn, heatOff, lab1;

loc h1: while v1 <= t1 & t1 <= v3 & v1 <= t2 & t2 <= v3
 wait {6.9 <= t1' & t1' <= 7.1};
    when True sync heatOff        do {t1'== t1}  goto nh1;
    when t1 == v3 sync lab1       do {t1'== t1}  goto h2;
    when t1 == v1 sync lab1       do {t1'== v2}  goto h1;
    when t2 == v1 sync lab1       do {t1'== t1}  goto h1;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h1;
loc h2: while v3 <= t1 & t1 <= v5 & v1 <= t2 & t2 <= v3
 wait {2.9 <= t1' & t1' <= 3.1};
    when True sync heatOff        do {t1'== t1}  goto nh2;
    when t2 == v1 sync lab1       do {t1'== t1}  goto h2;
    when t1 == v3 sync lab1       do {t1'== t1}  goto h1;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h3;
    when t1 == v5 sync lab1       do {t1'== v4}  goto h2;
loc h3: while v3 <= t1 & t1 <= v5 & v3 <= t2 & t2 <= v5
 wait {2.9 <= t1' & t1' <= 3.1};
    when True sync heatOff        do {t1'== t1}  goto nh3;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h2;
    when t2 == v5 sync lab1       do {t1'== t1}  goto h3;
    when t1 == v3 sync lab1       do {t1'== v4}  goto h3;
    when t1 == v5 sync lab1       do {t1'== v4}  goto h3;

loc nh1: while v1 <= t1 & t1 <= v3 & v1 <= t2 & t2 <= v3
 wait {0.025 <= -t1' & -t1' <= 0.035};
```

```
    when True sync heatOn        do {t1'== t1}  goto h1;
    when t1 == v3 sync lab1       do {t1'== t1}           goto nh2;
    when t1 == v1 sync lab1       do {t1'== v2}  goto nh1;
    when t2 == v1 sync lab1       do {t1'== t1}  goto nh1;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh1;
loc nh2: while v3 <= t1 & t1 <= v5 & v1 <= t2 & t2 <= v3
 wait {0.45 <= -t1' & -t1' <= 0.55};
    when True sync heatOn        do {t1'== t1}  goto h2;
    when t2 == v1 sync lab1       do {t1'== t1}  goto nh2;
    when t1 == v3 sync lab1       do {t1'== t1}  goto nh1;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh3;
    when t1 == v5 sync lab1       do {t1'== v4}  goto nh2;
loc nh3: while v3 <= t1 & t1 <= v5 & v3 <= t2 & t2 <= v5
 wait {0.65 <= -t1' & -t1' <= 0.75};
    when True sync heatOn        do {t1'== t1}  goto h3;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh2;
    when t2 == v5 sync lab1       do {t1'== t1}  goto nh3;
    when t1 == v3 sync lab1       do {t1'== v4}  goto nh3;
    when t1 == v5 sync lab1       do {t1'== v4}  goto nh3;

initially: nh2 & v3 < t1 & t1 < v4 & v2 < t2 & t2 < v3;
end

//-----------------------------------------------------
automaton zone2
//-----------------------------------------------------

state_var: t2;
input_var: t1, ta;
synclabs: heatOn, heatOff, lab2;

loc h1: while v1 <= t1 & t1 <= v2 & v1 <= t2 & t2 <= v2
 wait {0.025 <= -t2' & -t2' <= 0.035};
    //when True sync heatOff        do {t1'== t1}  goto nh1;
    when t1 == v3 sync lab2       do {t1'== t1}  goto h2;
    when t1 == v1 sync lab2       do {t1'== v2}  goto h1;
    when t2 == v1 sync lab2       do {t1'== t1}  goto h1;
    when t2 == v3 sync lab2       do {t1'== t1}  goto h1;
loc h2: while v2 <= t1 & t1 <= v3 & v1 <= t2 & t2 <= v2
 wait {0.45 <= t2' & t2' <= 0.55};
    //when True sync heatOff        do {t1'== t1}  goto nh2;
    when t2 == v1 sync lab2       do {t1'== t1}  goto h2;
    when t1 == v3 sync lab2       do {t1'== t1}  goto h1;
    when t2 == v3 sync lab2       do {t1'== t1}  goto h3;
    when t1 == v5 sync lab2       do {t1'== v4}  goto h2;
loc h3: while v3 <= t1 & t1 <= v4 & v1 <= t2 & t2 <= v2
```

```
 wait {0.65 <= t2' & t2' <= 0.75};
    //when True sync heatOff        do {t1'== t1}  goto nh3;
    when t2 == v3 sync lab2       do {t1'== t1}  goto h2;
    when t2 == v5 sync lab2       do {t1'== t1}  goto h3;
    when t1 == v3 sync lab2       do {t1'== v4}  goto h3;
    when t1 == v5 sync lab2       do {t1'== v4}  goto h3;

initially: h2 & v3 < t1 & t1 < v4 & v2 < t2 & t2 < v3;
end

//------------------------------------------------------
automaton ambient
//------------------------------------------------------

state_var: ta;
synclabs: heatOn, heatOff, powerOff, error;

loc always: while ta_l <= ta & ta <= ta_h  wait {ta' <= 0};

initially: always & ta_l <= ta & ta <= ta_h;
end

//------------------------------------------------------
// Composition
//------------------------------------------------------

sys = furnace & thermostat & zone1 & zone2 & ambient;


//------------------------------------------------------
automaton spec
//------------------------------------------------------

state_var: t1, cSpec;
synclabs: powerOn, powerOff,specTick;

loc check:

while t_m <= t1 & t1 <= t_M wait {True};
    when True sync powerOff do {True} goto dontCheck;

loc dontCheck: while t_bottom <= t1 wait {True};
    when True sync powerOn do {cSpec'== 0} goto waiting;

loc waiting: while t_bottom <= t1 & cSpec <= tau_spec
wait {1 <= cSpec & cSpec <= 1};
```

```
    when cSpec==tau_spec sync specTick do {True} goto checking;


initially: check & t_m <= t1 & t1 <= t_M;
end

//-----------------------------------------------------
// Simulation relation checking
//-----------------------------------------------------

//SIM_PRIME_WITH_REACH  = false;
is_sim(sys,spec);
R = get_sim(sys,spec);
R.print;

(End of experiment 3)\\
```

Experiment 4 In this experiment, we model the dynamics of both the zones fairly in depth. This is a 4×4 breakdown of the temperature state space, similar to the 6×6 shown in figure 7.

```
//-----------------------------------------------------
//     Constants
//-----------------------------------------------------

tau_sample := 2;       // sampling time of the controller
tau_spec := 5; // waiting time in the specification
tau_warmup  := 4;

deltaH := 5;       // delta above t_set for the thermostat to ignore
deltaL := 5;       // delta below t_set fot the thermostat to ignore

rc_l   := 0.2;     // lower bound on the rate of cooling
rc_h   := 0.4;     // upper bound on the rate of cooling

rh_l   := 0.5;     // lower bound on the rate of heating
rh_h   := 0.7;     // upper bound on the rate of heating

t_set  := 30;      // setpoint for the thermostat

t_0l   := 26;      // lower bound for the confidence interval of t_0
t_0h  := 28;      // upper bound for the confidence interval of t_0


t_hottest   := 50;    // hottest the room can get due to the limited
```

```
//power of the furnace, has to be > t_amb
t_amb       := 0;       // ambient temperature
//(has to be lower than t_hottest

t_m         := 20;      // lower bound on the temperature spec
t_M         := 40;      // upper bound on the temperature spec

t_f  := 90; // Hottest furnace temp.

tf_h  := 40; // Hottest furnace temp.
tf_l  := 38; // Hottest furnace temp.

ta_l := 0;
ta_h := 2;

sqw := (tf_h - ta_l)/4;

v1 := ta_l;
v2 := v1 + sqw;
v3 := v2 + sqw;
v4 := v3 + sqw;
v5 := v4 + sqw;

//----------------------------------------------------
automaton furnace
//----------------------------------------------------
state_var : tf,cf; //tf = temperature of the furnace,
//cf = clock of the furnace
synclabs: powerOn, powerOff, startHeat, stopHeat, heatOn, heatOff, auto;

loc poweredOff: while True wait {True};
when True sync powerOn do {cf'==0} goto warmingUp;
loc warmingUp: while cf <= tau_warmup wait {1 <= cf & cf <= 1};
    when cf == tau_warmup sync auto do {True} goto notHeating;
loc notHeating: while True wait {True};
    when True sync startHeat  do {True} goto startingHeat;
    when True  sync powerOff do {True} goto poweredOff;
loc heating: while True wait {tf' == 0};
    when True  sync stopHeat do {True} goto stoppingHeat;
    when True sync powerOff do {True} goto poweredOff;
loc startingHeat: while True wait {True};
    when True sync heatOn do {tf' == (tf_l+tf_h)/2} goto heating;
loc stoppingHeat: while True wait {True};
    when True sync heatOff do {True} goto notHeating;

initially: notHeating & tf_l<=tf & tf<=tf_h;
```

```
end

//--------------------------------------------------
automaton thermostat
//--------------------------------------------------

state_var: c; //c = clock variable of the thermostat
input_var: t1;
synclabs: startHeat, stopHeat, doNothing, tick;
loc idle: while c <= tau_sample wait {c' == 1};
    when c==tau_sample  sync tick do {c'==0} goto checking;
loc checking: while c <= 1 wait {c' == 1};
    when t1 <= (t_set - deltaL) sync startHeat do {c'==0} goto idle;
    when (t_set + deltaH) <= t1 sync stopHeat do {c'==0} goto idle;
    when (t_set - deltaL) <= t1 & t1 <= (t_set + deltaH) sync doNothing
     do {c'==0} goto idle;
initially: idle & c==0;
end

//--------------------------------------------------
automaton zone1
//--------------------------------------------------

state_var: t1;
input_var: t2, ta, tf;
synclabs: heatOn, heatOff, lab1;

loc h1: while v1 <= t1 & t1 <= v2 & v1 <= t2 & t2 <= v2
 wait {7.9 <= t1' & t1' <= 8.1};
    when True sync heatOff       do {t1'== t1}  goto nh1;
    when t1 == v2 sync lab1      do {t1'== t1}  goto h2;
    when t1 == v1 sync lab1      do {t1'== (v1+v2)/2}  goto h1;
    when t2 == v1 sync lab1      do {t1'== t1}  goto h1;
    when t2 == v2 sync lab1      do {t1'== t1}  goto h1;
loc h2: while v2 <= t1 & t1 <= v3 & v1 <= t2 & t2 <= v2
 wait {5.9 <= t1' & t1' <= 6.1};
    when True sync heatOff       do {t1'== t1}  goto nh2;
    when t2 == v1 sync lab1      do {t1'== t1}  goto h2;
    when t1 == v3 sync lab1      do {t1'== t1}  goto h3;
    when t2 == v2 sync lab1      do {t1'== t1}  goto h5;
    when t1 == v2 sync lab1      do {t1'== t1}  goto h1;
loc h3: while v3 <= t1 & t1 <= v4 & v1 <= t2 & t2 <= v2
 wait {3.9 <= t1' & t1' <= 4.1};
    when True sync heatOff       do {t1'== t1}  goto nh3;
    when t1 == v3 sync lab1      do {t1'== t1}  goto h2;
    when t2 == v1 sync lab1      do {t1'== t1}  goto h3;
```

```
    when t1 == v4 sync lab1       do {t1'== t1}  goto h4;
    when t2 == v2 sync lab1       do {t1'== t1}  goto h6;
loc h4: while v4 <= t1 & t1 <= v5 & v1 <= t2 & t2 <= v2
 wait {1.9 <= t1' & t1' <= 2.1};
    when True sync heatOff        do {t1'== t1}  goto nh4;
    when t1 == v4 sync lab1       do {t1'== t1}  goto h3;
    when t2 == v1 sync lab1       do {t1'== t1}  goto h4;
    when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto h4;
    when t2 == v2 sync lab1       do {t1'== t1}  goto h7;
loc h5: while v2 <= t1 & t1 <= v3 & v2 <= t2 & t2 <= v3
 wait {5.9 <= t1' & t1' <= 6.1};
    when True sync heatOff        do {t1'== t1}  goto nh5;
    when t2 == v2 sync lab1       do {t1'== t1}  goto h2;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h5;
    when t1 == v2 sync lab1       do {t1'== (v2+v3)/2}  goto h5;
    when t1 == v3 sync lab1       do {t1'== t1}  goto h6;
loc h6: while v3 <= t1 & t1 <= v4 & v2 <= t2 & t2 <= v3
 wait {3.9 <= t1' & t1' <= 4.1};
    when True sync heatOff        do {t1'== t1}  goto nh6;
    when t2 == v2 sync lab1       do {t1'== t1}  goto h3;
    when t1 == v3 sync lab1       do {t1'== t1}  goto h5;
    when t1 == v4 sync lab1       do {t1'== t1}  goto h7;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h8;
loc h7: while v4 <= t1 & t1 <= v5 & v2 <= t2 & t2 <= v3
 wait {1.9 <= t1' & t1' <= 2.1};
    when True sync heatOff        do {t1'== t1}  goto nh7;
    when t2 == v2 sync lab1       do {t1'== t1}  goto h4;
    when t1 == v4 sync lab1       do {t1'== t1}  goto h6;
    when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto h7;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h9;
loc h8: while v3 <= t1 & t1 <= v4 & v3 <= t2 & t2 <= v4
 wait {3.9 <= t1' & t1' <= 4.1};
    when True sync heatOff        do {t1'== t1}  goto nh8;
    when t1 == v3  sync lab1       do {t1'== (v3+v4)/2}  goto h8;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h6;
    when t2 == v4 sync lab1       do {t1'== t1}  goto h8;
    when t1 == v4 sync lab1       do {t1'== t1}  goto h9;
loc h9: while v4 <= t1 & t1 <= v5 & v3 <= t2 & t2 <= v4
 wait {1.9 <= t1' & t1' <= 2.1};
    when True sync heatOff        do {t1'== t1}  goto nh9;
    when t2 == v3 sync lab1       do {t1'== t1}  goto h7;
    when t1 == v4 sync lab1       do {t1'== t1}  goto h8;
    when t2 == v4 sync lab1       do {t1'== t1}  goto h10;
    when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto h9;
loc h10: while v4 <= t1 & t1 <= v5 & v4 <= t2 & t2 <= v5
 wait {1.9 <= t1' & t1' <= 2.1};
```

```
   when True sync heatOff        do {t1'== t1}  goto nh10;
   when t2 == v5 sync lab1       do {t1'== t1}  goto h10;
   when t2 == v4 sync lab1       do {t1'== t1}  goto h9;
   when t1 == v4 sync lab1       do {t1'== (v4+v5)/2}  goto h10;
   when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto h10;

loc nh1: while v1 <= t1 & t1 <= v2 & v1 <= t2 & t2 <= v2
 wait {0.025 <= -t1' & -t1' <= 0.035};
   when True sync heatOn        do {t1'== t1}  goto h1;
   when t1 == v2 sync lab1       do {t1'== t1}  goto nh2;
   when t1 == v1 sync lab1       do {t1'== (v1+v2)/2}  goto nh1;
   when t2 == v1 sync lab1       do {t1'== t1}  goto nh1;
   when t2 == v2 sync lab1       do {t1'== t1}  goto nh1;
loc nh2: while v2 <= t1 & t1 <= v3 & v1 <= t2 & t2 <= v2
 wait {0.45 <= -t1' & -t1' <= 0.55};
   when True  sync heatOn        do {t1'== t1}  goto h2;
   when t2 == v1 sync lab1       do {t1'== t1}  goto nh2;
   when t1 == v3 sync lab1       do {t1'== t1}  goto nh3;
   when t2 == v2 sync lab1       do {t1'== t1}  goto nh5;
   when t1 == v2 sync lab1       do {t1'== t1}  goto nh1;
loc nh3: while v3 <= t1 & t1 <= v4 & v1 <= t2 & t2 <= v2
 wait {0.65 <= -t1' & -t1' <= 0.75};
   when True sync heatOn        do {t1'== t1}  goto h3;
   when t1 == v3 sync lab1       do {t1'== t1}  goto nh2;
   when t2 == v1 sync lab1       do {t1'== t1}  goto nh3;
   when t1 == v4 sync lab1       do {t1'== t1}  goto nh4;
   when t2 == v2 sync lab1       do {t1'== t1}  goto nh6;
loc nh4: while v4 <= t1 & t1 <= v5 & v1 <= t2 & t2 <= v2
 wait {0.95 <= -t1' & -t1' <= 1.05};
   when True sync heatOn        do {t1'== t1}  goto h4;
   when t1 == v4 sync lab1       do {t1'== t1}  goto nh3;
   when t2 == v1 sync lab1       do {t1'== t1}  goto nh4;
   when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto nh4;
   when t2 == v2 sync lab1       do {t1'== t1}  goto nh7;
loc nh5: while v2 <= t1 & t1 <= v3 & v2 <= t2 & t2 <= v3
 wait {0.025 <= -t1' & -t1 '<= 0.035};
   when True sync heatOn        do {t1'== t1}  goto h5;
   when t2 == v2  sync lab1       do {t1'== t1}  goto nh2;
   when t2 == v3  sync lab1       do {t1'== t1}  goto nh5;
   when t1 == v2  sync lab1       do {t1'== (v2+v3)/2}  goto nh5;
   when t1 == v3  sync lab1       do {t1'== t1}  goto nh6;
loc nh6: while v3 <= t1 & t1 <= v4 & v2 <= t2 & t2 <= v3
 wait {0.45 <= -t1' & -t1' <= 0.55};
   when True sync heatOn        do {t1'== t1}  goto h6;
   when t2 == v2 sync lab1       do {t1'== t1}  goto nh3;
   when t1 == v3 sync lab1       do {t1'== t1}  goto nh5;
```

```
    when t1 == v4 sync lab1       do {t1'== t1}  goto nh7;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh8;
loc nh7: while v4 <= t1 & t1 <= v5 & v2 <= t2 & t2 <= v3
 wait {0.65 <= -t1' & -t1' <= 0.75};
    when True sync heatOn        do {t1'== t1}  goto h7;
    when t2 == v2 sync lab1       do {t1'== t1}  goto nh4;
    when t1 == v4 sync lab1       do {t1'== t1}  goto nh6;
    when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto nh7;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh9;
loc nh8: while v3 <= t1 & t1 <= v4 & v3 <= t2 & t2 <= v4
 wait {0.025 <= -t1' & -t1' <= 0.035};
    when True sync heatOn        do {t1'== t1}  goto h8;
    when t1 == v3 sync lab1       do {t1'== (v3+v4)/2}  goto nh8;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh6;
    when t2 == v4 sync lab1       do {t1'== t1}  goto nh8;
    when t1 == v4 sync lab1       do {t1'== t1}  goto nh9;
loc nh9: while v4 <= t1 & t1 <= v5 & v3 <= t2 & t2 <= v4
 wait {0.45 <= -t1' & -t1' <= 0.55};
    when True  sync heatOn        do {t1'== t1}  goto h9;
    when t2 == v3 sync lab1       do {t1'== t1}  goto nh7;
    when t1 == v4 sync lab1       do {t1'== t1}  goto nh8;
    when t2 == v4 sync lab1       do {t1'== t1}  goto nh10;
    when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto nh9;
loc nh10: while v4 <= t1 & t1 <= v5 & v4 <= t2 & t2 <= v5
 wait {0.025 <= -t1' & -t1' <= 0.035};
    when True sync heatOn        do {t1'== t1}  goto h10;
    when t2 == v5 sync lab1       do {t1'== t1}  goto nh10;
    when t2 == v4 sync lab1       do {t1'== t1}  goto nh9;
    when t1 == v4 sync lab1       do {t1'== (v4+v5)/2}  goto nh10;
    when t1 == v5 sync lab1       do {t1'== (v4+v5)/2}  goto nh10;


initially: nh6 & v3 < t1 & t1 < v4 & v2 < t2 & t2 < v3;
end

//---------------------------------------------------
automaton zone2
//---------------------------------------------------

state_var: t2;
input_var: t1, ta;
synclabs: heatOn, heatOff, lab2;

loc h1: while v1 <= t1 & t1 <= v2 & v1 <= t2 & t2 <= v2
 wait {0.025 <= -t2' & -t2' <= 0.035};
    //when True sync heatOff        do {t2'== t2}  goto nh1;
```

```
    when t1 == v2 sync lab2       do {t2'== t2}  goto h2;
    when t1 == v1 sync lab2       do {t2'== t2}  goto h1;
    when t2 == v1 sync lab2       do {t2'== (v1+v2)/2}  goto h1;
    when t2 == v2 sync lab2       do {t2'== (v1+v2)/2}  goto h1;
loc h2: while v2 <= t1 & t1 <= v3 & v1 <= t2 & t2 <= v2
 wait {0.45 <= t2' & t2' <= 0.55};
    //when True sync heatOff        do {t2'== t2}  goto nh2;
    when t2 == v1 sync lab2       do {t2'== (v1+v2)/2}  goto h2;
    when t1 == v3 sync lab2       do {t2'== t2}  goto h3;
    when t2 == v2 sync lab2       do {t2'== t2}  goto h5;
    when t1 == v2 sync lab2       do {t2'== t2}  goto h1;
loc h3: while v3 <= t1 & t1 <= v4 & v1 <= t2 & t2 <= v2
 wait {0.65 <= t2' & t2' <= 0.75};
    //when True sync heatOff        do {t2'== t2}  goto nh3;
    when t1 == v3 sync lab2       do {t2'== t2}  goto h2;
    when t2 == v1 sync lab2       do {t2'== (v1+v2)/2}  goto h3;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h4;
    when t2 == v2 sync lab2       do {t2'== t2}  goto h6;
loc h4: while v4 <= t1 & t1 <= v5 & v1 <= t2 & t2 <= v2
 wait {0.95 <= t2' & t2' <= 1.05};
    //when True sync heatOff        do {t2'== t2}  goto nh4;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h3;
    when t2 == v1 sync lab2       do {t2'== (v1+v2)/2}  goto h4;
    when t1 == v5 sync lab2       do {t2'== t2}  goto h4;
    when t2 == v2 sync lab2       do {t2'== t2}  goto h7;
loc h5: while v2 <= t1 & t1 <= v3 & v2 <= t2 & t2 <= v
3 wait {0.025 <= -t2' & -t2' <= 0.035};
    //when True sync heatOff        do {t2'== t2}  goto nh5;
    when t2 == v2 sync lab2       do {t2'== t2}  goto h2;
    when t2 == v3 sync lab2       do {t2'== (v2+v3)/2}  goto h5;
    when t1 == v2 sync lab2       do {t2'== t2}  goto h5;
    when t1 == v3 sync lab2       do {t2'== t2}  goto h6;
loc h6: while v3 <= t1 & t1 <= v4 & v2 <= t2 & t2 <= v3
 wait {0.45 <= t2' & t2' <= 0.55};
    //when True sync heatOff        do {t2'== t2}  goto nh6;
    when t2 == v2 sync lab2       do {t2'== t2}  goto h3;
    when t1 == v3 sync lab2       do {t2'== t2}  goto h5;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h7;
    when t2 == v3 sync lab2       do {t2'== t2}  goto h8;
loc h7: while v4 <= t1 & t1 <= v5 & v2 <= t2 & t2 <= v3
 wait {0.65 <= t2' & t2' <= 0.75};
    //when True sync heatOff        do {t2'== t2}  goto nh7;
    when t2 == v2 sync lab2       do {t2'== t2}  goto h4;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h6;
    when t1 == v5 sync lab2       do {t2'== t2}  goto h7;
    when t2 == v3 sync lab2       do {t2'== t2}  goto h9;
```

```
loc h8: while v3 <= t1 & t1 <= v4 & v3 <= t2 & t2 <= v4
 wait {0.025 <= -t2' & -t2' <= 0.035};
    //when True sync heatOff        do {t2'== t2}  goto nh8;
    when t1 == v3 sync lab2       do {t2'== t2}  goto h8;
    when t2 == v3 sync lab2       do {t2'== t2}  goto h6;
    when t2 == v4 sync lab2       do {t2'== (v3+v4)/2}  goto h8;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h9;
loc h9: while v4 <= t1 & t1 <= v5 & v3 <= t2 & t2 <= v4
 wait {0.45 <= t2' & t2' <= 0.55};
    //when True sync heatOff        do {t2'== t2}  goto nh9;
    when t2 == v3 sync lab2       do {t2'== t2}  goto h7;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h8;
    when t2 == v4 sync lab2       do {t2'== t2}  goto h10;
    when t1 == v5 sync lab2       do {t2'== t2}  goto h9;
loc h10: while v4 <= t1 & t1 <= v5 & v4 <= t2 & t2 <= v5
 wait {0.025 <= -t2' & -t2' <= 0.035};
    //when True sync heatOff        do {t2'== t2}  goto nh10;
    when t2 == v5 sync lab2       do {t2'== (v4+v5)/2}  goto h10;
    when t2 == v4 sync lab2       do {t2'== t2}  goto h9;
    when t1 == v4 sync lab2       do {t2'== t2}  goto h10;
    when t1 == v5 sync lab2       do {t2'== t2}  goto h10;
initially: h6 & v3 < t1 & t1 < v4 & v2 < t2 & t2 < v3;
end

//-----------------------------------------------------
automaton ambient
//-----------------------------------------------------

state_var: ta;
synclabs: heatOn, heatOff, powerOff, error;
loc always: while ta_l <= ta & ta <= ta_h
  wait {ta' == 0};
initially: always & ta_l <= ta & ta <= ta_h;
end

//-----------------------------------------------------
// Composition
//-----------------------------------------------------

sys = furnace & thermostat & zone1 & zone2 & ambient;


//-----------------------------------------------------
automaton spec
//-----------------------------------------------------
```

```
state_var: t1,cSpec;
synclabs: powerOn, powerOff;

loc check:

while t_m <= t1 & t1 <= t_M wait {1 <= cSpec & cSpec <= 1};
    when True   sync powerOff       do {True}           goto dontCheck;
loc dontCheck: while t_bottom <= t1 wait {1 <= cSpec & cSpec <= 1};
    when True   sync powerOn        do {cSpec'==0}          goto wait;
loc wait: while t_bottom <= t1 & c <= tau_spec wait {1 <= cSpec & cSpec <= 1};
    when c==tau_spec  sync tick     do {True}    goto checking;
initially: check & t_m <= t1 & t1 <= t_M;
end

//----------------------------------------------------
// Simulation relation checking
//----------------------------------------------------

SIM_PRIME_WITH_REACH  = false;
is_sim(sys,spec);
R = get_sim(sys,spec);
R.print;
```

(End of experiment 4)

References

[1] Goran Frehse. Compositional Verification of Hybrid Systems using Simulation Relations. PhD thesis, Radboud Universiteit Nijmegen, October 10, 2005.

[2] Goran Frehse. Compositional Verification of Hybrid Systems with Discrete Interaction using Simulation Relations. *In CACSD 2004: IEEE Conference on Computer Aided Control Systems Design, Taipei, Taiwan, September 1-4, 2004.*

[3] Goran Frehse, Zhi Han, Bruce Krogh. Assume-Guarantee Reasoning for Hybrid I/O-Automata by Over-Approximation of Continuous Interaction. *In CDC 2004: IEEE Conference on Decision and Control, Atlantis, Bahamas, December 14-17, 2004.*

[4] Goran Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. *International Journal on Software Tools for Technology Transfer (STTT) Volume 10, Number 3, June, 2008.* A preliminary version appread in Manfred Morari and Lothar Thiele, edi-

tors, Hybrid Systems: Computation and Control (HSCC'05), volume 3414 of Lecture Notes in Computer Science, pages 258-273, Springer-Verlag, 2005. (revised)

[5] André Platzer. Differential Dynamic Logic for Hybrid Systems. *Journal of Automated Reasoning, 41(2), pages 143-189, 2008.*

[6] André Platzer. Differential-algebraic Dynamic Logic for Differential-algebraic Programs. *Journal of Logic and Computation, 2008. Accepted for publication by Oxford University Press.*

[7] Wonhong Nam, P. Madhusudan, Rajeev Alur. Automatic Symbolic Compositional Verification by Learning Assumptions. *In Formal Methods in System Design, Vol. 32(3):207-234, 2008. SCI-E.*