

Deductive Verification of Hybrid Systems Using STeP ^{*}

Zohar Manna and Henny B. Sipma
manna | sipma@cs.stanford.edu

Computer Science Department, Stanford University
Stanford, CA 94305

Abstract. We investigate the feasibility of computer-aided deductive verification of hybrid systems. Hybrid systems are modeled by phase transition systems, in which activities specify the bounds on the derivatives of the continuous variables. We present a method for invariant generation based on static analysis of the phase transition system. The invariants produced can be used as auxiliary properties in the verification of temporal properties. We show that in some cases the invariants thus produced suffice to prove the main safety property.

1 Introduction

Deductive approaches to the verification of hybrid systems have been studied extensively. However this work has been mostly theoretical; few implementations exist to test the feasibility of these approaches on practical problems. Some exceptions are [26] and [6] where PVS is used to verify (part of) the steamboiler challenge problem [1].

On the other hand, algorithmic verification methods for hybrid systems, based on hybrid automata [2, 16], and implemented in the tool HyTech [18] have been successfully applied to many, relatively large practical examples, for example [20, 21]. However, HyTech is applicable only to rectangular hybrid automata, that is, systems with a finite control structure, in which the derivative of all continuous variables either is constant or lies in an interval bounded by constants. Although several ways have been identified to construct conservative rectangular approximations of systems that cannot be described by rectangular automata [17, 19], these steps may be informally justified and thus error prone. Although HyTech is able to do parametric analysis, due to the limitations of current polyhedra technology, it is usually restricted to systems with a few parameters unspecified; it expects fixed, explicit values for the rest of the parameters.

In general, algorithmic methods are preferable whenever they are applicable, because they are fully automatic. However, deductive methods are applicable

^{*} This research was supported in part by the National Science Foundation under grant CCR-95-27927, the Defense Advanced Research Projects Agency under NASA grant NAG2-892, ARO under grant DAAH04-95-1-0317, ARO under MURI grant DAAH04-96-1-0341, and by Army contract DABT63-96-C-0096 (DARPA).

to a larger class of systems, and, in general can handle systems with symbolic constants, parameterized systems, and nonlinear systems. The price of the generality of the deductive approach is the need for intermediate assertions and invariants and possibly interactive theorem proving.

In this paper we investigate the practical aspects of the deductive verification of hybrid systems by presenting a prototype implementation of a tool to assist in such verification. It is applicable to systems with infinite control structure and is not limited to rectangular hybrid systems. Our approach to the deductive verification of hybrid systems is based on the formalism of *phase transition systems*, introduced by Manna and Pnueli [22] as a model to describe hybrid systems. Phase transition systems are an extension of fair transition systems: *activities* are used to describe how continuous variables evolve over time, and a *progress condition* imposes constraints on the progress of time under various conditions.

Our tool is implemented as part of the STeP (Stanford Temporal Prover) verification system, an integrated toolset for verifying linear-time temporal properties of reactive systems. STeP’s deductive methods include *verification rules* and *verification diagrams*. In [22] it is shown that phase transition systems define an associated transition system that has the same set of behaviours. In the associated transition system activities are translated into regular transitions parameterized by their duration. This correspondence makes STeP’s deductive methods, originally developed for discrete systems, immediately applicable to hybrid systems.

STeP also provides tools for the automatic generation of invariants. STeP’s invariant generation methods for discrete systems are described in [10], while [11] presents a method applicable to real-time systems. Here we adapt the method for real-time systems and propose an additional method that takes advantage of some properties of activities. We show that for some systems the invariants thus generated are sufficient to prove the properties of interest.

2 Preliminaries

2.1 Computational Model: Transition Systems

As the underlying computational model for verification we use *transition systems* [23]. A transition system $\Phi = \langle V, \Theta, \mathcal{T} \rangle$ consists of

- V : A finite set of typed *system variables*. A *state* is a type-consistent interpretation of the system variables. The set of all states is called the *state space*, and is designated by Σ . We say that a state s is a p -state if s satisfies p , written $s \models p$.
- Θ : The *initial condition*, a satisfiable assertion characterizing the initial states.
- \mathcal{T} : A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a function

$$\tau : \Sigma \mapsto 2^\Sigma$$

mapping each state $s \in \Sigma$ into a (possibly empty) set of τ -successor states, $\tau(s) \subseteq \Sigma$. Each transition τ is defined by a *transition relation* $\rho_\tau(V, V')$, a first-order formula in which the unprimed variables refer to the values in

the current state s , and the primed variables refer to the values in the next state s' . Transitions may be parameterized, thus simulating an infinite set of similar transitions.

Computations A computation of a transition system $\Phi = \langle V, \Theta, \mathcal{T} \rangle$ is an infinite sequence of states $\sigma : s_0, s_1, s_2, \dots$, such that

- *Initiation*: s_0 is initial, that is, $s_0 \models \Theta$.
- *Consecution*: For each $j = 0, 1, \dots$, s_{j+1} is a τ -successor of s_j , that is, $s_{j+1} \in \tau(s_j)$ for some $\tau \in \mathcal{T}$.

A computation prefix is a finite sequence of states that satisfies Initiation and Consecution.

2.2 System Description: Phase Transition Systems

Transition systems are not a very convenient formalisms to describe hybrid systems, because of the discrete nature of the transitions: transitions update the value of the variables in a discrete manner, rather than let the values of variables vary continuously over time. Therefore we use *phase transition systems* [22] to describe hybrid systems. The phase transition system presented here extends the one presented in [22] with *differential inclusions*.

A phase transition system (PTS) $\Psi = \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$ has the same components as a transition system plus two additional components that allow us to describe how continuous variables evolve over time. First, a PTS imposes some additional constraints on V , Θ , and \mathcal{T} :

- V : The set of system variables is partitioned into a set \mathcal{D} of *discrete variables*, which can be of any type, a set \mathcal{C} of *clock variables*, and a set \mathcal{I} of *continuous variables* (also known as *integrators*). All variables in \mathcal{C} and \mathcal{I} must be of type real. We assume that the set of clock variables includes a variable T , called the masterclock. The masterclock records the progress of global time.
- Θ : The initial condition must satisfy $\Theta \rightarrow T = 0$.
- \mathcal{T} : The transitions in \mathcal{T} are considered discrete and are assumed to happen instantaneously; therefore we require that no transition modify the master clock, that is, for every transition $\tau \in \mathcal{T}$ we require:

$$\rho_\tau(V, V') \rightarrow T' = T .$$

The new components in a PTS are

- \mathcal{A} : A finite set of *activities*. Each activity $\alpha \in \mathcal{A}$ is described by an *activity relation*:

$$\rho_\alpha : p_\alpha \rightarrow \dot{\mathcal{I}}_d^\alpha = F^\alpha(V) \wedge \dot{\mathcal{I}}_b^\alpha \in [F_l^\alpha(V), F_u^\alpha(V)]$$

where p_α , called the *activation condition*, is a predicate over \mathcal{D} , and $\mathcal{I}_d^\alpha \cup \mathcal{I}_b^\alpha = \mathcal{I}$, the set of integrators. \mathcal{I}_d^α is the set of variables for which the derivative is fully specified in α , while the derivatives of the variables in \mathcal{I}_b^α are specified by differential inclusions.

Activity α is said to be *active* in state s if its activation condition p_α holds on s . The formula $\mathcal{I}_d^\alpha = F^\alpha(V)$ stands for

$$\dot{x}_i = F_i^\alpha(V), \quad \text{for } i = 1, \dots, m$$

and the formula $\mathcal{I}_b^\alpha \in [F_l^\alpha(V), F_u^\alpha(V)]$ stands for the *differential inclusion*

$$\dot{x}_i \in [F_{i,l}^\alpha(V), F_{i,u}^\alpha(V)], \quad \text{for } i = m + 1, \dots, n$$

where $\{x_1, \dots, x_n\} = \mathcal{I}$. The functions F_i^α specify how the continuous variables change over time while the system is in a p_α -state; the functions $F_{i,l}^\alpha(V), F_{i,u}^\alpha(V)$ are a lower and upper bound on the derivative of x_i . Each activity must specify an evolution constraint on each continuous variable. We assume that the integral of each F^α is well-defined and we require that F^α does not depend on variables specified by a differential inclusion.

Activities should be *time-invariant*, that is, F_i^α cannot explicitly refer to time elapsed since the start of the activity. This condition does not reduce the expressiveness, but may require the introduction of additional variables. For example, to specify that a variable x varies according to the square root of time in some activity, we cannot say $\dot{x} = \sqrt{t}$, but we have to say $\dot{x} = \sqrt{y}, \dot{y} = 1$. The reason for this condition is to make sure that the effects of two consecutive τ_α steps are the same as the effects of one single τ_α step with duration the sum of the two steps.

To ensure that the phase transition system is *time-deterministic* we require that the activities' activation conditions are mutually exclusive and exhaustive, that is $p_{\alpha_1} \rightarrow \neg p_{\alpha_2}$ for $\alpha_1 \neq \alpha_2$, and $\bigvee_{\alpha \in \mathcal{A}} p_\alpha$ must hold on the reachable states.

- Π : The *time-progress condition*. An assertion over V used to specify a global restriction over the progress of time.

Associated Transition System With each activity $\alpha \in \mathcal{A}$ we associate a parameterized transition $\tau[\Delta]$, which represents an infinite set of transitions, one for each possible interval duration Δ . We refer to these transitions as time-step transitions: these are the only transitions that can advance global time. If α has activity relation

$$p_\alpha \rightarrow \mathcal{I}_d^\alpha = F^\alpha(V) \wedge \mathcal{I}_b^\alpha \in [F_l^\alpha(V), F_u^\alpha(V)]$$

the transition relation of $\tau_\alpha[\Delta]$ is given by

$$\rho_{\tau_\alpha}[\Delta] : \left(\begin{array}{c} \Delta > 0 \wedge p_\alpha \wedge \mathcal{D}' = \mathcal{D} \wedge \mathcal{C}' = \mathcal{C} + \Delta \\ \wedge \\ \mathcal{I}_d^{\alpha'} = \mathcal{I}_d^\alpha + G^\alpha(\Delta) \\ \wedge \\ \mathcal{I}_b^\alpha + G_l^\alpha(\Delta) \leq \mathcal{I}_b^{\alpha'} \wedge \mathcal{I}_b^{\alpha'} \leq \mathcal{I}_b^\alpha + G_u^\alpha(\Delta) \\ \wedge \\ \forall E \in \Re \forall \delta \in (0, \Delta]. \left(\begin{array}{c} \mathcal{I}_b^\alpha + G_l^\alpha(\delta) \leq E \wedge E \leq \mathcal{I}_b^\alpha + G_u^\alpha(\delta) \\ \rightarrow \\ \Pi(\mathcal{D}, \mathcal{C} + \delta, \mathcal{I}_d^\alpha + G^\alpha(\delta), E) \end{array} \right) \end{array} \right)$$

where

$$G^\alpha(\delta) = \int_0^\delta F^\alpha dt, \quad G_l^\alpha(\delta) = \int_0^\delta F_l^\alpha dt, \quad G_u^\alpha(\delta) = \int_0^\delta F_u^\alpha dt$$

and $\Pi(\mathcal{D}, \mathcal{C}, \mathcal{I}_d^\alpha, \mathcal{I}_b^\alpha)$ is the progress condition.

In words, each time-step transition $\tau_\alpha[\Delta]$ is a transition that is enabled if it has a positive time duration Δ , its activity condition p_α holds, and the progress condition holds throughout the interval $(0, \Delta]$ for all values of the derivatives of the variables in \mathcal{I}_b^α . It is assumed that during this interval all continuous variables evolve according to the derivatives specified in the activity relation, all clocks increase uniformly with time, and all discrete variables stay the same. If the transition is taken, the values of the variables in the successor state(s) are constrained by the primed expressions in the transition relation.

The progress condition is similar to the **tcp** predicate introduced in [24]: it constrains the time that the system can reside in a particular configuration.

The phase transition system $\Psi = \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$ defines the associated transition system $\Phi = \langle V, \Theta, \mathcal{T}_H \rangle$, where

$$\mathcal{T}_H = \mathcal{T} \cup \mathcal{T}_A, \quad \text{where } \mathcal{T}_A = \{\tau_\alpha[\Delta] \mid \alpha \in \mathcal{A}, \Delta \in \mathbb{R}^+\}$$

Computations A computation of a PTS Ψ is an infinite sequence of states $\sigma : s_0, s_1, s_2, \dots$, such that

- σ is a computation of Φ , where Φ is the associated transition system defined by Ψ , and
- *Time Divergence*: the value of the masterclock T grows beyond any bound, that is, the sequence $s_0[T], s_1[T], \dots$ grows beyond any bound.

A hybrid system is called *non-Zeno* if every finite sequence of states that is a computation prefix of the associated transition system can be extended into a computation. In this paper we restrict ourselves to non-Zeno systems.

2.3 Verification of safety properties

A *safety property* is a property expressible by a formula of the form $\Box p$, for a past temporal formula p (see [23] for definitions of past formula and the semantics of \Box). This includes *invariances*, where p is an assertion. In this case the formula states that p should be true in every accessible state of the system.

Because of the possibility to associate a transition system with a phase transition system, many verification rules presented in [23] can be reused for the verification of phase transition systems. In this paper we will use the *invariance rule* INV, shown in Figure 1, to prove some properties of hybrid systems. These rules reduce the system validity of a temporal formula to the general validity of a set of first-order verification conditions. In the rules $\{\varphi\} \tau \{\psi\}$ stands for the formula

$$(\varphi(V) \wedge \rho_\tau(V, V')) \rightarrow \psi(V') \quad , \quad \forall \Delta . (\varphi(V) \wedge \rho_\tau[\Delta](V, V')) \rightarrow \psi(V')$$

for a regular and a parameterized transition τ , respectively.

For PTS Ψ and assertions φ, p ,
I1. $\varphi \rightarrow p$
I2. $\Theta \rightarrow \varphi$
I3. $\{\varphi\} \tau \{\varphi\}$ for each $\tau \in \mathcal{T}_H$
$\Psi \models \Box p$

Fig. 1. Invariance rule INV

We say that an assertion p is *inductive* for a hybrid system Ψ if $\Box p$ can be proved using rule INV with φ equal to p (that is, p holds initially and is preserved by every transition). If these verification conditions can be proved assuming a set of properties S , we say that p is *inductive relative to S* .

3 STeP

The Stanford Temporal Prover, STeP, is a tool for the deductive and algorithmic verification of reactive systems [8, 9, 11].

STeP implements *verification rules* and *verification diagrams* for deductive verification. A collection of decision procedures for built-in theories, including integers, reals, datatypes and equality is combined with propositional and first-order reasoning to simplify verification conditions, proving many of them automatically. For those that cannot be established automatically, an interactive Gentzen-style theorem prover is available. Features such as parameterization and transitions originating from activities introduce quantifiers in verification conditions. Fortunately, the required quantifier instantiations are often “obvious” in that they use instances that can be provided by the decision procedures themselves. Accordingly, an integration of first-order reasoning and decision procedures was developed that can automatically discharge many verification conditions that would otherwise require the use of the interactive prover [12].

To enable symbolic manipulation of first-order formulas in the theory of real closed fields, we are planning to integrate STeP with REDLOG [13], a package that forms a front-end to the computer algebra system REDUCE [14]. Some of the verification conditions generated by the case studies reported in this paper were proved automatically by a version of REDLOG made available on the web.

4 Generation of Invariants

STeP provides tools for automatic generation of invariants based on static analysis of transition systems for reactive systems [10] and real-time systems [11]. These invariants are invaluable as auxiliary properties in deductive verification. We will describe two techniques for automatically generating invariants for hybrid systems.

For a PTS $\Psi = \langle V, \Theta, \mathcal{T}_D, \mathcal{A}, \Pi \rangle$, and associated transition system $\Phi = \langle V, \Theta, \mathcal{T}_H \rangle$, define

$$Post_D(X) = \bigvee_{\tau \in \mathcal{T}_D} post(\tau, X), \quad Post_A(X) = \bigvee_{\tau \in \mathcal{T}_H - \mathcal{T}_D} post(\tau, X)$$

where

$$post(\tau, X) = \exists V^0 . X(V^0) \wedge \rho_\tau(V^0, V)$$

and for a parameterized transition $\tau[\Delta]$

$$post(\tau, X) = \exists \Delta, V^0 . X(V^0) \wedge \rho_\tau[\Delta](V^0, V)$$

Thus, $Post_D(X)$ characterizes all the states that can be reached from a state satisfying X by a discrete transition, and $Post_A(X)$ characterizes all the states that can be reached from a state satisfying X by a time-step transition.

Invariant 1 The first invariant is similar to that described in [11] for real-time systems.

$$Inv_1 : \Theta \vee Post_D(true) \vee Post_A(true)$$

Inv_1 characterizes the set of states that is either an initial state or can be reached by either a discrete transition or a time-step transition, starting from anywhere in the state space. It is not hard to see that Inv_1 is an invariant of Ψ .

As we may want to apply an invariant to every verification condition, it is desirable to minimize the number of quantifiers it contains. In STeP the existential quantifiers generated for the discrete transitions are used with universal force when appearing in assumptions and are mostly eliminated by STeP's simplifier. Similar to [11] we can approximate $Post_A(true)$ by the progress condition Π , since

$$post(\tau_\alpha, true) = \exists \Delta, \mathcal{D}^0, \mathcal{C}^0, \mathcal{I}^0 . \rho_{\tau_\alpha}[\Delta](V^0, V)$$

is equivalent to

$$p_\alpha \wedge \exists \Delta \left(\begin{array}{c} \Delta > 0 \\ \wedge \\ \exists \mathcal{I}_b^{\alpha 0} \left(\begin{array}{c} \mathcal{I}_b^{\alpha 0} + G_t^\alpha(\Delta) \leq \mathcal{I}_b^\alpha \wedge \mathcal{I}_b^\alpha \leq \mathcal{I}_b^{\alpha 0} + G_u^\alpha(\Delta) \\ \wedge \\ \forall E, \delta \in (0, \Delta] \left(\begin{array}{c} \mathcal{I}_b^{\alpha 0} + G_t^\alpha(\delta) \leq E \wedge E \leq \mathcal{I}_b^{\alpha 0} + G_u^\alpha(\delta) \\ \rightarrow \\ \Pi(\mathcal{D}, \mathcal{C} + \delta - \Delta, \mathcal{I}_d^\alpha + G^\alpha(\delta) - G^\alpha(\Delta), E) \end{array} \right) \end{array} \right) \end{array} \right)$$

by taking $\mathcal{D}^0 = \mathcal{D}$, $\mathcal{C}^0 = \mathcal{C} - \Delta$ and $\mathcal{I}_d^{\alpha 0} = \mathcal{I}_d^\alpha - G^\alpha(\Delta)$, which in turn implies

$$p_\alpha \wedge \Pi(\mathcal{D}, \mathcal{C}, \mathcal{I}_d^\alpha, \mathcal{I}_b^\alpha)$$

by taking $\delta = \Delta$ and $E = \mathcal{I}_b^\alpha$, and thus we have

$$Post_A \rightarrow \Pi(\mathcal{D}, \mathcal{C}, \mathcal{I}_d^\alpha, \mathcal{I}_b^\alpha)$$

as required.

In [7] and [25] a similar method is used for the generation of invariants for untimed programs and hardware respectively.

Invariant 2 The second invariant takes advantage of the time-invariance property of activities. Time invariance ensures that the possible effects of taking two successive τ_α transitions of duration Δ_1 and Δ_2 are the same as taking one τ_α transition of duration $\Delta_1 + \Delta_2$, that is

$$\rho_{\tau_\alpha}[\Delta_1] \circ \rho_{\tau_\alpha}[\Delta_2] = \rho_{\tau_\alpha}[\Delta_1 + \Delta_2]$$

Based on this property we can claim

Claim 1 *Given a phase transition system $\Phi = \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$, the following is an invariant of Φ :*

$$Inv_2 : \Theta \vee Post_D(true) \vee Post_A(Post_D(true) \vee \Theta)$$

Justification: Assume, by contradiction, that there is some state s that is accessible in a computation of Ψ , but does not satisfy Inv_2 . Clearly s cannot be an initial state or the result of a discrete transition, so it must be the final state of a time-step transition, $\tau_\alpha[\Delta]$. Let s^0 be the state from which τ_α was taken. Clearly s^0 cannot be the result of a discrete transition, or an initial state, so it must, like s , be the final state of a time-step transition, τ_{α_1} . However, τ_{α_1} , where $\alpha_1 \neq \alpha$ cannot be followed immediately by τ_α , by the requirement that $p_{\alpha_1} \wedge p_\alpha$ be unsatisfiable, that the activation conditions only depend on discrete variables, and that a time-step transition cannot modify any discrete variables; two distinct time-step transitions always have to be separated by a discrete transition. Thus s^0 must be the final state of another time-step transition $\tau_\alpha[\Delta_1]$. However, by time invariance, the effect of $\tau_\alpha[\Delta_1]$ followed by $\tau_\alpha[\Delta]$ is the same as taking the single time-step transition $\tau_\alpha[\Delta_1 + \Delta]$; repeating the same argument for the starting state of $\tau_\alpha[\Delta_1]$ we can conclude, by induction, that $\tau_\alpha[\Delta]$ cannot be preceded by another τ_α time-step transition.

In the following section we will see that this invariant is strong enough to prove the safety property of the water-level monitor.

5 Example

We verified several (symbolic versions) of the case studies reported in the HyTech literature. Translation from a hybrid automaton [16] to a phase transition system is straightforward. Given a hybrid automaton $\mathcal{H} = \langle X, (V, E), init, inv, flow, jump \rangle$ where X is a set of variables, (V, E) a set of nodes and edges, $init$ a mapping from nodes to assertions denoting the initial condition, inv a mapping from nodes to assertions denoting node invariants, $flow$ a mapping from nodes to relations over $X \cup \dot{X}$ specifying the derivatives of the continuous variables, and $jump$ a mapping from edges to relations over $X \cup X'$ denoting the discrete transitions, the corresponding phase transition system is $\Psi = \langle X \cup \{s\}, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$, where s is a new (discrete) variable with domain V ,

$$\begin{aligned} \Theta &= \bigvee_{v \in V} (s = v \wedge init(v)), \\ \mathcal{T} &= \{\tau \mid \rho_\tau = jump(e) \wedge e \in E\}, \end{aligned}$$

$$\mathcal{A} = \{\alpha \mid \rho_\alpha = (s = v \rightarrow flow(v)) \wedge v \in V\},$$

$$\Pi = \bigwedge_{v \in V} s = v \rightarrow inv(v) .$$

5.1 Water-level monitor

To illustrate our methods we will describe the verification of the water-level monitor system shown in Figure 2, taken from [3]; its description as a hybrid

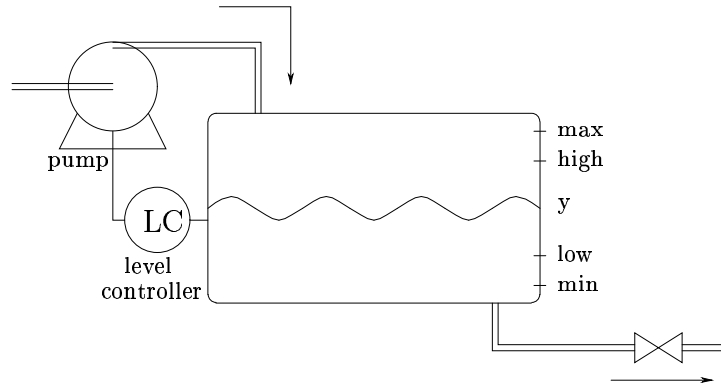


Fig. 2. Water-level monitor system

automaton is shown in Figure 3, and its description as a phase transition system, as entered in STeP is shown in Figure 4. The system consists of a watertank

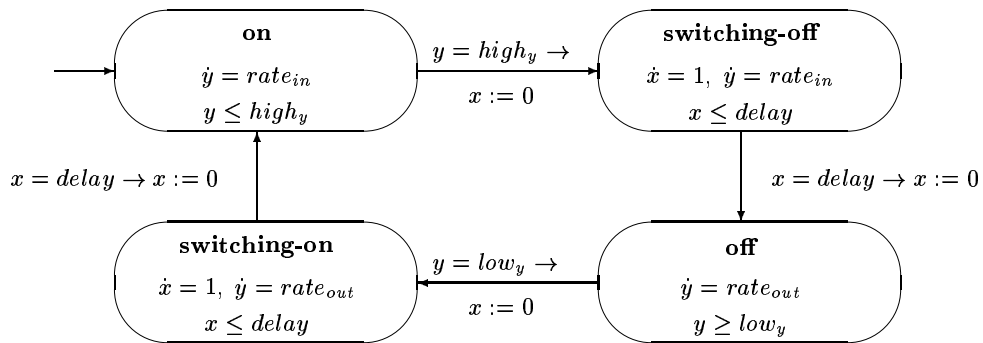


Fig. 3. Water level monitor – hybrid automaton

```

Hybrid Transition System Waterlevel Controller

type valveStates = {on, switching_off,off,switching_on}

in min_y, max_y : real where min_y <= max_y
in low_y: real where low_y >= min_y
in high_y: real where high_y <= max_y /\ high_y > low_y
in delay: real where delay > 0
in rate_in,rate_out: real where rate_in > 0 /\ rate_out < 0

local s : valveStates where s = on
clock x where x = 0
continuous y where y = low_y

Progress
(s = on --> y <= high_y) /\
(s = switching_off --> x <= delay) /\
(s = off --> y >= low_y) /\
(s = switching_on --> x <= delay)

Transition switch_off:
enable s = on /\ y = high_y
assign s := switching_off, x := 0

Transition isoff:
enable s = switching_off /\ x = delay
assign s := off, x := 0

Transition switch_on:
enable s = off /\ y = low_y
assign s := switching_on, x := 0

Transition ison:
enable s = switching_on /\ x = delay
assign s := on, x := 0

Activity Aon:
enable s = on \/ s = switching_off
assign Deriv(y) := rate_in

Activity Aoff:
enable s = off \/ s = switching_on
assign Deriv(y) := rate_out

```

Fig. 4. The Waterlevel Controller phase transition system

that supplies water to a customer at a constant rate. The level, y , in the tank is controlled by a controller, which observes the level via a level sensor. When the level drops below low_y , the controller starts a pump to refill the tank, and when the level rises above $high_y$ the pump is turned off again. When the pump is on, the level rises with rate $rate_{in}$, when the pump is off, the level drops with rate $rate_{out}$. However, there is a time delay of $delay$ seconds between the time the controller sends the signal to the pump and the time the flow is established or the pump is stopped

The property we want to prove about this system is that the level stays between a lower and upper limit, expressed by the linear-time temporal logic formula

$$safe : \Box(min_y \leq y \wedge y \leq max_y)$$

assuming there is sufficient margin between max_y and $high_y$, and between min_y and low_y , expressed by the axioms

$$\begin{aligned} max_y &\geq high_y + rate_{in} * delay \\ min_y &\leq low_y + rate_{out} * delay \end{aligned}$$

Not surprisingly, the property *safe* is not inductive, that is, after application of rule INV, taking $\varphi = p$, all first-order verification conditions simplify to true automatically except those for *Aon* and *Aoff*, which in fact are not valid. Rather than trying to strengthen the property to make it inductive, we generate the invariants described in Section 4.

We first generate $Post_D(true)$, which results (after simplification) in

$$Post_D(true) : x = 0 \wedge \left(\begin{array}{c} s = \mathbf{on} \vee s = \mathbf{off} \\ \vee \\ low_y = y \wedge s = \mathbf{switching-on} \\ \vee \\ high_y = y \wedge s = \mathbf{switching-off} \end{array} \right)$$

and we use this to generate $Post_A(Post_D \vee \Theta)$, resulting in

$$Post_A(Post_D \vee \Theta) : \left(\begin{array}{c} x > 0 \\ \wedge \\ (s = \mathbf{off} \rightarrow low_y \leq y) \\ \wedge \\ (s = \mathbf{on} \rightarrow y \leq high_y) \\ \wedge \\ ((s = \mathbf{switching-on} \vee s = \mathbf{switching-off}) \rightarrow x \leq delay) \\ \wedge \\ \left(\begin{array}{c} s = \mathbf{off} \vee s = \mathbf{on} \\ \vee \\ s = \mathbf{switching-on} \wedge y = low_y + rate_{out} \cdot x \\ \vee \\ s = \mathbf{switching-off} \wedge y = high_y + rate_{in} \cdot x \end{array} \right) \end{array} \right)$$

Taking the disjunction, we obtain the invariants we need to make the property *safe* inductive:

$$\begin{aligned} s = \mathbf{switching-on} &\rightarrow y = low_y + rate_{out} \cdot x \wedge x \leq delay \\ s = \mathbf{switching-off} &\rightarrow y = high_y + rate_{in} \cdot x \wedge x \leq delay \\ x &\geq 0 \end{aligned}$$

With these invariants the two remaining verification conditions simplify to true, where some of the non-linear clauses were proved by REDLOG [13].

Note that the system verified here cannot be verified by the current version of HyTech due to the use of symbolic constants for rate constants which results in non-linear terms.

5.2 Other Systems Verified

Other systems verified using STeP include the temperature controller [2], the railroad crossing, the three versions of the nuclear reactor (clock translation, linear approximation, and rectangular approximation) [5], and the cat and mouse example [22]. In most of these systems the automatic invariant generator generated some of the required invariants, but the user had to supply additional invariants to make the main safety property inductive. Verification of the above systems with symbolic constants instantiated with numbers were mostly automatic, apart from providing some invariants not provided by the automatic invariant generator. Verification of these systems with symbolic constants generally required some, usually trivial, user guidance in the interactive theorem prover. More examples of hybrid systems verified with STeP will appear on the STeP webpage: <http://rodin.stanford.edu/>.

6 Conclusion

We demonstrated the feasibility of computer-aided deductive verification of hybrid systems. We verified with STeP symbolic versions of (admittedly small) examples previously verified by HyTech. The verification of the symbolic versions usually required some user interaction, the verification of the instantiated systems (that is, the systems verified by HyTech) was mostly automatic (apart from providing some invariants). Currently the main limitation is the lack of decision procedures for real arithmetic, which makes it necessary to prove some, mathematically trivial, first-order verification conditions interactively, which is tedious. Hopefully this problem will be ameliorated with the integration of REDLOG.

Considering that the current implementation is still rather limited, our preliminary results suggest a high potential for deductive methods for the verification of hybrid systems.

Acknowledgements: We thank Nikolaj Bjørner and Tomás Uribe for their feedback and comments.

References

1. ABRIAL, J. R., BÖRGER, E., AND LANGMAACK, H., Eds. *Formal Methods for Industrial Applications*, vol. 1165 of *LNCS*. Springer-Verlag, 1996.
2. ALUR, R., COURCOUBETIS, C., HALBWACHS, N., HENZINGER, T. A., HO, P.-H., NICOLLIN, X., OLIVERO, A., SIFAKIS, J., AND YOVINE, S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 138, 1 (1995), 3–34.
3. ALUR, R., COURCOUBETIS, C., HENZINGER, T. A., AND HO, P.-H. Hybrid automata: An algorithmic approach to the specification and analysis of hybrid systems. In Grossman et al. [15], pp. 209–229.
4. ALUR, R., AND HENZINGER, T. A., Eds. *Proc. 8th Intl. Conference on Computer Aided Verification* (July 1996), vol. 1102 of *LNCS*, Springer-Verlag.
5. ALUR, R., HENZINGER, T. A., AND HO, P. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Engin.* 22, 3 (Mar. 1996), 181–201.
6. ARCHER, M., AND HEITMEYER, C. Verifying hybrid systems modeled as timed automata: A case study. In *Proc. 1st Intl. Workshop Hybrid and Real-time Systems (HART)* (1997), O. Maler, Ed., vol. 1201 of *LNCS*, Springer-Verlag.
7. BENSALÉM, S., LAKHNECH, Y., AND SAIDI, H. Powerful Techniques for the Automatic Generation of Invariants. In Alur and Henzinger [4], pp. 323–335.
8. BJØRNER, N. S., BROWNE, A., CHANG, E. S., COLÓN, M., KAPUR, A., MANNA, Z., SIPMA, H. B., AND URIBE, T. E. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Alur and Henzinger [4], pp. 415–418.
9. BJØRNER, N. S., BROWNE, A., CHANG, E. S., COLÓN, M., KAPUR, A., MANNA, Z., SIPMA, H. B., AND URIBE, T. E. STeP: The Stanford Temporal Prover, User's Manual. Tech. Rep. STAN-CS-TR-95-1562, Computer Science Department, Stanford University, Nov. 1995.
10. BJØRNER, N. S., BROWNE, A., AND MANNA, Z. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science* 173, 1 (Feb. 1997), 49–87. Preliminary version appeared in 1st Intl. Conf. on Principles and Practice of Constraint Programming, vol. 976 of *LNCS*, pp. 589–623, Springer-Verlag, 1995.
11. BJØRNER, N. S., MANNA, Z., SIPMA, H. B., AND URIBE, T. E. Deductive verification of real-time systems using STeP. In *4th Intl. AMAST Workshop on Real-Time Systems* (May 1997), vol. 1231 of *LNCS*, Springer-Verlag, pp. 22–43.
12. BJØRNER, N. S., STICKEL, M. E., AND URIBE, T. E. A practical integration of first-order reasoning and decision procedures. In *Proc. of the 14th Intl. Conference on Automated Deduction* (July 1997), vol. 1249 of *LNCS*, Springer-Verlag, pp. 101–115.
13. DOLZMANN, A., AND STURM, T. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin* 31, 2 (June 1997), 2–9.
14. FÜR INFORMATIONSTECHNIK BERLIN, K. Z. Z. REDUCE symbolic math system. <http://www.zib.de/Symbolik/reduce/>, 1995.
15. GROSSMAN, R. L., NERODE, A., RAVN, A. P., AND RISCHEL, H., Eds. *Hybrid Systems* (1993), vol. 736 of *LNCS*, Springer-Verlag.
16. HENZINGER, T. A. The theory of hybrid automata. In *Proc. 11th IEEE Symp. Logic in Comp. Sci.* (1996), IEEE Computer Society Press, pp. 278–292.
17. HENZINGER, T. A., AND HO, P. Algorithmic analysis of nonlinear hybrid systems. In Wolper [27], pp. 225–238.
18. HENZINGER, T. A., HO, P., AND WONG-TOI, H. A user guide to HYTECH. In *TACAS 95: First Intl. Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (1995), E. Brinksma, W. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, Eds., vol. 1019 of *LNCS*, Springer-Verlag, pp. 41–71.

19. HENZINGER, T. A., AND WONG-TOI, H. Linear phase-portrait approximations for nonlinear hybrid systems. In *Hybrid Systems III* (1996), R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., vol. 1066 of *LNCS*, Springer-Verlag, pp. 377–388.
20. HENZINGER, T. A., AND WONG-TOI, H. Using HyTECH to synthesize control parameters for a steam boiler. In Abrial et al. [1].
21. HO, P.-H., AND WONG-TOI, H. Automated analysis of an audio control protocol. In Wolper [27], pp. 381–394.
22. MANNA, Z., AND PNUELI, A. Clocked transition systems. In *Proc. of the Intl. Logic and Software Engineering Workshop* (Aug. 1995). Beijing, China.
23. MANNA, Z., AND PNUELI, A. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
24. NICOLLIN, X., OLIVERO, A., SIFAKIS, J., AND YOVINE, S. An approach to the description and analysis of hybrid systems. In Grossman et al. [15], pp. 149–178.
25. SU, J. X., DILL, D. L., AND BARRETT, C. W. Automatic generation of invariants for processor verification. In *1st Intl. Conf. on Formal Methods in Computer-Aided Design* (Nov. 1996), vol. 1166 of *LNCS*, Springer-Verlag, pp. 377–388.
26. VITT, J., AND HOOMAN, J. Assertional specification and verification using PVS of the steam boiler control system. In Abrial et al. [1], pp. 453–472.
27. WOLPER, P., Ed. *Proc. 7th Intl. Conference on Computer Aided Verification* (July 1995), vol. 939 of *LNCS*.