Modeling and Verifying a Temperature Control System using Continuous Action Systems

Ralph-Johan Back Cristina Cerschi
Turku Centre for Computer Science (TUCS),
Lemminkäisenkatu 14 A, FIN-20520, Turku, Finland

Abstract. We describe and verify a real-time temperature control system for a nuclear reactor tank, using a generalization of action systems to hybrid systems as our formal framework. The analyzed control system is a linear hybrid system, combining discrete control with continuous dynamics. Our work can be seen as a case study on the applicability of the hybrid action system formalism to study the reachability problem, i.e., to prove that an unsafe state can not be reached by executing the system.

Keywords Action systems, Continuous action systems, Forward analysis, Invariance property, Reachability

1 Introduction

The safety-critical nature of most hybrid control systems has encouraged work on their formal modeling and verification. The goal is to provide a mathematical proof, which shows that a given model of a system satisfies its safety requirements. In this paper, we apply the generalized model of action systems for hybrid systems, developed by Back, Petre and Porres [3], to formally describe the real-time linear temperature control system inside a nuclear reactor tank. The model allows us to analyze the reachability properties of the system, in particular to prove that by executing the system we can not reach an unsafe state.

The action systems formalism, introduced by Back and Kurki-Suonio [4], is used to model and analyze parallel and distributed systems. The behavior of an action system is essentially that of Dijkstra’s guarded iteration statement [6] on the state variables: the initialization statement is executed first, thereafter, as long as there are enabled actions, one action at a time is nondeterministically chosen and executed. The extension of this approach to continuous action systems [3] provides a unified framework for handling both discrete and continuous behavior. The continuous action system can be seen as a collection of time functions, defined in a piecewise manner. This approach lets us model both discrete and continuous functions in the same way, without separating discrete events from continuous laws. The system’s formal representation is therefore homogenous, making its behavior easy to understand and reason about.

For general hybrid systems, the hybrid automata analysis methods [1, 2], can be applied with certain limitations. The automated analysis of hybrid systems [1] needs efficient algorithms to represent and approximate state sets. The verification methodology that is based on abstracted automata [14], which has simpler dynamics, faces the inconvenience that the abstractions that
are created depend on the property to be proved. Different specifications may require different abstractions, so proving different properties of the same hybrid system implies creating different abstractions.

This case study demonstrates how our model [3] can be applied to specify and prove safety properties for a temperature control system. This is a typical example of a hybrid control system. We show that we can use the same verification techniques for hybrid systems as for ordinary action systems.

The structure of this paper is as follows. In section 2 we briefly present the basic action system notion. Section 3 introduces the continuous action system approach. In section 4, we describe the temperature control system, using the continuous action system model. We also give the semantics of the system, by translating it into an equivalent (discrete) action system where time is explicitly advanced. Section 5 proves that the undesired state is not reachable, by proving an invariance property for the hybrid system, using the traditional forward analysis technique. Some aspects of the proofs in this section are presented in the Appendix. Section 6 presents an attempt to evaluate the formal model proposed in this paper and the verification technique used, following the discussion in [10]. Conclusions and related work are presented in section 7.

2 Action Systems

The action systems formalism [4] is a framework for specification and refinement of concurrent programs. It is based on an extended version of the guarded command language of Dijkstra [6].

An action system is in general a collection of actions, or guarded commands, which are executed one at a time. Parallel behavior is modeled by interleaved actions, i.e., by two or more actions that can be executed in any order.

An action system $A$ is a statement of the form:

$$A \overset{\text{def}}{=} [[ \text{var } x : T \cdot S_0 ; \text{do } A_1 \cdot \ldots \cdot A_m \text{ od } ]] : y$$

on state variables (attributes) $x \cup y$. The local variables $x$ only exist during the execution of the action system. They are first initialized by $S_0$, after which the actions $A_1\ldots A_m$ are executed repeatedly, as long as one of them is enabled. The global variables $y$ exist before and after the execution of the action system. For specifying the state attributes, we define a finite set $Attr$ of attribute names and assume that each attribute name in $Attr$ is associated with a non-empty set of values. This set of values is the type of the attribute. If the attribute $x$ takes values from $Val$, we say that $x$ has the type $Val$ and we write it as $x : Val$. The name and the type completely specify the attribute. We consider several predefined types, like “Real” for the set of real numbers, “Real,” for the set of non-negative real numbers and “Bool” for the boolean values $\{T, F\}$.

An action $A$ is of the form

$$A ::= g \rightarrow S,$$

where $g$ is the guard of $A$ and $S$ is the statement (body) of $A$.

A statement $S$ is defined by the grammar:

$$S ::= \text{abort} \quad \text{(abortion, nontermination)}$$
where $S_1$, ..., $S_m$ are statements, $g_1$... $g_m$ are predicates (boolean conditions), $x$ a variable or a list of variables, and $e$ an expression or a list of expressions. The action “abort” always fails and is used to model disallowed behaviors. Actions can be much more general, but this simple syntax suffices for the purpose of this paper. An action $g \rightarrow S$ is enabled if its guard $g$ holds.

The execution of an action system is as follows. The initialization $S_0$ sets the variables to some specific values, using a sequence of assignments. Then enabled actions are repeatedly chosen and executed. The chosen actions will change the values of the variables in a way that is determined by the action body. Two or more actions can be enabled at the same time, in which case one of them is chosen for execution, in a demonically nondeterministic way. The computation terminates when no action is enabled. Termination of an action system means the termination of the control over the system, which means that the state will evolve no more, fixing the final values of the variables forever.

An action system is not usually regarded in isolation, but as a part of a more complex system. The rest of the system (the environment) communicates with the action system via shared (imported and exported) variables, referred to as the global variables.

A predicate $I$ is an invariant of the action system $A$, if it:

1. holds after the initialization, i.e., if
   
   $\text{true }\{S_0\} I$

2. is preserved by each action $g_i \rightarrow S_i$, i.e., if
   
   $I \land \{S_i\} I, \ i = 1, ..., m$.

Here $p \{S\} q$ denotes the standard partial correctness of statement $S$ with respect to precondition $p$ and postcondition $q$ [7].

3 Continuous Action Systems

A continuous action system [3] consists of a set of attributes that form the state of the system and a set of actions that act upon the attributes:

$$C = \text{def } |(\text{var } x: T \bullet S_0; \text{ do } g_1 \rightarrow S_1 \cdots \circ g_m \rightarrow S_m \text{ od })| : y$$ (2)

Here $x = x_1, \ldots, x_n$ are the controlled attributes or program variables of the system, $S_0$ is a statement that initializes these attributes, while $g_i \rightarrow S_i, i = 1, \ldots, m$, are the actions of the system. The attributes $y = y_1, \ldots, y_k$ are defined in the environment of the continuous action system. An attribute $x$ is a function of time, where time is assumed to vary over the non-negative real numbers. The value of an attribute can be read and its value can be changed.
Changing the value means assigning to the attribute a new time function that may change the future behavior of the attribute but not its past.

An implicit variable \textit{now} is used to denote the present time and can be referred to in expressions. By using the \textit{now} variable in an expression, we can correlate the behavior of the model with the passage of time. Therefore, this formalism is well-suited for modeling real-time systems.

The initialization statement will set the attributes to some specific functions of time, using a sequence of assignments. The actions will change the values of the attributes in the prescribed way, provided they are enabled. This model allows two or more actions to be enabled at the same time, in which case one of the enabled actions is chosen for execution, in a (demonically) nondeterministic way.

The next time instance when an action is enabled may well be the same as the previous time instance when an action was enabled, i.e., time need not progress between two enabled actions. This allows us to model both discrete computation and continuous behavior in the same framework (a discrete computation does not take any time).

Functions are below described using \( \lambda \) - abstraction, and we write \( f.x \) for the application of function \( f \) to argument \( x \). We explain the meaning of \( C \) by translating it into an ordinary action system. The continuous action system’s \( (C) \) semantical interpretation is given by the following (discrete) action system \( \bar{C} \):

\[
\bar{C} = \left[ \begin{array}{l}
\text{var } \text{now}: \text{Real}, \ x: \text{Real}\rightarrow T \bullet \\
\text{now} : = 0; \ \bar{S}_0; \ N \\
\text{do} \\
\quad \bar{g}_1 \rightarrow \bar{S}_1; \ N \mid \ldots \mid \bar{g}_m \rightarrow \bar{S}_m; \ N \\
\text{od} \\
\end{array} \right] : y,
\]

where

\[
N = \text{def } \text{now} : = \text{next.gg.now}
\]

and the operation “next” is defined by:

\[
\text{next.gg.t} = \text{def } \min \{ t' \geq t \mid \text{gg.t'} \} , \text{ if } \exists t' \geq t \text{ such that } \text{gg.t'}
\]

, otherwise

In \( \bar{C} \), the attribute \textit{now} is declared, initialized and updated explicitly. It models the moments of time that are of interest for the system, i.e., the starting time and the succeeding moments when some action is enabled. The value of \textit{now} is updated by the statement “N”. In the definition of “next”, \( \text{gg} = g_1 \lor \ldots \lor g_m \) is the disjunction of all guards of the actions. Thus, the function “next” models the moments of time when at least one action is enabled. Only at these moments can the future behavior of attributes be modified. If no action will be ever enabled, then the second branch of the definition will be followed. In this case the system terminates, i.e., the attributes will evolve forever according to the functions assigned to last. We assume in this paper that the minimum in the definition of “next” always exists, i.e., a \textit{continuous action system} is well-defined when \( \min \{ t' \geq t \mid \text{gg.t'} \} \) is well-defined.

Let us introduce the notation:
Thus, \( f / t_0 / g \) behaves as \( f \) before \( t_0 \), and as \( g \) after \( t_0 \).

In \( C \), the condition \( \overline{g} \) stands for the application of \( g \) to \( \text{now} \). For instance, \( x = 0 \) denotes \( (x = 0).\text{now} \equiv (x.\text{now} = 0).\text{now} \equiv (x.\text{now}) = 0 \). An assignment \( x_i := e \) in \( S_i \) is again understood as denoting the following assignment in \( \overline{S}_i \):

\[
x_i := x_i / \text{now} / e
\]

The statement \( \overline{S}_i \) is \( S_i \) with these changes. The continuous action system is essentially just a way of defining a collection of time functions \( x_1, \ldots, x_n \) over the non-negative reals, in a stepwise manner. The steps form a sequence of intervals \( I_0, I_1, I_2, \ldots \), where each interval \( I_k \) is either a left closed interval of the form \( [t_k, t_{k+1}) \) or a closed interval of the form \( [t_k, t_{k}] \), i.e., a point. The action system determines a family of functions \( x_1, \ldots, x_n \), which are stepwise defined over this sequence of intervals and points. The extremes of these intervals correspond to the control points of the system where a digital discrete action is performed.

Another important observation regards the possibility of Zeno behavior. The definition of a continuous action system proposed in [3] does not guarantee that the sequence of generated intervals will cover all the non-negative reals. They might only cover an initial segment of these. In this case, there is a limit point of time that the action system reaches when the number of iterations reaches infinity. The simple explanation of the behavior of the hybrid system is then not sufficient. However, in that case, we assume that the system is restarted at the limit point, and repeat the process again. This is meaningful if all attribute values converge to a well defined value in the limit. This restart can be carried out as many times as needed. Thus, a continuous action system may have multiple limit points in its execution. The standard action system semantics does not allow multiple limit points, so this is a point where the semantics really has to be extended. For simplicity we will here assume that the system \( \tilde{C} \) does not exhibit Zeno behavior, i.e., that the value of \( \text{now} \) grows without bounds if the action system \( \tilde{C} \) does not terminate. Thus, a single limit point is sufficient. The absence of Zeno behavior means that the action system will define the values of the attributes for the whole domain of Real_+.

4 The Temperature Control System (TCS)

Our example system is taken from a study of hybrid systems using algorithmic techniques, by Alur et al. [1]. The system controls the coolant temperature in a reactor tank by moving two independent control rods. Controlling a nuclear reactor means controlling the multiplication of neutrons in the reactor core. When the control rods (which are made of materials that absorb neutrons) are pulled out of the core, more neutrons are available and the chain reaction speeds up, producing more heat. If they are inserted into the core, more neutrons are absorbed, and the chain reaction slows or stops, reducing the heat.

The goal is to maintain the coolant between the minimum temperature \( \theta_m \) and the maximum temperature \( \theta_M \). When the temperature reaches its maximum value \( \theta_M \) the tank must be refrigerated with one of the rods. The temperature rises at a rate \( v_r \) and decreases at rates \( v_1 \) and \( v_2 \), depending on which rod is being used. For safety reasons, a rod can be moved again only if
T time units have elapsed since the end of its previous movement. If the temperature of the coolant can not decrease because there is no available rod, a complete shutdown is required.

4.1 The Continuous Action System Model

This system can be described as a continuous action system, as follows. The system’s variables are:

- \( \theta \) that measures the temperature inside the reactor tank
- \( x_1 \) that measures the time elapsed since the last use of rod1
- \( x_2 \) that measures the time elapsed since the last use of rod2
- \( state \) that stores the state of the system.

In order to correlate the execution of an action with the passage of time, we introduce a clock variable, which measures the time elapsed since it was set to zero.

The operation

\[
\text{reset} \ (c) =_{\text{def}} c := (\lambda t \cdot t - \text{now})
\]

will reset the clock.

Note that the assignment \( c := (\lambda t \cdot t - \text{now}) \) in the hybrid action system really stands for

\[
c := (\lambda t \cdot \text{if } t < \text{now} \text{ then } c.t \text{ else } t - \text{now} \text{ fi})
\]

in the translation of this system to an action system with explicit time.

The continuous action system for describing the temperature control system is as follows:

\[
TCS = \{ \begin{array}{l}
\text{var} \quad state : \text{Real}; \\
x_1, x_2, c : \text{Real}; \\
\theta : \text{Real}; \\
state := 0; \\
\text{reset}(c); \\
x_1 := (\lambda t \cdot T_1 + c.t); \\
x_2 := (\lambda t \cdot T_2 + c.t); \\
\theta := (\lambda t \cdot \theta_0 + v_r \cdot t)
\end{array}
\}
\]

\[
\text{do } \{ \text{cool with rod1} \}
\]

\[
\begin{array}{l}
state = 0 \land \theta = \theta_M \land x_1 \geq T \rightarrow \\
\text{reset} \ (c); \\
\theta := (\lambda t \cdot \theta_M - v_1 \cdot c.t); \\
state := 1
\end{array}
\]

\[
\text{[] } \{ \text{release rod1} \}
\]

\[
\begin{array}{l}
state = 1 \land \theta = \theta_m \rightarrow \\
\text{reset} \ (c); \text{reset} \ (x_i); \\
\theta := (\lambda t \cdot \theta_m + v_i \cdot c.t); \\
state := 0
\end{array}
\]

\[
\text{[] } \{ \text{cool with rod2} \}
\]
\[\text{state} = 0 \land \theta = \theta_M \land x_2 \geq T \rightarrow\]
\[\text{reset} (c);\]
\[\theta := (\lambda t \cdot \theta_M - v_2 \cdot c \cdot t);\]
\[\text{state} := 2\]

\[\text{release rod2}\]
\[\text{state} = 2 \land \theta = \theta_m \rightarrow\]
\[\text{reset} (c); \text{reset} (x_2);\]
\[\theta := (\lambda t \cdot \theta_m + v_r \cdot c \cdot t);\]
\[\text{state} := 0\]

\[\text{shutdown}\]
\[\text{state} = 0 \land \theta = \theta_M \land x_1 < T \land x_2 < T \rightarrow\]
\[\text{abort}\]

\text{od}
\[\theta_0, \theta_m, \theta_M, v_r, v_1, v_2, T_1, T_2, T\]

Here, \(T_1, T_2, \theta_0\) are constants, so that \(0 \leq T_1 \leq T, 0 \leq T_2 \leq T, 0 \leq \theta_0 \leq \theta_M\).

The system is first initialized to state 0, the clock is reset and at time point zero, inside state 0, we have \(x_1 := T_1, x_2 := T_2, \theta := \theta_0\). After this, the system starts evolving by increasing the time point continuously. The first action is enabled when the system has reached the maximum temperature \(\theta_M\) and the first rod is available \(x_1 \geq T\). The first action body is then executed: the clock is reset, and the tank is refrigerated with rod1 (the temperature \(\theta\) starts decreasing linearly at rate \(v_1\)), and the system enters state 1 by a discrete transition. The second action is enabled when the temperature reaches its minimum value \(\theta_m\) and state = 1. The second action body is then executed: both clock variable \(c\) and clock \(x_1\) (that measures the passed time since the previous movement of rod1) are reset and the system returns to state 0, where \(\theta\) increases linearly at rate \(v_r\). Similarly to the first action, action 3 is enabled if the system has reached the maximum temperature \(\theta_M\) and the second rod is available \(x_2 \geq T: \text{at least T time units have passed since it has been last used}\). The system then enters state 2 where the temperature starts decreasing at rate \(v_2\). Action 4, being symmetric to action 2, is enabled when the temperature reaches its minimum value \(\theta_m\), and this time state = 2. The temperature then starts increasing at rate \(v_r\). The last action has abort as its body, thus expressing that the shutdown state is not desired. It becomes enabled when the system is in state 0, reaches its maximum temperature \((\theta = \theta_M)\) and none of the rods is available: \(x_1 < T \land x_2 < T\).

### 4.2 The Translated Action System Model

The translated action system \(\overline{TCS}\), where time is explicitly advanced is as follows:

\[
\overline{TCS} = \| \text{var} \ \text{state} : \text{Real} \rightarrow \{0,1,2,3\}; \]
\[
x_1, x_2, c : \text{Real} \rightarrow \text{Real}; \]
\[
\theta : \text{Real} \rightarrow \text{Real}; \]
\[
\text{start, now} : \text{Real}^* \]
\[
\text{now} := 0; \]
\[
\text{state} := (\lambda t \cdot 0); \]
\[
c := (\lambda t \cdot t); \]

Here, \(T_1, T_2, \theta_0\) are constants, so that \(0 \leq T_1 \leq T, 0 \leq T_2 \leq T, 0 \leq \theta_0 \leq \theta_M\).
\[ x_1 := (\lambda t \cdot T_1 + c \cdot t); \]
\[ x_2 := (\lambda t \cdot T_2 + c \cdot t); \]
\[ \theta := (\lambda t \cdot \theta_0 + v_r \cdot t); \]
\[ \text{start} := \text{now}; \]
\[ \text{now} := \min \{t' \geq \text{now} \mid \text{gg}.t'\}; \]
\[ \text{do} \{ \text{cool with rod1} \}
\[ \text{state} := 0 \land \theta := \theta_0 \land x_1 \land \text{now} \geq T \rightarrow \]
\[ c := c / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ \theta := \theta / \text{now} / (\lambda t \cdot \theta_0 - v_1 * c \cdot t); \]
\[ \text{state} := \text{state} / \text{now} / (\lambda t \cdot 1); \]
\[ \text{start} := \text{now}; \]
\[ \text{now} := \min \{t' \geq \text{now} \mid \text{gg}.t'\}; \]
\[ \text{\{release rod1\}} \]
\[ \text{state} := 1 \land \theta := \theta_0 \land x_1 \land \text{now} \geq T \rightarrow \]
\[ c := c / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ x_1 := x_1 / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ \theta := \theta / \text{now} / (\lambda t \cdot \theta_0 + v_r * c \cdot t); \]
\[ \text{state} := \text{state} / \text{now} / (\lambda t \cdot 2); \]
\[ \text{start} := \text{now}; \]
\[ \text{now} := \min \{t' \geq \text{now} \mid \text{gg}.t'\}; \]
\[ \text{\{cool with rod2\}} \]
\[ \text{state} := 0 \land \theta := \theta_0 \land x_2 \land \text{now} \geq T \rightarrow \]
\[ c := c / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ x_2 := x_2 / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ \theta := \theta / \text{now} / (\lambda t \cdot \theta_0 - v_2 * c \cdot t); \]
\[ \text{state} := \text{state} / \text{now} / (\lambda t \cdot 2); \]
\[ \text{start} := \text{now}; \]
\[ \text{now} := \min \{t' \geq \text{now} \mid \text{gg}.t'\}; \]
\[ \text{\{release rod2\}} \]
\[ \text{state} := 2 \land \theta := \theta_0 \land x_2 \land \text{now} \geq T \rightarrow \]
\[ c := c / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ x_2 := x_2 / \text{now} / (\lambda t \cdot t - \text{now}); \]
\[ \theta := \theta / \text{now} / (\lambda t \cdot \theta_0 + v_r * c \cdot t); \]
\[ \text{state} := \text{state} / \text{now} / (\lambda t \cdot 2); \]
\[ \text{start} := \text{now}; \]
\[ \text{now} := \min \{t' \geq \text{now} \mid \text{gg}.t'\}; \]
\[ \text{\{shutdown\}} \]
\[ \text{state} := 0 \land \theta := \theta_0 \land x_2 \land \text{now} \land T \land x_2 \land \text{now} < T \rightarrow \]
\[ \text{\{shutdown\}} \]
\[ \text{\} : \theta_0, \theta_m, \theta_0, v_r, v_1, v_2, T_1, T_2, T \]

Variable now has been explicitly introduced and the assignment now := min{t' ≥ now | gg.t'} gives the next time instance when the disjunction of the guards of the actions, g_1 ∨ g_2 ∨ g_3 ∨ g_4 ∨ g_5 holds (i.e., the next time when at least one guard is true, so some action is enabled).
We have introduced here the variable \( \text{start} \), which stores the time moment when the system starts evolving in any state, after taking a discrete transition.

5 Reachability Verification

We know that if the temperature rises to its maximum and it cannot decrease because no rod is available, a complete shutdown is required. The question is whether the system will ever reach the shutdown state. A state \( \sigma' \) is reachable from the state \( \sigma \) if there is a run of the hybrid system \( H \) that starts in \( \sigma \) and ends in \( \sigma' \) [1]. Usually, we want to prove that some bad condition \( g \) (like the shutdown condition) is not reachable. This we can do by proving that some condition \( I \) is an invariant of the system, and that \( I \Rightarrow \neg g \). As every reachable state satisfies \( I \), this then shows that every reachable state satisfies \( \neg g \), i.e. a state where \( g \) holds cannot be reached.

Let \( \Delta \theta = \theta_M - \theta_m \). Clearly, the time the coolant needs to increase its temperature from \( \theta_m \) to \( \theta_M \) is \( \tau_r = \Delta \theta / v_r \), and the refrigeration times using rod1 and rod2 are \( \tau_1 = \Delta \theta / v_1 \) and \( \tau_2 = \Delta \theta / v_2 \), respectively.

The sequence of heating and refrigeration is shown in Fig. 1:

![Fig. 1. Heating and refrigeration times](image)

Clearly, if \( \tau_r \geq T \) (temperature rises at a rate slower than the time of recovery of the rods), then the shutdown state is unreachable. However, this can be a far too strong condition for avoiding the undesired state. Inspecting Fig. 1 we find a weaker condition:

\[
2\tau_1 + \tau_r \geq T \land 2\tau_2 + \tau_r \geq T
\]  
(7)

i.e., if the time between two insertions of the same rod is greater than the time of recovery of the rod, the shutdown state is not reachable. We will prove that this is a sufficient condition for avoiding the undesired state. We therefore make the general assumption that relation (7) is true.

In order to prove that formula \( I \) is an invariant of our system, it is sufficient to prove that

\[
\text{true} \{ \emptyset; N \} I
\]  
(8a)

and
where
\[ N = \begin{cases} \text{start} : = \text{now}; \\
\text{now} : = \min \left\{ t' \geq \text{now} \mid gg.t' \right\}, \end{cases} \]

assigns variable \text{start} to \text{now} first, and afterwards sets \text{now} to the next time instance when the disjunction of the guards of the actions in $TCS$, $(gg = g_1 \lor g_2 \lor g_3 \lor g_4 \lor g_5)$ holds (i.e., the next moment when at least one action is enabled).

### 5.1 Expressing the Invariant with the State chart

Finding the right invariant for proving a safety property is far from being trivial. Therefore, we start by generating the state chart of the temperature control system to get a first approximation of the invariant. Then, we keep adding information to the system’s states in order to figure out an invariant strong enough to ensure safety.

The following state chart shows the states that the system can be in, and the properties that hold in each state. It is essentially a hybrid automaton view of the temperature control system.

![The Temperature Control System’s Hybrid Automaton](image)

This figure describes a first invariant of the system. The invariant is the following (expressed in terms of $TCS$):

\[ I \land \bar{g}_i \{ | S_i \cup N | \} I, \ i = 1, \ldots, 5 \] (8b)
\[ I = \text{def} \ (\forall \; t \in \; [\text{start}, \; \text{now}] \cdot \\
\text{state.start} = 0 \Rightarrow \text{state}.t = 0 \land \\
d\theta / dt = v_i \; \text{in} \; [\text{start}, \; \text{now}] \land \\
dx_1 / dt = 1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
x_2 / dt = 1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
\theta.\text{start} = \theta_m \land (x_1.\text{start} = 0 \lor x_1.\text{start} = 0) \land \\
\text{state.start} = 1 \Rightarrow \text{state}.t = 1 \land \\
d\theta / dt = -v_1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
dx_1 / dt = 1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
x_2 / dt = 1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
\theta.\text{start} = \theta_m \land x_1.\text{start} \geq T \land \\
\text{state.start} = 2 \Rightarrow \text{state}.t = 2 \land \\
d\theta / dt = -v_2 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
x_1 / dt = 1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
x_2 / dt = 1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
\theta.\text{start} = \theta_m \land x_2.\text{start} \geq T \land \\
\text{state.start} = 3 \Rightarrow \theta.\text{start} = \theta_m \land x_1.\text{start} < T \land x_2.\text{start} < T) \] 

The invariant thus shows the basic continuous behavior that holds in each state, as well as the discrete transitions.

It is easy to check on the translated form of the temperature control system, that it really has the properties described by the above state chart. By inspecting in TCS the expressions of each action’s guard and each action’s body, proving that (9) holds becomes trivial.

### 5.2 Finding a Stronger Invariant

Although we have extracted a first form of the invariant, we need to find a stronger one, in order to be able to prove that \text{state}3 is unreachable.

Adding information to the basic state features encapsulated in relation (9), leads us to a new invariant. We can add property \( \theta \leq \theta_m \) to each of the states 0, 1 and 2.

We obtain the following stronger invariant, also expressed in terms of TCS:

\[ I' = \text{def} \ (\forall \; t \in \; [\text{start}, \; \text{now}] \cdot \text{state.start} = 0 \Rightarrow \; \theta.t \leq \theta_m \land \\
\text{state}.t = 0 \land d\theta / dt = v_i \; \text{in} \; [\text{start}, \; \text{now}] \land \\
(dx_1 / dt = 1, \; dx_2 / dt = 1) \; \text{in} \; [\text{start}, \; \text{now}] \land \\
\theta.\text{start} = \theta_m \land (x_1.\text{start} = 0 \lor x_1.\text{start} = 0) \land \\
\text{state.start} = 1 \Rightarrow \; \theta.t \leq \theta_m \land \\
\text{state}.t = 1 \land d\theta / dt = -v_1 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
(dx_1 / dt = 1, \; dx_2 / dt = 1) \; \text{in} \; [\text{start}, \; \text{now}] \land \\
\theta.\text{start} = \theta_m \land x_1.\text{start} \geq T \land \\
\text{state.start} = 2 \Rightarrow \; \theta.t \leq \theta_m \land \\
\text{state}.t = 2 \land d\theta / dt = -v_2 \; \text{in} \; [\text{start}, \; \text{now}] \land \\
(dx_1 / dt = 1, \; dx_2 / dt = 1) \; \text{in} \; [\text{start}, \; \text{now}] \land \\
\theta.\text{start} = \theta_m \land x_1.\text{start} \geq T \land x_2.\text{start} < T) \]
\[ \theta_{\text{start}} = \theta_m \land x_2 \text{start} \geq T \land \text{state.start} = 3 \Rightarrow \theta_{\text{start}} = \theta_m \land x_1 \text{start} < T \land x_2 \text{start} < T \]

The enriched state chart is shown in the following figure:

Fig. 3. The TCS state chart with the added property, \( \theta \leq \theta_m \)

Let us show that:

\[
I_{\theta} \overset{\text{def}}{=} (\forall t \in [\text{start, now}) \bullet \\
\text{state.start} = 0 \Rightarrow (\theta_t \leq \theta_m \land \text{state.t} = 0) \\
\land \text{state.start} = 1 \Rightarrow (\theta_t \leq \theta_m \land \text{state.t} = 1) \\
\land \text{state.start} = 2 \Rightarrow (\theta_t \leq \theta_m \land \text{state.t} = 2)) 
\]

is a property of the temperature control system.

We apply standard forward analysis technique on the translated model of the temperature control system. Thus, we have to prove that (10) is established by the initialization and then that it is preserved by every action. We show here the proofs for the initialization statement and for action 1 (cooling with rod1). The calculation of \( gg \) for the proofs is presented in the Appendix.

We assume \( v_1, v_2, v_1 \in \mathbb{R}\setminus\{0\}, \theta_m, \theta_M \geq 0 \) and that the choice of the rod to use as coolant is demonically nondeterministic, in case both rods are available.

**Proof of (10)**

(10a) **Initialization**
We have to prove that \( \text{true} \} \overline{\delta}_0; N \} I_\theta' \) holds. The initialization statement \( \overline{\delta}_0; N \) establishes that

\[
\begin{align*}
\text{now} & = 0; \\
\text{state} & = (\lambda t \cdot 0); \\
c & = (\lambda t \cdot t); \\
x_1 & = (\lambda t \cdot T_1 + c.t); \\
x_2 & = (\lambda t \cdot T_2 + c.t); \\
\theta & = (\lambda t \cdot \theta_0 + v_r * t); \\
\text{start} & = \text{now}; \\
\text{now}' & = \min \{ t' \geq \text{now} \mid gg.t' \}
\end{align*}
\]

We have to prove that the partial invariant \( I_\theta' \) is satisfied by these assignments. Thus, we have that

\[
\begin{align*}
I_\theta' [\text{now}' / \text{now}] & \equiv \{ \text{definition of the invariant} \} \\
(\forall t \in [\text{start}, \text{now}']) \bullet (\lambda t \cdot 0).\text{start} = 0 \Rightarrow (\lambda t \cdot \theta_0 + v_r * t).t \leq \theta_M \wedge (\lambda t \cdot 0).t = 0 \\
& \wedge (\lambda t \cdot 0).\text{start} = 1 \Rightarrow (\lambda t \cdot \theta_0 + v_r * t).t \leq \theta_M \wedge (\lambda t \cdot 0).t = 1 \\
& \wedge (\lambda t \cdot 0).\text{start} = 2 \Rightarrow (\lambda t \cdot \theta_0 + v_r * t).t \leq \theta_M \wedge (\lambda t \cdot 0).t = 2) \\
& \equiv \{ \text{start} = \text{now} = 0, \text{now}' = \min \{ t' \geq 0 \mid gg.t' \} \} \\
(\forall t \in [0, \text{now}']) \bullet (\lambda t \cdot \theta_0 + v_r * t).t \leq \theta_M) \\
& \equiv \{ \text{now}' = \min \{ t' \geq 0 \mid \theta_0 + v_r * t = \theta_M \} \} \\
(\forall t \mid 0 \leq t < (\theta_M - \theta_0) \vee t \leq (\theta_M - \theta_0)) \\
& \equiv \{ \text{logic} \} \\
& \text{true}
\end{align*}
\]

Thus, we have showed that \( I_\theta' \) holds after the initialization, i.e., that it holds from the moment 0 until the first moment an action is enabled. Next, we shall compute the verification condition for the first action (cooling with rod1) and show that the invariant also holds after this action.

\[(10b)\] Cooling with rod1

We assume that \( I_\theta' \) holds on \([\text{start}, \text{now}]\), that \( \overline{\gamma}_1 \) is true and that the local variables have been updated by the assignments of the body of action 1. Thus, we assume that

\[
I_\theta' \\
\wedge \text{state}.\text{now} = 0 \wedge \theta.\text{now} = \theta_M \wedge x_1.\text{now} \geq T \\
\wedge c' = c \mid \text{now} / (\lambda t \cdot t \cdot \text{now}) \\
\wedge \theta' = \theta / \text{now} / (\lambda t \cdot \theta_M - v_1 \ast c.t) \\
\wedge \theta'.\text{state} = \text{state} / \text{now} / (\lambda t \cdot 1) \\
\wedge \text{start}' = \text{now} \\
\wedge \text{now}' = \min \{ t' \geq \text{now} \mid gg.t' \}
\]

We have to prove that the added information in the invariant holds after these assignments.

We have that

\[ I_\theta' \left[ c' \theta' \text{ state}', \text{ start}', \text{ now}' \right] \]

\[ \leq \{ \text{definition of } I_\theta' \} \]

\[(\forall t \mid \text{ start}' \leq t < \text{ now}' \bullet \text{ state}'.\text{start}' = 0 \Rightarrow (\theta'.t \leq \theta_M \land \text{ state}'.t = 0) \]
\[\land \text{ state}'.\text{start}' = 1 \Rightarrow (\theta'.t \leq \theta_M \land \text{ state}'.t = 1) \]
\[\land \text{ state}'.\text{start}' = 2 \Rightarrow (\theta'.t \leq \theta_M \land \text{ state}'.t = 2) \]

\[\leq \{ \text{replacing updated variables state}', \theta', \lambda - \text{reduction, computing that now}' = \text{ now} + \tau_1 \} \]

\[(\forall t \mid \text{ now} \leq t < (\text{ now} + \tau_1) \bullet 1 \Rightarrow (\theta_M - v_1 \ast (t - \text{ now}) \leq \theta_M \land 1 = 1) \]
\[\land 1 = 1 \Rightarrow (\theta_M - v_1 \ast (t - \text{ now}) \leq \theta_M \land 1 = 1) \]
\[\land 1 = 2 \Rightarrow (\theta_M - v_1 \ast (t - \text{ now}) \leq \theta_M \land 1 = 2) \]

\[\leq \{ \text{logic} \} \]

\[(\forall t \mid \text{ now} \leq t < (\text{ now} + \tau_1) \bullet \theta_M - v_1 \ast (t - \text{ now}) \leq \theta_M \]

\[\leq \{ \theta' = \theta_M - v_1 \ast (t - \text{ now}) \text{ is decreasing starting from } \theta_M, v_1 > 0, (t - \text{ now}) \geq 0 \} \]

true

Therefore, we have proved that \( I(I \land I_\theta') \) holds after the discrete transition from state0 to state1 is taken. The proofs for the rest of possible safe transitions are similar, and are omitted here.

5.3. **Expressing the Final Invariant and Proving the Safety Property of TCS**

Our final goal is to provide sufficient assurance that the system evolves on the safe side. This is equivalent to proving that state3 can never be reached. Informally, the safety property reduces to proving that in any state, \( \theta \leq \theta_M \), and also that in state0 we always have (at least) one available rod, i.e., that \( (x_1 \geq T \lor x_2 \geq T) \).

It follows that even though the invariant we have found is added with some new condition, it is still weak, i.e., it can not ensure safety. Thus, we need to keep on adding information until we reach a strong enough invariant. Clearly, the information that is missing regards the clocks \( x_1 \) and \( x_2 \), which measure the elapsed time since the last use of rod1 and rod2, respectively.

Reasoning on Fig. 1, we see that we can add more properties to the state chart in Fig. 3., properties that can provide us with enough information for proving that state3 doesn’t belong to the reachable states of the system.

These properties are:

\[ (\forall t \in [\text{ start, now}) \bullet \text{ state}.t = 0 \Rightarrow ((x_1.t = t - \text{ start} \land x_2.t \geq \tau_1 + t - \text{ start}) \]
\[\lor (x_2.t = t - \text{ start} \land x_1.t \geq \tau_1 + t - \text{ start})) \]
\[\land (\text{ now} - \text{ start} = \tau_1)) \]
\[\land \text{ state}.t = 1 \Rightarrow (x_2.t \geq \tau_1 + t - \text{ start}) \land (\text{ now} - \text{ start} = \tau_1) \]
\[\land \text{ state}.t = 2 \Rightarrow (x_1.t \geq \tau_1 + t - \text{ start}) \land (\text{ now} - \text{ start} = \tau_1) \]
\[\land \text{ state}.t = 3 \Rightarrow \text{ false} \]
Beside the added properties regarding \( x_1 \) and \( x_2 \), we added some information regarding the time interval in between the initial moment (denoted by \( \text{start} \)) when the system enters a state and the final moment (denoted by \( \text{now} \)) of evolution of the system in the same state, moment that enables a transition to another (reachable) state. These properties also need to be proved, but we are skipping those proofs here.

We add all these new properties to the previous state chart, thus getting the new state chart in Fig. 4.

\[
\begin{align*}
0 & \quad \frac{d\theta}{dt} = v_1 \\
& \quad \frac{dx_1}{dt} = 1 \\
& \quad \frac{dx_2}{dt} = 1 \\
\text{now} - \text{start} = \tau_1 \\
\forall t \in [\text{start}, \text{now}) & \quad \theta.t \leq \theta_M \\
& \lor (x_1.t = t - \text{start} \land x_2.t \geq \tau_r + \tau_1 + t - \text{start}) \\
& \lor (x_1.t = t - \text{start} \land x_2.t \geq \tau_r + \tau_2 + t - \text{start}) \\
\theta = \theta_n \\
\text{false} \\
\text{now} - \text{start} = \tau_2 \\
\forall t \in [\text{start}, \text{now}) & \quad x_1 := 0 \\
\theta = \theta_M \land x_1 \geq T \\
\theta = \theta_M \land x_2 \geq T \\
\theta = \theta_M \land x_2 \geq \tau_1 + 1 - \text{start} \\
\theta = \theta_M \land x_2 \geq T \\
\text{now} - \text{start} = \tau_1 \\
\forall t \in [\text{start}, \text{now}) & \quad x_1 := 0 \\
\theta = \theta_n \\
\theta = \theta_M \land x_1 < T \land x_2 < T \\
\theta = \theta_M \land x_2 < T \\
\theta = \theta_M \land x_1 > T \land x_2 > T \\
\theta = \theta_M \land x_1 < \tau_r + \tau_2 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_1 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_2 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_1 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_2 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_1 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_2 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_1 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_2 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_1 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_2 + t - \text{start} \\
\theta = \theta_M \land x_1 < \tau_r + \tau_1 + t - \text{start} \\
\text{false} \\
\end{align*}
\]

Fig. 4. The TCS State chart with added properties about \( x_1 \), \( x_2 \)

It follows that the final invariant will be \( I' \) together with condition (11). One needs to choose the values of \( T_1 \) and \( T_2 \) so that this invariant will hold right from the start. This means that we can have either \( (T_1 = \tau_r + \tau_1 \text{ and } T_2 = 0) \) or \( (T_2 = \tau_r + \tau_1 \text{ and } T_1 = 0) \). Even without this choice, the invariant will hold after both rods have been used.

As an example, we are going to prove that condition (11) holds for releasing rod1, i.e., after the system has taken a discrete transition from \( \text{state}_1 \) to \( \text{state}_0 \), thus resetting \( x_1 \). The proofs for the initialization statement and for cooling with rod1 or rod2 are simpler, thus we are not presenting them in this paper.

We assume that the following properties hold:

\[
\begin{align*}
I' & \quad \land (\forall t \in [\text{start}, \text{now}) \land \text{state.start} = 0 \Rightarrow ((x_1,t = t - \text{start} \land x_2,t \geq \tau_r + \tau_1 + t - \text{start}) \\
& \lor (x_2,t = t - \text{start} \land x_1,t \geq \tau_r + \tau_2 + t - \text{start}) \\
& \land (\text{now} - \text{start} = \tau_1)) \\
& \lor (\text{state.start} = 1 \Rightarrow (x_2,t \geq \tau_r + t - \text{start}) \land (\text{now} - \text{start} = \tau_1))
\end{align*}
\]
∧ state.start = 2 => (x₁,t ≥ τ₁ + t - start) ∧ (now − start = τ₂)
∧ state.start = 3 => false
∧ state.now = 1 ∧ θ.now = θₚₙ
∧ c' = c \ now / (λt · t - now)
∧ x₁' = x₁ / now / (λt · t - now)
∧ θ' = θ / now / (λt · θₚₙ + v₁* c.t)
∧ state' = state / now / (λt · 0)
∧ start' = now
∧ now' = min {t' ≥ now | gg.t'}

We need to prove that the new information holds after these assignments. We have that

(∀ t ∈ [start', now') • state'.start' = 0 => ((x₁'.t = t - start' ∧ x₂.t ≥ τ₂ + t - start')
   ∨ (x₂.t = t - start' ∧ x₁'.t ≥ τ₁ + t - start'))

∧ state'.start' = 1 => (x₂.t ≥ τ₂ + t - start')
∧ state'.start' = 2 => (x₁'.t ≥ τ₁ + t - start')
∧ state'.start' = 3 => false)
≡ {start' = now, state'.now = 0}
(∀ t ∈ {now, now') • (x₁'.t = t - now ∧ x₂.t ≥ τ₂ + t - now)
   ∨ (x₂.t = t - now ∧ x₁'.t ≥ τ₁ + t - now))
≡ {substituting for x₁'.t = t - now}
(∀ t ∈ {now, now') • x₂.t ≥ τ₂ + t - now)
≡ {state.start = 1, so x₂.t ≥ (τ₂ + t - start) in [start, now), so x₂.now ≥ (τ₂ + now − start) =>
   x₂.now ≥ τ₂ + t₁, dx₂/dt = 1 in [now, now')}
(∀ t ∈ {now, now') • (x₂.t ≥ τ₂ + t₁ + (t - now))
true

Therefore, the invariant holds after the system has entered state0 from state1.
Action 4 (release rod2) is symmetric to the second action, so the invariant holds after taking the transition from state2 to state0, following the same proof rule.

What is left is to prove that these properties mean that the last action is never enabled, i.e.,
that

I ∧ 飏₅ = false

As state1 and state2 are safe states, condition (12) reduces to proving that:

(∀ t ∈ [start, now) • state.t = 0 => (((x₁,t = t - start ∧ x₂.t ≥ τ₂ + t - start)
   ∨ (x₂.t = t - start ∧ x₁,t ≥ τ₁ + t - start))
∧ (now − start = τ₁))
∧ (state.now = 0 ∧ θ.now = θₚₙ ∧ x₁.now < T ∧ x₂.now < T)
≡ { (now − start = τ₁ in state0) ∧ (x₂.now ≥ τ₂ + t₁ + (now − start) = 2τ₂ + t₁) ∨ (x₂.now ≥ τ₂ + t₂ + (now − start) = 2τ₂ + t₂ + (now − start) = 2τ₂ + t₂) ∧ (assumption (7): 2τ₂ + t₁ + T ∧ 2τ₂ + t₂ + T) => (x₁.now ≥ T ∨ x₂.now ≥ T)}
false
Therefore, conditions (8a,b) are met, implying that, under the $2\tau_r + \tau_1 \geq T \land 2\tau_r + \tau_2 \geq T$ assumption, the undesired shutdown state is not reachable.

6 Discussion

Following the discussion section in [10], we attempt to evaluate the formal framework we have used for modeling and verifying the safety-critical system studied here.

Are the formal descriptions easy to understand? The continuous action system model [3] offers an intuitive representation of a hybrid system, resembling an implementation of the system in a programming language. The requirements specifications look natural enough, when expressed as guarded statements, thus making the behavior of the system easily understandable. When shifting to the representation of the system where time is explicitly advanced, one does not have to rewrite the specifications, but just replace the variables with implicit time with the same variables with explicit time. This translated representation gives the semantics of the system. We used traditional forward analysis technique for verification purposes, rather than backward (weakest precondition) analysis, as the former can be easier to follow. We are not concerned here with termination properties but just with proving a safety property.

How hard is it to construct a proof using this method? Forward analysis as the verification method used in this paper, though not difficult, required some work. The hardest part was finding the right invariant for proving the safety property. Even though model-checking lets practitioners check automatically whether a given model of the system satisfies certain properties, it is not that powerful used alone, as it only verifies whether a subfamily of solutions satisfy those properties of interest. Carrying out a hand proof requires the ability to do formal proofs, but on the other hand the kind of proofs developed with the method used in this paper are fit for mechanical proof checking. In practice, interactive theorem provers (like HOL) are needed for automated support.

Does the proof yield information other than just the fact that the model of the system is correct? Yes. The invariant and the forward analysis itself as the verification method, even though require considerable effort, provide useful insight about the behavior of the system. Our approach allows references to historical values of the attributes in guards and expressions. In contrast, model-checking methods provide only an assertion that the implementation satisfies the desired properties [10].

Does the formalism scale up to handle larger systems? This is the big question. Quite a lot of automated support is needed for the method to be practical. Future work includes case studies done on larger systems. The complexity of the system adds complexity to the invariants. Parallel composition techniques can help in analyzing a larger system and further decomposition of the problem can simplify finding the proper invariant.

How easy is it to modify the model of the system and the proofs? Changing the specifications implies changing the proofs. Using an interactive theorem prover might help in uncovering the parts that need to be changed by rerunning the proofs quickly.
7 Conclusions and Related Work

Modeling and verification of hybrid systems require a rigorous formal framework, with proof techniques that are able to handle both discrete computing and continuous behavior. For more information about other formalisms developed for real-time systems, the reader is referred to [9].

In this paper we applied the generalized hybrid action system model to formalize and verify a hybrid system for temperature control, giving a formal proof for a safety property by using the same verification techniques as for ordinary action systems. Our approach, the hybrid action systems model [3], offers a homogenous and intuitive representation of a hybrid system, resembling an implementation of the system in a programming language. This makes the system’s behavior easy to understand, in spite of its hybrid nature that might involve complex dynamics.

The way of reasoning presented in this paper starts from the continuous action system model and its translation into an ordinary action system. Then it continues with extracting a first approximation of the invariant from the state chart representation of the system (or its hybrid automaton) and afterwards it progressively adds properties for strengthening the invariant in order to ensure the system’s safety. The verification method used is the classical forward analysis technique. Thus, the approach used for this case study lets us reason about both linear and nonlinear functions, without separating the discrete events from the continuous laws.

Reachability analysis for hybrid systems represents one of the most important and difficult problems to handle. In [1, 2], Alur, Henzinger et al. apply algorithmic analysis techniques using hybrid automata, techniques based on constructing the reachable region of linear hybrid systems, providing decidability and undecidability results for classes of linear hybrid systems (see also [13]). For general hybrid systems, the hybrid automata analysis methods can be applied with certain limitations. In [1], they perform symbolic model-checking for timed automata (introduced in [11]), illustrated on the temperature control system, using KRONOS to compute the characteristic set of state predicates, therefore using different values for the parameters. In this paper we give a general mathematical proof. As emphasized in section 6, the proof technique used in this paper, i.e, traditional forward analysis, is amenable to mechanical proof checking. In contrast to the model-checking technique used in [1], which provides only an assertion that the model of the temperature control system satisfies a safety property, our approach, i.e., proving an invariance property, offers useful key insights about the behavior of the system. In practice, this feature can contribute to the design stage of a new system, as adding information to the system’s states might suggest adding design details.

In cases when the reachability construction fails, the reachability verification method is applied [12]. The user has first to guess (heuristically) the reachable region and then verify that the guess is correct. The method is almost fully automated (there are no automated guess heuristics), but in case that the guessed region is not directly inductive, new variables and constraints have to be added, making the method more complicated.

Our approach, the generalized action system formalism [3] is suited for modeling and verification of mission-critical systems, as it allows for the explicit failure of the system (modeled by the “abort” statement) and also allows references to historical values of the attributes in guards and expressions. These make our formalism more expressive than hybrid automata [1].

The hybrid constraint (language) approach (Hybrid cc) [8] requires the user to be able to express as constraints various aspects of the given hybrid automaton. This means that a
constraint system is needed that is expressive enough, a requirement sometimes difficult to satisfy.

The verification methodology based on abstracted automata, developed by Puri and Varaiya [14], which has simpler dynamics, faces the inconvenience that the abstractions that are created depend on the property to be proved. Different specifications may require different abstractions, so proving different properties of the same hybrid system implies creating different abstractions.

The strong point of our approach is that it allows almost any type of function in the dynamic laws characterizing the continuous behavior, compared to Rönkkö’s and Li’s linear hybrid action systems [15], where only smooth functions (without discontinuities) can be handled. Due to atomicity, the kind of action systems that were considered in [15] cannot model hybrid systems where continuous steps have nondeterministic ending time. In the presented approach, by defining the hybrid action system as a collection of piecewise time functions, we are allowed to also reason about functions with nondeterministic ending time. The approach introduced in [15] doesn’t have an implicit notion of time and it is not intended to model real-time systems, whereas our model facilitates the description of real-time systems.

Future work involves looking at refinement of hybrid systems, based on the refinement calculus techniques [5], and analyzing hybrid action systems with interactive control.

Acknowledgements

The authors would like to thank Luigia Petre, Ivan Porres Paltor and Mauno Rönkkö for their valuable comments on this case study.

References


In Proof of (10a)

\begin{align*}
gg . t' & \equiv (\lambda t \cdot \theta t = 0) \land (\theta t = \theta M) \land (x_1.t' \geq T).t' \\
(\lambda t \cdot \theta t = 1) \land (\theta t = \theta m) . t' \lor \\
(\lambda t \cdot \theta t = 0) \land (\theta t = \theta M) \land (x_2.t' \geq T).t' \lor \\
(\lambda t \cdot \theta t = 2) \land (\theta t = \theta m) . t' \lor \\
(\lambda t \cdot \theta t = 0) \land (\theta t = \theta M) \land (x_1.t' < T \land x_2.t' < T).t'
\end{align*}

In Proof of (10b)

\begin{align*}
gg [\theta : = (\lambda t \cdot \theta_0 + v_r . t)].t' & \equiv \\
(\lambda t \cdot \theta_0 + v_r . t) \land (\theta_0 + v_r . t) . t' = \theta M \land (x_1.t' \geq T) \lor \\
(\lambda t \cdot \theta_0 + v_r . t) = \theta m \lor \\
(\lambda t \cdot \theta_0 + v_r . t) = \theta M \land (x_2.t' \geq T) \lor \\
(\lambda t \cdot \theta_0 + v_r . t) = \theta m \lor \\
(\lambda t \cdot \theta_0 + v_r . t) = \theta M \land (x_1.t' < T \land x_2.t' < T) \lor \\
\{ \lambda \text{reduction} \} \\
(0 = 0 \land \theta_0 + v_r . t' = \theta M \land x_1.t' \geq T) \lor \\
(0 = 1 \land \theta_0 + v_r . t' = \theta m) \lor \\
(0 = 0 \land \theta_0 + v_r . t' = \theta M \land x_2.t' \geq T) \lor \\
(0 = 2 \land \theta_0 + v_r . t' = \theta m) \lor \\
(0 = 0 \land \theta_0 + v_r . t' = \theta M \land x_1.t' < T \land x_2.t' < T) \lor \\
\{ \text{logic} \} \\
(0 = 0 \land \theta_0 + v_r . t' = \theta M) \\
\{ \text{logic} \} \\
\theta_0 + v_r . t' = \theta M
\end{align*}
\[(1 = 0 \land \theta.t' = \theta_M \land x_2.t' \geq T) \lor \]
\[(1 = 2 \land \theta.t' = \theta_m) \lor \]
\[(1 = 0 \land \theta.t' = \theta_M \land x_1.t' < T \land x_2.t' < T) \equiv \{ \text{logic} \}
\]
\[\theta.t' = \theta_m\]

*In Proof of (11)*

\[\text{gg [state : = (} \lambda t \cdot 0\text{),} t'\]
\[\equiv \{ \text{similarly to (10b)} \}
\]
\[(0 = 0 \land \theta.t' = \theta_M \land x_1.t' \geq T) \lor \]
\[(0 = 1 \land \theta.t' = \theta_m) \lor \]
\[(0 = 0 \land \theta.t' = \theta_M \land x_2.t' \geq T) \lor \]
\[(0 = 2 \land \theta.t' = \theta_m) \lor \]
\[(0 = 0 \land \theta.t' = \theta_M \land x_1.t' < T \land x_2.t' < T) \equiv \{ \text{logic} \}
\]
\[\theta.t' = \theta_M\]