Verification of Linear Hybrid Action Systems

Mauno Rönkkö

Turku Centre for Computer Science
Department of Computer Science, Åbo Akademi University
Lemminkäisenkatu 14A, FIN-20520 Turku, Finland
`mronkko@abo.fi`

Xuandong Li

Turku Centre for Computer Science
Department of Computer Science, Åbo Akademi University
Lemminkäisenkatu 14A, FIN-20520 Turku, Finland
`xuandong.li@abo.fi`

current address:
Department of Computer Science and Technology
Nanjing University, Nanjing
Jiangsu, P.R.China 210093
`lxd@nju.edu.cn`

Turku Centre for Computer Science
TUCS Technical Report No 296
9th August 1999
ISBN 952-12-0507-5
ISSN 1239-1891

Abstract

Action Systems is a predicate transformer based formalism for the development of reactive and distributed systems by refinement. Lately, Action Systems was extended with a differential action that models continuous behaviour. Such action systems capture hybrid systems, where a discrete controller interacts with some continuously evolving environment.

Recently, we investigated a class of Action Systems called Linear Hybrid Action Systems that capture linear hybrid systems. These action systems can formally be translated to linear hybrid automata, for which there exists tools for automated analysis, e.g., HyTech. However, since Linear Hybrid Action Systems is a strict subclass of linear hybrid automata, the translation necessarily introduces more complexity to the model making the analysis harder. Therefore, the contribution of this paper is to propose two efficient complementary semi-decision algorithms for the analysis of Linear Hybrid Action Systems, an invariant verification algorithm of linear complexity and a reachability algorithm of squared complexity.

Keywords: action systems, linear hybrid systems, model-checking, invariants

TUCS Research Group
Programming Methodology Research Group

1 Introduction

Action systems [3, 4, 5] are a predicate transformer based systems used especially in the development of reactive and concurrent systems. Recently, Rönkkö and Ravn [15] extended Action Systems with a new action called the *differential action*. It allows the use of differential equations in capturing continuous behaviour. The differential action has a predicate transformer semantics like the other actions, and therefore, it fits seamlessly into the Action Systems framework. Action Systems with differential actions model hybrid systems where a discrete controller interacts with some continuously evolving environment.

In the development of Action Systems the use of refinement plays a central role. The differential action makes no exception here. As shown by Rönkkö and Sere [16], there are simple refinement laws for the differential action allowing efficient development of provably correct hybrid systems.

However, the use of refinement does not remove the need for automated analysis. One is often interested in analysing some special properties that are conveniently captured by reachability analysis. Other formalisms, e.g., *linear hybrid automata*, provide automated support for this type of analysis. Motivated by this, we studied in a previous paper a class of Action Systems called *Linear Hybrid Action Systems* that model *linear hybrid systems* [14]. A linear hybrid system can be described by using only *linear expressions*, i.e., linear combination of variables with constant coefficients. We showed that there exists a simple translation function allowing formal translation of a linear hybrid action system to a linear hybrid automaton as defined, e.g., by Henzinger et al. [10]. We also showed that Linear Hybrid Action Systems is a strict subclass of linear hybrid automata. However, this is a drawback when considering the analysis, because it means that the translation necessarily introduces some additional complexity to the model. This, in turn, makes the analysis harder.

Therefore, the contribution of this paper is to propose two complementary algorithms for the analysis of Linear Hybrid Action Systems. The first algorithm that we present is an invariant verification algorithm that can be utilized, e.g., with a theorem prover such as PVS [8]. This algorithm is of linear complexity over the number of actions in a linear hybrid action system. Note that the number of actions in a linear hybrid action system corresponds to the number locations in a linear hybrid automaton [14]. The second algorithm is a reachability algorithm with a squared complexity over the number of actions in a linear hybrid action system. This algorithm is a depth-first search algorithm similar to the ones used in model-checking. Due to the predicate transformer semantics model-checking is not often used with action systems, although Back and von Wright have given a trace semantics for Action Systems [6]. Both of these algorithms are semi-decision algorithms. In other words, neither of them is guaranteed to give a result concerning an arbitrary property of any linear hybrid action system. However, due to their difference in approach these two algorithms are complementary. We shall characterize in the conclusion the properties that guide the selection of the algorithm to be used.

There has not been any work done on the algorithmic analysis of (linear) hybrid action systems before. As mentioned, this issue has been well investigated in other formalism. A good overview can be found in the proceedings of hybrid systems published as Lecture Notes in Computer Science by Springer-Verlag, volumes 736, 999, 1066, 1201, 1273, 1386, 1567, and 1569. Due to this good theoretical foundation there exists also many tools supporting the analysis of real-time and hybrid systems, e.g., HyTech[11] that is used for linear hybrid automata, and UppAll[7] that is used for timed automata. Theorem provers have also been used for verifying real-time and hybrid systems, e.g., by Vitt and Hooman [17].

Overview.

We start by defining actions in Section 2. In Section 3 we present Linear Hybrid Action Systems that model linear hybrid systems. In Section 4 we discuss invariance proofs and present an algorithm for verifying invariance properties of Linear Hybrid Action Systems. In Section 5 we consider algorithmic analysis of Linear Hybrid Action Systems and present an algorithm for reachability analysis. In this section we also discuss the use of the two given algorithms. Finally, follows the conclusion in Section 6.

2 Actions

An action is any statement in Dijkstra's guarded command language [9]. Also, a pure guarded command can be used as an action. Actions operate on a fix set of program variables, and therefore, predicates over these variables specify states in a system.

When we consider a given action A, we may speak of a *postcondition* q. It is a predicate specifying desirable states after the execution of an action. The meaning of an action is defined with the *weakest precondition* predicate transformer $\mathrm{wp}(A, q)$. It is a predicate describing the largest set of states from which the action A terminates and reaches the postcondition q. The *termination*, i.e., reaching an arbitrary state, is given by $\mathrm{t}(A) \ \hat{=}\ \mathrm{wp}(A, \mathit{true})$. An action A is said to *abort* in all the states where $\neg\mathrm{t}(A)$ holds. We may also speak of enabledness of an action, or *guard condition*, i.e., $\mathrm{g}(A) \ \hat{=}\ \neg\mathrm{wp}(A, \mathit{false})$. An action A is said to be enabled in all those states where $\mathrm{g}(A)$ holds.

Let $q[E/X]$ denote the textual substitution of free variables X with expressions E in a predicate q. Also, let e and p be predicates, A and B be actions, and τ and δ be real variables and not appearing in X, e, p, or q. The actions that we consider in this paper are defined as follows.

$$
\begin{array}{lll}
\mathrm{wp}(\mathbf{skip}, q) & \mathrel{\widehat{=}} q & (\textit{skip}) \\
\mathrm{wp}(X := E, q) & \mathrel{\widehat{=}} q[E/X] & (\textit{assignment}) \\
\mathrm{wp}(A; B, q) & \mathrel{\widehat{=}} \mathrm{wp}(A, \mathrm{wp}(B, q)) & (\textit{seq. composition}) \\
\mathrm{wp}(A \mathbin{[\!]} B, q) & \mathrel{\widehat{=}} \mathrm{wp}(A, q) \wedge \mathrm{wp}(B, q) & (\textit{non-det. choice}) \\
\mathrm{wp}(p \to A, q) & \mathrel{\widehat{=}} p \Rightarrow \mathrm{wp}(A, q) & (\textit{guarded command}) \\
\mathrm{wp}(\mathbf{do}\, A\, \mathbf{od}, q) & \mathrel{\widehat{=}} \exists\, n : n = \min\{i \mid i \geq 0 \wedge \neg \mathrm{g}(A^{i+1})\} & (\textit{iteration}) \\
 & \quad .\, \mathrm{wp}(A^n, q) & \\
\mathrm{wp}(e : \dot{X} = F(X), q) & \mathrel{\widehat{=}} \exists!\, \Phi : (\Phi(0) = X \wedge \dot{\Phi} = F(\Phi)) & (\textit{differential action}) \\
 & \quad .\, \exists\, \delta : \delta = \inf\{\tau \mid \tau \geq 0 \wedge \neg e[\Phi(\tau)/X]\} & \\
 & \quad\quad .\, q[\Phi(\delta)/X] &
\end{array}
$$

where $A^0 \mathrel{\widehat{=}} \mathbf{skip}$ and $A^{n+1} \mathrel{\widehat{=}} A; A^n$. Informally, the above actions are described as follows. *Skip* models an action that does nothing. *Assignment* models an action that changes the values of variables X according to expressions E. *Sequential composition* models the execution of an action B after an action A. *Non-deterministic* choice models the non-deterministic selection and execution of either of the actions A or B. *Guarded command* models the execution of an action A when a guard p holds. *Iteration* models the repetitive execution of an action A as long as it is enabled. *Differential action* models continuous behaviour.

In the differential action e is a predicate over the variables X, and $\dot{X} = F(X)$ is a system of differential equations. A differential action, $e : \dot{X} = F(X)$, evolves continuously the values of X provided that the evolution guard e holds. The evolution, which is given by a system of differential equations, stops when the values of X have reached the border of e. In the definition for the differential action, the first existential quantification requires the existence of a unique solution Φ for $\dot{X} = F(X)$, which is an *autonomous* system of differential equations, i.e., it may not refer to time explicitly. Time is implicit and it is modelled by the variable τ. The second existential quantification requires that the evolution eventually reaches the border of the evolution guard e and that the reached values satisfy the postcondition q. The variable δ is used for denoting the time, when the border of e is reached. If e does not hold at the beginning of an evolution, δ becomes zero forcing the evolution not to start at all. Hence, the semantics resembles the semantics for an iteration given earlier. The differential action is deterministic in its ending time.

3 Linear Hybrid Action Systems

An important class of hybrid systems is the class of linear hybrid systems. In these systems all the continuous behaviour progresses at a constant rate. Linear hybrid systems can be described by using only linear expressions, i.e., linear combination of variables with constant coefficients. Such a class of hybrid systems is covered by, e.g., linear hybrid automata.

In the literature, there are more results for the linear hybrid systems than for the non-linear hybrid systems. For instance, the verification of hybrid automata in general is difficult, but there are some results for the analysis and verification of linear hybrid automata, see Alur et al.[1] and Li et al. [18]. Moreover, a non-linear hybrid system can be approximated with a linear hybrid system [11].

The subclass of Action Systems modelling linear hybrid systems is called Linear Hybrid Action Systems. A linear hybrid action system $\mathcal{A}$ is an initialized block [14]

$$\mathcal{A} \;\widehat{=}\; |[\,\mathbf{var}\ X : T \;\bullet\; X := E_0;\ \mathbf{do}\, A_1 \,[\!]\, .. \,[\!]\, A_n\ \mathbf{od}\,]| \;\;,$$

where each A_i is of form $p_i \rightarrow X := E_i;\ e_i : \dot{X} = C_i$, such that all the expressions in p_i, E_i, and e_i are linear expressions. Moreover, e_i is a predicate describing a group of closed intervals, $p_i \Rightarrow e_i[E_i/X]$ holds, and C_i is a vector of constants.

The expression $\mathbf{var}\ X : T$ declares a list of variables X with types T. These variables form the *state space* of $\mathcal{A}$. The action $X := E_0$ initializes all the variables X by expressions E_0. These expressions are constant expressions and may not refer to any variable, because the values of the variables are undefined before the initialization.

The number of actions A_i in $\mathcal{A}$ is finite and countable, i.e., $i \in 1..n$. As already mentioned, we assume that each action A_i is of form $p_i \rightarrow X : = E_i;\ e_i : \dot{X} = F_i(X)$, such that $p_i \Rightarrow e_i[E_i/X]$ holds. In other words, each action is a guarded command, where a differential action follows an assignment statement, and the evolution guard of the differential action holds always after the execution of the preceding assignment statement. An arbitrary action system is rewritten into this form by using refinement [2, 16]. An enabled action A_i is selected non-deterministically for execution after the initialization. There are no fairness assumptions about the selection of actions. The execution of an action A_i is always atomic. This means that also the continuous evolution of the differential action in A_i proceeds without interruption up to the border of its evolution guard. The reached values of the variables form a new state, that is called the *successive state*. A successive state is always deterministic. Therefore, for an action A_i and the initial values of the variables X the execution time of A_i can be calculated as the time that it takes for the continuous evolution Φ to reach the border of the evolution guard. The execution time is returned by the expression $\delta = \inf\{\tau \mid \tau \geq 0 \wedge \neg e[\Phi(\tau)/X]\}$. Note that an action with no termination for the continuous evolution does not have a successive state. Only after a successful execution of one action, the other enabled actions have a chance for execution. If two enabled actions refer to disjoint variables, their execution can be in any order. Hence, this models parallelism by interleaving.

The execution of $\mathcal{A}$ terminates when none of the actions is enabled, i.e., $\neg\,\mathrm{g}(A_1 \,[\!]\, .. \,[\!]\, A_n)$. Likewise, an action system aborts when an executed action A_i aborts, i.e., $\neg\mathrm{t}(A_i)$ holds.

Example 1. As an example of a linear hybrid action system we use a water-level monitor example that has been presented in several papers before, e.g., by Alur et al. [1] and Li et al. [18]. In this example, the task of the water-level monitor is to keep the water level of a tank within a certain interval. The monitor can either pump water into the tank or out of the tank. It takes two seconds for the monitor to switch the pumping direction. When the water is being pumped into the tank, the water level rises at the rate of 1 inch per second, and when the water is being pumped out of the tank, the water level falls at the rate of two inches per second. There are two sensors in the tank for sensing when the water level is at five or ten inches. When the water level reaches the sensor at five inches the monitor switches from pumping water out from the tank to pumping water into the tank. Likewise, when the water reaches the sensor at ten inches the monitor switches from pumping water into the tank to pumping water out from the tank. Figure 1 depicts the characteristics of the water tank.

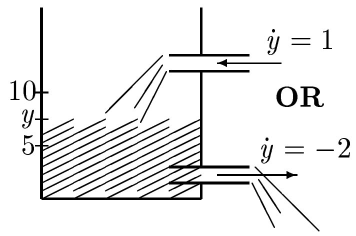

Figure 1: The water tank with threshold water levels and flow rate effects on the water level in inches per second.

The linear hybrid action system below is a specification of the water-level monitor.

$$
\begin{array}{lll}
\mathit{Monitor} & \widehat{=} & |[\ \mathbf{var}\ x, y : I\!R, I\!R \ \bullet\ x, y := 2, 1; \\
 & & \quad \mathbf{do}\ x = 2 \wedge y \leq 5 \ \rightarrow x, y := x, y;\ \ y \leq 10 : \dot{x}, \dot{y} = 1, 1 \\
 & & \quad\ \ [\!]\ \ y = 10 \qquad\qquad \rightarrow x, y := 0, y;\ \ x \leq 2 : \dot{x}, \dot{y} = 1, 1 \\
 & & \quad\ \ [\!]\ \ x = 2 \wedge y \geq 10 \rightarrow x, y := x, y;\ \ y \geq 5 : \dot{x}, \dot{y} = 1, -2 \\
 & & \quad\ \ [\!]\ \ y = 5 \qquad\qquad\ \ \rightarrow x, y := 0, y;\ \ x \leq 2 : \dot{x}, \dot{y} = 1, -2 \\
 & & \quad \mathbf{od} \\
 & &]|
\end{array}
$$

There are two variables, x and y, in $\mathit{Monitor}$. The actual water level is denoted by y, and x is a clock variable used for modelling the two second delay associated with the switching of the pumping direction. Initially, $x=2$ and the water level is at one inch, $y=1$.

There are four actions in $\mathcal{M}onitor$. The first action models the filling of the tank. $\mathcal{M}onitor$ may enter this phase, whenever the guard $x=2 \wedge y\leq 5$ holds. This happens, in particular, after the initialization. The reset action $x, y := x, y$ after the guard states that the values of the variables do not changed when the monitor decides to start filling the tank. According to the differential action, $y\leq 10 : \dot{x}, \dot{y}=1, 1$, the filling proceeds till the water level y in the tank reaches the sensor located at ten inches. Also, the clock variable x advances during the filling.

The second action models the two second delay when the monitor switches from pumping water into the tank to pumping water out of the tank. In order to measure the two second delay, the reset action sets the clock variable x to zero. Note that during the delay, the water level still rises at the rate of one inch per second.

Similarly to the first action, the third action models the phase when the water is being pumped out of tank. During this phase the water level falls at the rate of two inches per second. This phase ends when the water level y in the tank reaches the sensor located at five inches.

The fourth action models the two second delay when the monitor switches from pumping water out of the tank to pumping water into the tank. During this delay, the water level still falls at the rate of two inches per second.
End of example

Given a specification of a linear hybrid system, such as the specification for the water-level monitor above, one often wishes to verify some safety properties of that system. For instance, in the case of the water-level monitor, we would like to show that the water tank does not overflow. Such properties are often expressed as invariance properties. The task is then to show that the invariance properties form an invariant of a specification, i.e., the specified system remains always at states governed by the invariant. In the following we investigate how to prove safety properties of linear hybrid action systems as invariance properties by using the weakest precondition semantics of actions.

4 Invariance Proofs for Linear Hybrid Action Systems

As mentioned earlier, actions operate on a fix set of program variables. Therefore, a state in the system is solely identified by the values of the variables. This allows the use of predicates to specify states in a system. A predicate that holds at any state in the system is called an *invariant*. Consider an action A and a predicate inv. If from any state where inv holds the execution of A leads always to a state where inv still holds, the predicate *inv* is said to be the invariant for the action A. Formally, this is stated as a condition $inv \Rightarrow \mathrm{wp}(A,\ inv)$ whenever A is an *atomic* action, i.e., an action executed in one step.

When considering the invariant for an iteration, the condition given above is not sufficient. For instance, the predicate $x = 0 \vee x = 2$ is not an invariant for an iteration $\mathbf{do}\ x<2 \rightarrow x:=x+1\ \mathbf{od}$, although

$$x=0 \vee x=2 \Rightarrow \mathrm{wp}(\ \mathbf{do}\ x<2 \rightarrow x:=x+1\ \mathbf{od},\ x=0 \vee x=2)$$

holds. Clearly, the iteration above traverses through a non-invariant state, $x=1$, when starting from an invariant state, $x=0$. The reason why the given invariance condition does not work with the iteration is that the weakest precondition merges all the intermediate and final states to a single predicate, whereby all the causilities are lost. One way of avoiding this is to inspect if the atomic actions inside the iteration preserve the invariant by themselves. This method is formalized in the following theorem. For the formalization we need the concept of *monotonicity*. An action A is monotonic, if $\mathrm{wp}(A,p) \Rightarrow \mathrm{wp}(A,q)$ holds under the assumption that $p \Rightarrow q$ holds [9]. As shown elsewhere [9, 16] all the actions we consider in this paper are monotonic.

Theorem 1. Let $\mathbf{do}\ A_1 [\!] .. [\!] A_n\ \mathbf{od}$ be an iteration where each action A_i, $i \in 1..n$, is atomic and monotonic. Also, let *inv* be a predicate which is an invariant for every action, i.e. $\forall\, i : 1 \leq i \leq n\ .\ inv \Rightarrow \mathrm{wp}(A_i,\ inv)$. Then, an execution of the iteration starting from a state satisfying the invariant traverses only the states satisfying the invariant, i.e., $\forall\, m : 0 \leq m\ .\ inv \Rightarrow \mathrm{wp}((A_1 [\!] .. [\!] A_n)^m,\ inv)$.

Proof. We prove the claim by induction. The proof for the case where $m=0$ is trivial, since $inv \Rightarrow \mathrm{wp}(\mathbf{skip},\ inv)$ holds trivially. In the case of $m=1$, the predicate $inv \Rightarrow \mathrm{wp}(A_1 [\!] .. [\!] A_n,\ inv)$ holds as well, because it unfolds into the assumption $\forall\, i : 1 \leq i \leq n\ .\ inv \Rightarrow \mathrm{wp}(A_i,\ inv)$, which holds. The induction assumption is that $inv \Rightarrow \mathrm{wp}((A_1 [\!] .. [\!] A_n)^j,\ inv)$ holds for some $j \geq 0$. The proof for the induction step goes as follows

$$\begin{array}{ll}
 & inv \Rightarrow \mathrm{wp}((A_1 [\!] .. [\!] A_n)^{j+1},\ inv) \\
\Leftrightarrow & \{\textit{unfolding}\} \\
 & \quad inv \Rightarrow \mathrm{wp}((A_1 [\!] .. [\!] A_n)^j; (A_1 [\!] .. [\!] A_n),\ inv) \\
\Leftrightarrow & \{\textit{sequential composition}\} \\
 & \quad inv \Rightarrow \mathrm{wp}(\ (A_1 [\!] .. [\!] A_n)^j,\ \mathrm{wp}((A_1 [\!] .. [\!] A_n),\ inv)\ ) \\
\Leftarrow & \{\textit{monotonicity and } inv \Rightarrow \mathrm{wp}(A_1 [\!] .. [\!] A_n,\ inv)\} \\
 & \quad inv \Rightarrow \mathrm{wp}((A_1 [\!] .. [\!] A_n)^j,\ inv) \\
\Leftarrow & \{\textit{induction assumption}\} \\
 & \quad \textit{true}
\end{array}$$

Therefore, the invariant holds also for the induction step, and the claim holds by induction. □

Note that Theorem 1 provides only a sufficient requirement, not a necessary one. For instance, consider an invariant $x=1 \vee y=1$, and a sequential composition of an action and an iteration, $x, y := 1, 1;$ **do** $x=1 \rightarrow x:=0$ **od**. Despite that $x=1 \vee y=1 \Rightarrow \text{wp}(x=1 \rightarrow x:=0,\ x=1 \vee y=1)$ does not hold, the predicate $x=1 \vee y=1$ is an invariant. The reason for this is that the guard $x=1$ in the iteration is not as strong as it could be. By strengthening the guard to $x=1 \wedge y=1$, which does not change the behaviour of the iteration, one can show by using Theorem 1 that the predicate $x=1 \vee y=1$ is indeed an invariant. In this sense, Theorem 1 is general enough to be useful for proving invariance properties for iterations.

As it became apparent from the example above, Theorem 1 does not assume anything about the termination. These two properties, termination and invariance, are often considered separately.

4.1 Continuous invariant

Analogously to the iteration, the simple invariance condition given at the beginning of this section is not sufficient for the differential action. For instance, the predicate $x = 0 \vee x = 2$ is not an invariant for a differential action $0 \leq x \leq 2 : \dot{x} = 1$, although

$$x=0 \vee x=2 \Rightarrow \text{wp}(0 \leq x \leq 2 : \dot{x}=1,\ x=0 \vee x=2)$$

holds. A continuous evolution starting from the state $x=0$ traverses through infinitely many states, where $x = 0 \vee x = 2$ does not hold until it reaches the state $x = 2$. The reason why the invariance condition does not work with the differential action is similar to the reason for the iteration. The weakest precondition merges all the intermediate and final states to a single predicate, whereby all the causilities are lost. Therefore, we need to consider a stronger notion of invariance, *continuous invariant.*

Definition 1. Let *inv* be a predicate over variables X. Also, let $e : \dot{X} = F(X)$ be a differential action, and t be a variable not in X. Then, *inv* is a continuous invariant, if

$$\forall \delta : \delta \geq 0 \ .\ inv \Rightarrow \text{wp}(t{:=}0; e \wedge t \leq \delta : \dot{X}, \dot{t} = F(X), 1,\ inv)$$

holds.
End of definition

Definition 1 simply states that an invariant is a continuous invariant for a differential action, if any evolution of the differential action starting from a state satisfying the invariant remains at all times in the set of states satisfying the invariant. Thus, continuous invariance is a special form of *stability* as known in the theory of differential equations [12].

For the kind of the differential actions appearing in linear hybrid action systems, i.e., differential actions of form $e : \dot{X} = C$ where C is a vector of constants, there exists a simple condition for checking if a predicate is a continuous invariant. For this we need the concept of a *convex* set of states. A predicate p describes a convex set of states over variables X, if two states $X=U$ and $X=V$, such that $X=U \ \wedge \ X=V \ \Rightarrow \ p$, are connected with a line $\Phi(\tau) = \tau \cdot U + (1-\tau) \cdot V$ for which $\forall \tau : 0 \leq \tau \leq 1 \ . \ X = \Phi(\tau) \ \Rightarrow \ p$ holds.

Theorem 2. Let $e : \dot{X} = C$ be a differential action where C is a vector of constants. Also, let *inv* be a predicate describing a convex set of states over variables X, such that $inv \ \Rightarrow \ \mathrm{wp}(e : \dot{X} = C, \ inv)$. Then, *inv* is a continuous invariant.

Proof. By assumption $inv \ \Rightarrow \ \mathrm{wp}(e : \dot{X} = C, \ inv)$ holds. Therefore, there exists a solution curve $\Phi(\tau) = U + C \cdot \tau$, where U is an initial value for X when the evolution starts, i.e., $\Phi(0) = U$. Moreover, the evolution described by Φ terminates, i.e., there exists δ such that the predicate $\delta = \inf\{\tau \mid \tau \geq 0 \wedge \neg e[\Phi(\tau)/X]\}$ holds. We denote the reached values by V, i.e., $\Phi(\delta) = V$. By assumption, the predicate $X = U \ \wedge \ X = V \ \Rightarrow \ inv$ holds. Since Φ is a line connecting the two states $X = U$ and $X = V$, and *inv* describes a convex set of states, we know that the predicate $\forall \tau : 0 \leq \tau \leq \delta \ . \ X = \Phi(\tau) \ \Rightarrow \ inv$ holds as well. Therefore, *inv* holds at all states traversed by Φ. Since we did not limit the selection of $\Phi(0)$ and $\Phi(\delta)$, the evolution Φ can be any evolution for which $X = \Phi(0) \ \wedge \ X = \Phi(\delta) \ \Rightarrow \ inv$ holds. Therefore, any evolution starting from a state satisfying *inv* remains at all times within the states satisfying *inv*. Hence, *inv* is a continuous invariant. $\square$

Note that Theorem 2 provides only a sufficient condition for proving continuous invariance. For instance, an invariant describing two separate convex sets cannot be proven to be a continuous invariant for a differential action by using Theorem 2. However, due to the form of actions in linear hybrid action systems Theorem 2 is indeed general enough to be practical in proving continuous invariance properties for linear hybrid action systems.

4.2 Invariant for linear hybrid action system

In order to prove that a predicate is an invariant for a linear hybrid action system, one has to show that it holds at all states in the system. This means that the predicate must be an invariant not only for the discrete behaviour, but also for the continuous behaviour in the system. Following the idea of Theorem 1 we first consider when a predicate is an invariant for an individual action in a linear hybrid action system.

Lemma 1. Let C be a vector of constants, p, e, and *inv* be predicates over variables X, as well as, E be an expression over variables X. Moreover, all the expressions in e, p, and E are assumed to be linear expressions. Also, e describes a group of closed intervals, such that $inv \wedge e$ describes a convex set and $p \Rightarrow e[E/X]$ holds. Then, *inv* is both an invariant and a continuous invariant for an action $p \rightarrow X := E;\ e : \dot{X} = C$, if the conditions

$$\begin{array}{ll} 1) & inv \Rightarrow \mathrm{wp}(p \rightarrow X := E,\ inv), \ \ and \\ 2) & inv \Rightarrow \mathrm{wp}(p \rightarrow X := E;\ e : \dot{X} = C,\ inv \wedge e) \end{array}$$

hold.

Proof. As mentioned in the beginning of this section, the condition (1) ensures that *inv* is an invariant for the discrete change. Therefore, we only need to show that the condition (2) ensures that *inv* is also a continuous invariant for the continuous behaviour. First of all, consider the conjunction $inv \wedge e$. By assumption this conjunction describes a convex set. By Theorem 2 it suffices to show that $inv \wedge e \Rightarrow \mathrm{wp}(e : \dot{X} = C,\ inv \wedge e)$ holds to prove that $inv \wedge e$ is a continuous invariant for the continuous behaviour. Now we show that after the discrete change $inv \wedge e$ holds, i.e., $inv \Rightarrow \mathrm{wp}(p \rightarrow X := E,\ inv \wedge e)$ holds. By the assumption $p \Rightarrow e[E/X]$ we know that e holds after the discrete change, i.e., $\mathrm{wp}(p \rightarrow X := E, e)$ holds. Then, also the predicate $inv \Rightarrow \mathrm{wp}(p \rightarrow X := E, e)$ holds. Merging this logically with the condition (1), we get

$$inv \Rightarrow (\ \mathrm{wp}(p \rightarrow X := E, e) \wedge \mathrm{wp}(p \rightarrow X := E, inv)\ )$$

which in turn can be rewritten as

$$inv \Rightarrow p \Rightarrow (\ \mathrm{wp}(X := E, e) \wedge \mathrm{wp}(X := E, inv)\ ) \ \ .$$

Since the assignment statement is conjunctive [9], i.e., the predicate $\mathrm{wp}(X := E, e) \wedge \mathrm{wp}(X := E, inv) \Leftrightarrow \mathrm{wp}(X := E, e \wedge inv)$ holds, we can rewrite the expression above as

$$inv \Rightarrow \mathrm{wp}(p \rightarrow X := E, e \wedge inv) \ \ ,$$

and thus, we have shown that $e \wedge inv$ indeed holds after the discrete change. Therefore, proving the condition (2) proves also indirectly that $inv \wedge e \Rightarrow \mathrm{wp}(e : \dot{X} = C,\ inv \wedge e)$ holds. But then, $inv \wedge e$ is a continuous invariant for the continuous behaviour. Since we showed above that from any state satisfying *inv* the discrete change always leads to a state satisfying $inv \wedge e$, we know that any continuous behaviour of the action in question satisfies the predicate $inv \wedge e$, and thus, satisfies trivially also the predicate *inv*. Therefore, the predicate *inv* holds at all states, whereby *inv* is both an invariant and a continuous invariant. □

With the help of Lemma 1 we can formulate a theorem that is used for proving invariance properties of a linear hybrid action system.

Theorem 3. Let $\mathcal{A}$ be a linear hybrid action system, i.e.,

$$\mathcal{A} \;\hat{=}\; |[\, \mathbf{var}\ X : T \ \bullet\ X := E_0;\ \mathbf{do}\, A_1 \,[\!]\, .. \,[\!]\, A_n\, \mathbf{od}\,]| \ ,$$

where each A_i is of form $p_i \rightarrow X := E_i;\ e_i : \dot{X} = C_i$, such that $p_i \Rightarrow e_i[E_i/X]$ holds, all the expressions in p_i, E_i, and e_i are linear expressions. Moreover, e_i is a predicate describing a group of closed intervals, and C_i is a vector of constants. Let *inv* be a predicate, such that for any evolution guard e_i in $\mathcal{A}$ the conjunct $inv \wedge e_i$ describes a convex set. Then, *inv* is an invariant for $\mathcal{A}$ if the conditions

1) $\quad true \Rightarrow \mathrm{wp}(X := E_0,\ inv)$
2) $\quad \forall\, i : 1 \leq i \leq n.\ inv \Rightarrow \mathrm{wp}(p_i \rightarrow X := E_i,\ inv)$, and
3) $\quad \forall\, i : 1 \leq i \leq n.\ inv \Rightarrow \mathrm{wp}(p_i \rightarrow X := E_i;\ e_i : \dot{X} = C_i,\ inv \wedge e_i)$

hold.

Proof. The condition (1) ensures that after initialization the initial state of $\mathcal{A}$ satisfies the predicate *inv*. By Lemma 1 the conditions (2) and (3) ensure that *inv* is an invariant and a continuous invariant for all the actions in $\mathcal{A}$. By induction, like in Theorem 1, all these three conditions together prove that *inv* holds holds at all states in $\mathcal{A}$. Thus, *inv* is an invariant for $\mathcal{A}$. □

Similarly to Theorem 1, also Theorem 3 provides only a sufficient condition for checking an invariance properties of a linear hybrid action system. In other words, the algorithm constructed based on Theorem 3 is a semi-decision algorithm. It cannot be used for proving that a predicate is not an invariant for a linear hybrid action system. The algorithm can only return a reliable positive result,i.e., when a predicate is indeed an invariant.

The proof technique used in Theorem 3 provides an algorithm of linear complexity for checking an invariance property. This complexity corresponds to a linear complexity over the number of locations in a linear hybrid automaton [14]. This kind of efficiency is a necessity, when considering invariance verification of huge hybrid systems. However, due to the weakest precondition semantics an implementation of such an algorithm would require the use of some theorem prover, like PVS [8]. A similar approach in a discrete setting has been studied earlier by Rönkkö [13].

Example 2. We illustrate the use of Theorem 3 with the water-level monitor example, $\mathcal{M}onitor$, that was given in Example 1. We wish to show that in that example the water level y stays at all times within an interval of $1 \leq y \leq 12$. Before we may proceed with the proof, Theorem 3 requires us to show that the conjunct of the invariant with any of the evolution guards describes a convex set. In the case of the water-level monitor this step is quite clear, because the invariant, $1 \leq y \leq 12$, as well as all the evolution guards, $y \leq 10$, $x \leq 2$, $y \geq 5$, and $x \leq 2$ describe a convex set and an intersection of two convex sets is still a convex set. Therefore, the conjunct of the invariant with any of the evolution guards describes a convex set.

First we show that the invariant holds after initialization, the condition (1) of Theorem 3.

$$
\begin{array}{ll}
& \mathit{true} \Rightarrow \mathrm{wp}(x, y := 2, 1,\ 1 \le y \le 12) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\} \\
& \ \mathit{true} \Rightarrow 1 \le 1 \le 12 \\
\Leftrightarrow & \{\mathit{simplification}\} \\
& \ \mathit{true}
\end{array}
$$

Next we show that the invariant is an invariant for all the discrete behaviour, the condition (2) of Theorem 3. For brevity we show here only the proof for the first action. The proofs for the other three actions are shown in Appendix A.

$$
\begin{array}{ll}
& 1 \le y \le 12 \Rightarrow \mathrm{wp}(x=2 \wedge y \le 5 \rightarrow x, y := x, y,\ 1 \le y \le 12) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\ \mathit{for\ guarded\ command}\} \\
& \ 1 \le y \le 12 \Rightarrow (\ x=2 \wedge y \le 5 \Rightarrow \mathrm{wp}(x, y := x, y,\ 1 \le y \le 12)\ ) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\ \mathit{for\ assignment}\} \\
& \ 1 \le y \le 12 \Rightarrow (\ x=2 \wedge y \le 5 \Rightarrow 1 \le y \le 12\ ) \\
\Leftrightarrow & \{\mathit{unfolding\ second}\ \Rightarrow,\ \mathit{simplification}\} \\
& \mathit{true}
\end{array}
$$

Finally, we show that the invariant is a continuous invariant for all the actions, the condition (3) of Theorem 3. For brevity we show here only the proof for the first action. The proofs for the other three actions are shown in Appendix B.

$$
\begin{array}{ll}
& 1 \le y \le 12 \\
& \Rightarrow \mathrm{wp}(x=2 \wedge y \le 5 \rightarrow x, y := x, y;\ y \le 10 : \dot{x}, \dot{y} = 1, 1, \\
& \qquad 1 \le y \le 12 \wedge y \le 10) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\ \mathit{for\ guarded\ command,\ and\ simplifying\ postcondition}\} \\
& \ 1 \le y \le 12 \\
& \ \Rightarrow (\ x=2 \wedge y \le 5 \\
& \qquad \Rightarrow \mathrm{wp}(x, y := x, y;\ y \le 10 : \dot{x}, \dot{y} = 1, 1,\ \ 1 \le y \le 10)\ ) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\ \mathit{for\ sequential\ composition}\} \\
& \ 1 \le y \le 12 \\
& \ \Rightarrow (\ x=2 \wedge y \le 5 \\
& \qquad \Rightarrow \mathrm{wp}(x, y := x, y, \\
& \qquad\qquad \mathrm{wp}(y \le 10 : \dot{x}, \dot{y} = 1, 1,\ \ 1 \le y \le 10)\ )\ ) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\ \mathit{for\ differential\ action}\} \\
& \ 1 \le y \le 12 \\
& \ \Rightarrow (\ x=2 \wedge y \le 5 \Rightarrow \mathrm{wp}(x, y := x, y,\ 1 \le y \le 10\ )\ ) \\
\Leftrightarrow & \{\mathit{unfolding}\ \mathrm{wp}\ \mathit{for\ assignment}\} \\
& \ 1 \le y \le 12 \Rightarrow (\ x=2 \wedge y \le 5 \Rightarrow 1 \le y \le 10\ ) \\
\Leftrightarrow & \{\mathit{unfolding\ second}\ \Rightarrow\} \\
& \ 1 \le y \le 12 \Rightarrow (\ x \ne 2 \vee y > 5 \vee 1 \le y \le 10\ ) \\
\Leftrightarrow & \{\mathit{arithmetic\ simplification}\} \\
& \mathit{true}
\end{array}
$$

End of example

5 Algorithmic Analysis of Linear Hybrid Action Systems

In this section, we consider algorithmic analysis of linear hybrid action systems. First, we introduce timed traces for representing the behaviour of linear hybrid action systems. Then, we develop an algorithm for checking a class of linear action systems for reachability.

5.1 Behavioral semantics of Linear Hybrid Action Systems

For analyzing linear hybrid action systems, we introduce *timed traces* for representing the behaviour of linear hybrid action systems. A timed trace is of the form

$$X_0 \xrightarrow{(A_1,t_1)} X_1 \xrightarrow{(A_2,t_2)} \ldots \xrightarrow{(A_m,t_m)} X_m \tag{1}$$

where any X_i $(0 \leq i \leq m)$ is a state of linear hybrid action system, and for any j $(1 \leq j \leq m)$, A_j is an action and t_j is nonnegative real number. A timed trace of the form (1) represents that the system starts from the state X_0, executes action A_1 for t_1 time units such that state X_1 is reached, then executes action A_2 for t_2 time units such that state X_2 is reached, so on and so forth, at last executes action A_m for t_m time units such that state X_m is reached.

Definition 2. For a linear hybrid action system

$$\mathcal{A} \;\hat{=}\; |[\,\mathbf{var}\ X : T \ \bullet\ X := E_0; \mathbf{do}\, A_1 \,[\!]\, .. \,[\!]\, A_n\ \mathbf{od}\,]| \ ,$$

a timed trace of the form

$$X_0 \xrightarrow{(B_1,t_1)} X_1 \xrightarrow{(B_2,t_2)} \ldots \xrightarrow{(B_m,t_m)} X_m$$

represents a behaviour if and only if the following conditions hold:

1) $X_0 = E_0$;
2) for any B_i $(1 \leq i \leq m)$, there is A_j $(1 \leq j \leq n)$ such that $B_i = A_j$;
3) for any B_i $(1 \leq i \leq m)$ of the form

$$p_i \rightarrow X := E_i;\ e_i : \dot{X} = C_i \,,$$

p_i is satisfied by X_{i-1}; and
4) for any i $(1 \leq i \leq m)$, X_i is a successive state of X_{i-1} as explained in Section 3.

End of definition

Let σ be a timed trace which is of the form

$$X_0 \xrightarrow{(A_1,t_1)} X_1 \xrightarrow{(A_2,t_2)} \ldots \xrightarrow{(A_m,t_m)} X_m \,.$$

If all A_i $(1 \leq i \leq m)$ are distinct, i.e., $\forall\ 1 \leq i,j \leq m\ .\ i \neq j \ \Rightarrow\ A_i \neq A_j$, then we say that σ is a *simple* timed trace.

Definition 3. A linear hybrid action system $\mathcal{A}$ is *loop-closed* if and only if for any simple timed trace which represents a behaviour of $\mathcal{A}$ and is of the form

$$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \ldots \stackrel{(A_m,t_m)}{\longrightarrow} X_m ,$$

the following condition holds:

- If there is A_i $(1 \leq i \leq m)$ such that

  $$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \ldots \stackrel{(A_m,t_m)}{\longrightarrow} X_m \stackrel{(A_i,t)}{\longrightarrow} X_{m+1}$$

  represents a behaviour of $\mathcal{A}$, then $X_{i-1} = X_m$.

End of definition

For example, the linear hybrid action system in Example 1 describing a water-level monitor is loop-closed, because its behaviour is given as a timed trace

$$(2,1) \stackrel{(A_1,9)}{\longrightarrow} (11,10) \stackrel{(A_2,2)}{\longrightarrow} (2,12) \stackrel{(A_3,3.5)}{\longrightarrow} (5.5,5) \stackrel{(A_4,2)}{\longrightarrow} (2,1) \stackrel{(A_1,9)}{\longrightarrow} \ldots \quad ,$$

where a state is a tuple of the values of the variables, (x, y).

5.2 Reachability Analysis

Now we consider the problem of checking reachability for linear hybrid action systems. For a linear hybrid action system $\mathcal{A}$, for a predicate p, if there is a timed trace σ of the form

$$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \ldots \stackrel{(A_m,t_m)}{\longrightarrow} X_m$$

such that σ represents a behaviour of $\mathcal{A}$ and that p is satisfied by X_m, then we say that there is a reachable state satisfying p. The reachability problem is to check if there is a reachable state satisfying a predicate for a linear hybrid action system. For example, if we want to check if the water-level monitor in Example 1 satisfies the requirement that the water-level must be kept in between 1 and 12 inches, we can check if there is a reachable state satisfying $y < 1 \vee y > 12$ for the action system describing the water-level monitor in Example 1. In the following, we develop an algorithm for solving the problem for loop-closed linear hybrid action systems.

Intuitively, we can solve the problem by checking every timed trace which represents the behaviour of a linear hybrid action system. But we know that for a linear hybrid action system, the number of timed traces representing its behaviour could be infinite, and the timed traces could be infinite. However, the following theorem shows that for loop-closed linear hybrid action systems, we can solve the problem by checking a finite set of finite timed traces.

Theorem 4. Let $\mathcal{A}$ be a loop-closed linear hybrid action system, and p be a predicate. Then there is a reachable state satisfying p if and only if there is simple timed trace σ of the form

$$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \dots \stackrel{(A_m,t_m)}{\longrightarrow} X_m$$

such that σ represents a behaviour of $\mathcal{A}$ and that p is satisfied by X_m.

Proof. It is clear that claim holds to one direction: if there is simple timed trace σ of the form

$$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \dots \stackrel{(A_m,t_m)}{\longrightarrow} X_m$$

such that σ represents a behaviour of $\mathcal{A}$ and that p is satisfied by X_m, then there is a reachable state satisfying p. The claim can be proved to other direction as follows. Since there is a reachable state satisfying p, there is a timed trace σ of the form

$$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \dots \stackrel{(A_m,t_m)}{\longrightarrow} X_m$$

such that σ represents a behaviour of $\mathcal{A}$ and that p is satisfied by X_m. If σ is a simple timed trace, then the claim holds. Suppose that σ is not a simple timed trace. Let i $(1 \le i \le m)$ be such that $A_1, A_2, \dots, A_{i-1}$ are distinct and that there is A_j $(1 \le j \le i-1)$ satisfying $A_j = A_i$. Since $\mathcal{A}$ is loop-closed, $X_{j-1} = X_{i-1}$. It follows that by removing $X_{j-1}, X_j, \dots, X_{i-2}$ from σ, we get a timed trace σ' of the form

$$X_0 \stackrel{(A_1,t_1)}{\longrightarrow} X_1 \stackrel{(A_2,t_2)}{\longrightarrow} \dots \stackrel{(A_{j-2},t_{j-2})}{\longrightarrow} X_{j-2} \stackrel{(A_{j-1},t_{j-1})}{\longrightarrow} X_{i-1} \stackrel{(A_i,t_i)}{\longrightarrow} \dots \stackrel{(A_m,t_m)}{\longrightarrow} X_m$$

such that σ' is a behaviour of $\mathcal{A}$. By applying the above step repeatedly, we can get a simple timed trace whose last state satisfies p, i.e., the claim holds. $\Box$

Based on Theorem 4, we can now give an algorithm that checks reachability for linear hybrid action systems, see Figure 2. The algorithm is based on depth-first search method to check if a state satisfying a predicate p is reached by a simple timed trace representing the behaviour of a linear hybrid action system $\mathcal{A}$. The main data structures in the algorithm include a list *currentpath* of nodes of the form (A, t, X) which is used for recording current simple timed trace, a variable *node* which is used for recording the current node, and a boolean variable *unclosed* which is used for representing if the action system is loop-closed. The algorithm traverses simple timed traces starting from the initial node $(\perp, \perp, X_0)$. The simple timed trace that we have so far traversed is stored in the list *currentpath*. For each node (A, t, X) which is a new successive node of the last node in *currentpath* by executing action A for t time units such that X is reached, we first check if X satisfies p. If this happens, the algorithm returns **true**, and we are done. Otherwise we check if the current simple timed trace and the current node are such that the action system is not loop-closed. In this case, the variable *unclosed* is set to **true**. However at this point the algorithm does not add the current node into *currentpath*, because of Theorem 4, it backtracks instead. Finally if the current node is a new node which does not satisfy p, we add it into *currentpath* and start the search from it. The procedure terminates always because there is only a finite number of simple timed traces in the action system.

Notice that the algorithm can give the answer **true** for some linear hybrid action system which are not loop-closed, but not for all. It is, thus, a decision procedure for loop-closed linear hybrid action systems and a semi-decision procedure for non loop-closed linear hybrid action systems. The algorithm is of at worst squared complexity over the number of actions in a linear hybrid system.

```
currentpath := ⟨(⊥, ⊥, X0)⟩; unclosed := false;
if X0 satisfies p then return true;
repeat
   node := the last node of currentpath;
   if node has no new successive node
   then delete the last node of currentpath
   else begin
           node := (A, t, X), which is a new successive node of node;
           if X satisfies p then return true;
           if there is a node (A', t', X') in currentpath
              such that A = A' and X ≠ X'
           then unclosed := true;
           if there is no node (A', t', X') in currentpath
              such that A = A'
           then append node to currentpath;
        end
until currentpath = ⟨⟩;
if unclosed = false then return false  else  return undecided;
```

Figure 2: Algorithm checking reachability

The algorithm in Figure 2 can be used for verifying that the water-level monitor in Example 1 does not overflow, i.e., a state satisfying $y < 1 \vee y > 12$ is not reachable in Example 1. The following is an example of a linear hybrid system with an invariant that cannot be verified with the invariant verification algorithm, but can be checked with the algorithm in Figure 2.

Example 3. Consider a linear hybrid action system

$$\begin{array}{lll} \mathcal{X}only & \hat{=} & |[\;\; \textbf{var}\ x, y : I\!R, I\!R \bullet\ x, y := 1, 1; \\ & & \quad\;\; \textbf{do}\ x \leq 1 \rightarrow x := 1;\ x \geq 0 : \dot{x} = -1\ \textbf{od} \\ & &]| \;\; . \end{array}$$

Clearly, for this action system a predicate $x = 1 \vee y = 1$ is an invariant. As it was mentioned in Section 4, the guard in $\mathcal{X}only$ is too weak for the invariance verification algorithm. However, it is easy to see that with the algorithm in Figure 2 we can verify that a state $x \neq 1 \wedge y \neq 1$ is not reachable. Therefore, $x = 1 \vee y = 1$ is indeed an invariant.
End of example

6 Conclusions

In this paper we proposed two algorithms for analysis and verification of linear hybrid action systems that capture linear hybrid systems.

The first algorithm in Theorem 3 is an invariant verification algorithm of linear complexity over the number of actions. The algorithm is a semi-decision algorithm that can only return a reliable positive result, i.e., when a predicate is indeed an invariant. Also, due to the weakest precondition semantics an implementation of the algorithm would require the use of some theorem prover, like PVS.

The second algorithm in Figure 2 is a reachability algorithm of at worst squared complexity over the number of actions. This algorithm is a decision procedure for loop-closed linear hybrid action systems and a semi-decision procedure for non loop-closed linear hybrid action systems. Moreover, this algorithm could be implemented by using conventional programming languages.

The two given algorithms are complementary. For loop-closed linear hybrid action systems it is better to use the algorithm given in Figure 2. This algorithm is also more reliable, when the invariant is described as a disjunct of several properties. For non loop-closed linear hybrid action systems it is better to use the algorithm given in Theorem 3. Since both of these algorithms are semi-decisional there are such kind of linear hybrid action systems with invariance properties that cannot be verified by using either of these two algorithms. One possibility is then to use refinement for transforming the linear hybrid action systems into such form that the algorithms are applicable [16]. However, since the refinement process is hard to be automated, a topic for future research is to extend these two algorithms to handle a wider class of action systems. Also, another important topic for future work is to give mechanical support to the two algorithms presented in this paper.

References

[1] R. Alur, C. Couroubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho et al. The Algorithmic Analysis of Hybrid Systems. In *Theoretical Computer Science*, 138, 1995, pp. 3-34.

[2] R. J. R. Back. *Atomicity Refinement in a Refinement Calculus Frame-Work.* Reports on computer science and mathematics 141, Åbo Akademi, 1993.

[3] R. J. R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proc. of the 2nd ACM SIGACT–SIGOPS Symp. on Principles of Distributed Computing*, pp. 131–142, 1983.

[4] R. J. R. Back and K. Sere. Stepwise refinement of action systems. *Structured Programming*,**12**, pp. 17-30, 1991.

[5] M. Butler, E. Sekerinski, and K. Sere. An Action System Approach to the Steam Boiler Problem. In J-R. Abrial, E. Borger, H. Langmaack (eds.) *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165, Springer-Verlag, 1996.

[6] R. J. R. Back and J. von Wright. *Trace Refinement of Action Systems.* In J. Parrow, editor, *CONCUR94*, LNCS 666, Springer-Verlag, 1994.

[7] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, 22-24 October, 1995.

[8] J. Crow, S. Owre, J. Rushby, N. Shankar, M. Srivas. *A Tutorial Introduction to PVS.* Presented at *WIFT'95: Workshop on Industrial-Strength Formal Specification Techniques*, Boca Raton, Florida, April 1995.

[9] E. W. Dijkstra. *A Discipline of Programming.* Prentice–Hall International, 1976.

[10] T. A. Henzinger. The Theory of Hybrid Automata. In Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996), pp. 278-292.

[11] T. A. Henzinger and P.-H. Ho. HyTech: The Cornell hybrid technology tool. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, LNCS 999, pp. 265-293, Springer-Verlag, 1995.

[12] J. La Salle. *Stability by Liapunov's Direct Method with Applications.* Academic Press, 1961.

[13] Mauno Rönkkö. *Using PVS for Verifying Safety in Action Systems.* Technical Report A/1996/2, Department of Computer Science and Applied Mathematics, University of Kuopio, 1996.

[14] Mauno Rönkkö and Xuandong Li. *Linear Hybrid Action Systems.* No. 245, Technical Reports, Turku Centre for Computer Science, Åbo Akademi University, Finland, March 1999.

[15] M. Rönkkö and A. P. Ravn. Action Systems with Continuous Behaviour. In P. J. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry, editors, *Proceedings of Hybrid Systems V*, LNCS 1567, pp. 304-323, Springer-Verlag, 1999.

[16] M. Rönkkö and K. Sere. Refinement and Continuous Behaviour. In F.W. Vaandrager, and J.H. van Schuppen, editors, *Proceedings of Hybrid Systems: Computation and Control*, LNCS 1569, pp. 223-237, Springer-Verlag, 1999.

[17] J. Vitt and J. Hooman. Assertional Specification and Verification using PVS of the Steam Boiler Control System. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165, pp. 453-472, Springer-Verlag, 1996.

[18] Li Xuandong, Zheng Tao, Hou Jianmin, Zhao Jianhua, and Zheng Guoliang. *Hybrid Regular Expressions.* In Hybrid Systems: Computation and Control LNCS 1386, pp. 384-399, Springer-Verlag, 1998.

A Invariant Proofs for Discrete Behaviour

Proof for the second action.

$$
\begin{aligned}
& 1 \le y \le 12 \Rightarrow \mathrm{wp}(y = 10 \rightarrow x, y := 0, y,\ 1 \le y \le 12) \\
\Leftrightarrow\ & \{\textit{unfolding } \mathrm{wp} \textit{ for guarded command}\} \\
& 1 \le y \le 12 \Rightarrow (\ y = 10 \Rightarrow \mathrm{wp}(x, y := 0, y,\ 1 \le y \le 12)\ ) \\
\Leftrightarrow\ & \{\textit{unfolding } \mathrm{wp} \textit{ for assignment}\} \\
& 1 \le y \le 12 \Rightarrow (\ y = 10 \Rightarrow 1 \le y \le 12\ ) \\
\Leftrightarrow\ & \{\textit{unfolding second } \Rightarrow\} \\
& 1 \le y \le 12 \Rightarrow (\ y \ne 10 \vee 1 \le y \le 12\ ) \\
\Leftrightarrow\ & \{\textit{simplification}\} \\
& \textit{true}
\end{aligned}
$$

Proof for the third action.

$$
\begin{aligned}
& 1 \le y \le 12 \Rightarrow \mathrm{wp}(x = 2 \wedge y \ge 10 \rightarrow x, y := x, y,\ 1 \le y \le 12) \\
\Leftrightarrow\ & \{\textit{unfolding } \mathrm{wp} \textit{ for guarded command}\} \\
& 1 \le y \le 12 \Rightarrow (\ x = 2 \wedge y \ge 10 \Rightarrow \mathrm{wp}(x, y := x, y,\ 1 \le y \le 12)\ ) \\
\Leftrightarrow\ & \{\textit{unfolding } \mathrm{wp} \textit{ for assignment}\} \\
& 1 \le y \le 12 \Rightarrow (\ x = 2 \wedge y \ge 10 \Rightarrow 1 \le y \le 12\ ) \\
\Leftrightarrow\ & \{\textit{unfolding second } \Rightarrow\} \\
& 1 \le y \le 12 \Rightarrow (\ x \ne 2 \vee y < 10 \vee 1 \le y \le 12\ ) \\
\Leftrightarrow\ & \{\textit{simplification}\} \\
& \textit{true}
\end{aligned}
$$

Proof for the fourth action.

$$
\begin{aligned}
& 1 \le y \le 12 \Rightarrow \mathrm{wp}(y = 5 \rightarrow x, y := 0, y,\ 1 \le y \le 12) \\
\Leftrightarrow\ & \{\textit{unfolding } \mathrm{wp} \textit{ for guarded command}\} \\
& 1 \le y \le 12 \Rightarrow (\ y = 5 \Rightarrow \mathrm{wp}(x, y := 0, y,\ 1 \le y \le 12)\ ) \\
\Leftrightarrow\ & \{\textit{unfolding } \mathrm{wp} \textit{ for assignment}\} \\
& 1 \le y \le 12 \Rightarrow (\ y = 5 \Rightarrow 1 \le y \le 12\ ) \\
\Leftrightarrow\ & \{\textit{unfolding second } \Rightarrow\} \\
& 1 \le y \le 12 \Rightarrow (\ y \ne 5 \vee 1 \le y \le 12\ ) \\
\Leftrightarrow\ & \{\textit{simplification}\} \\
& \textit{true}
\end{aligned}
$$

B Invariant Proofs for Continuous Behaviour

Proof for the second action.

$$
\begin{array}{ll}
 & 1 \le y \le 12 \\
 & \Rightarrow \ \mathrm{wp}(y=10 \rightarrow x, y := 0, y;\ \ x \le 2 : \dot{x}, \dot{y} = 1, 1, \\
 & \qquad\quad 1 \le y \le 12 \ \wedge \ x \le 2) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for guarded command}\} \\
 & \ 1 \le y \le 12 \\
 & \ \ \Rightarrow \ (\ y = 10 \\
 & \qquad\quad \Rightarrow \ \mathrm{wp}(x, y := 0, y;\ \ x \le 2 : \dot{x}, \dot{y} = 1, 1, \quad 1 \le y \le 12 \ \wedge \ x \le 2)\ ) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for sequential composition}\} \\
 & \ 1 \le y \le 12 \\
 & \ \ \Rightarrow \ (\ y = 10 \\
 & \qquad\quad \Rightarrow \ \mathrm{wp}(x, y := 0, y, \\
 & \qquad\qquad\quad \mathrm{wp}(x \le 2 : \dot{x}, \dot{y} = 1, 1, \quad 1 \le y \le 12 \ \wedge \ x \le 2)\ )\ ) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for differential action}\} \\
 & \ 1 \le y \le 12 \\
 & \ \ \Rightarrow \ (\ y = 10 \ \Rightarrow \ \mathrm{wp}(x, y := 0, y,\ 1 \le y + (2 - x) \le 12 \ \wedge \ x \le 2\ )\ ) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for assignment}\} \\
 & \ 1 \le y \le 12 \ \Rightarrow \ (\ y = 10 \ \Rightarrow \ 1 \le y + (2 - 0) \le 12 \ \wedge \ 0 \le 2\ ) \\
\Leftrightarrow & \{\textit{simplification}\} \\
 & \ 1 \le y \le 12 \ \Rightarrow \ (\ y = 10 \ \Rightarrow \ -1 \le y \le 10\ ) \\
\Leftrightarrow & \{\textit{unfolding second} \ \Rightarrow\} \\
 & \ 1 \le y \le 12 \ \Rightarrow \ (\ y \ne 10 \ \vee \ -1 \le y \le 10\ ) \\
\Leftrightarrow & \{\textit{arithmetic simplification},\ \textit{y belongs to}\ \mathbb{R}\} \\
 & \ 1 \le y \le 12 \ \Rightarrow \ \textit{true} \\
\Leftrightarrow & \{\textit{simplification}\} \\
 & \ \textit{true}
\end{array}
$$

Proof for the third action.

$$
\begin{array}{ll}
& 1 \le y \le 12 \\
& \Rightarrow \mathrm{wp}(x\!=\!2 \wedge y\!\ge\!10 \rightarrow x,y := x,y;\ y\!\ge\!5 : \dot{x},\dot{y}\!=\!1,-2, \\
& \qquad 1 \le y \le 12 \wedge y\!\ge\!5) \\
\Leftrightarrow & \{\textit{unfolding}\ \mathrm{wp}\ \textit{for guarded command, and simplifying postcondition}\} \\
& 1 \le y \le 12 \\
& \Rightarrow (\ x\!=\!2 \wedge y\!\ge\!10 \\
& \qquad \Rightarrow \mathrm{wp}(x,y := x,y;\ y\!\ge\!5 : \dot{x},\dot{y}\!=\!1,-2,\ \ 5 \le y \le 12)\ ) \\
\Leftrightarrow & \{\textit{unfolding}\ \mathrm{wp}\ \textit{for sequential composition}\} \\
& 1 \le y \le 12 \\
& \Rightarrow (\ x\!=\!2 \wedge y\!\ge\!10 \\
& \qquad \Rightarrow \mathrm{wp}(x,y := x,y, \\
& \qquad\qquad \mathrm{wp}(y\!\ge\!5 : \dot{x},\dot{y}\!=\!1,-2,\ \ 5 \le y \le 12)\ )\ ) \\
\Leftrightarrow & \{\textit{unfolding}\ \mathrm{wp}\ \textit{for differential action}\} \\
& 1 \le y \le 12 \\
& \Rightarrow (\ x\!=\!2 \wedge y\!\ge\!10 \Rightarrow \mathrm{wp}(x,y := x,y,\ y\!\ge\!5\ )\ ) \\
\Leftrightarrow & \{\textit{unfolding}\ \mathrm{wp}\ \textit{for assignment}\} \\
& 1 \le y \le 12 \Rightarrow (\ x\!=\!2 \wedge y\!\ge\!10 \Rightarrow y\!\ge\!5\ ) \\
\Leftrightarrow & \{\textit{unfolding second}\ \ \Rightarrow\} \\
& 1 \le y \le 12 \Rightarrow (\ x\!\ne\!2 \vee y\!<\!10 \vee y\!\ge\!5\ ) \\
\Leftrightarrow & \{\textit{arithmetic simplification},\ \textit{y belongs to}\ \mathbb{R}\} \\
& 1 \le y \le 12 \Rightarrow \textit{true} \\
\Leftrightarrow & \{\textit{simplification}\} \\
& \textit{true}
\end{array}
$$

Proof for the fourth action.

$$
\begin{array}{ll}
& 1 \leq y \leq 12 \\
& \Rightarrow \mathrm{wp}(y=5 \rightarrow x, y := 0, y;\ x \leq 2 : \dot{x}, \dot{y} = 1, -2, \\
& \qquad 1 \leq y \leq 12 \wedge x \leq 2) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for guarded command}\} \\
& 1 \leq y \leq 12 \\
& \Rightarrow (\ y=5 \\
& \qquad \Rightarrow \mathrm{wp}(x, y := 0, y;\ x \leq 2 : \dot{x}, \dot{y} = 1, -2,\ \ 1 \leq y \leq 12 \wedge x \leq 2)\ ) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for sequential composition}\} \\
& 1 \leq y \leq 12 \\
& \Rightarrow (\ y=5 \\
& \qquad \Rightarrow \mathrm{wp}(x, y := 0, y, \\
& \qquad\qquad \mathrm{wp}(x \leq 2 : \dot{x}, \dot{y} = 1, -2,\ \ 1 \leq y \leq 12 \wedge x \leq 2)\ )\ ) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for differential action}\} \\
& 1 \leq y \leq 12 \\
& \Rightarrow (\ y=5 \Rightarrow \mathrm{wp}(x, y := 0, y,\ 1 \leq y - 2 \cdot (2 - x) \leq 12 \wedge x \leq 2\ )\ ) \\
\Leftrightarrow & \{\textit{unfolding } \mathrm{wp} \textit{ for assignment}\} \\
& 1 \leq y \leq 12 \Rightarrow (\ y=5 \Rightarrow 1 \leq y - 2 \cdot (2 - 0) \leq 12 \wedge 0 \leq 2\ ) \\
\Leftrightarrow & \{\textit{simplification}\} \\
& 1 \leq y \leq 12 \Rightarrow (\ y=5 \Rightarrow 5 \leq y \leq 16\ ) \\
\Leftrightarrow & \{\textit{unfolding second } \Rightarrow\} \\
& 1 \leq y \leq 12 \Rightarrow (\ y \neq 5 \vee 5 \leq y \leq 16\ ) \\
\Leftrightarrow & \{\textit{arithmetic simplification, } y \textit{ belongs to } \mathbb{R}\} \\
& 1 \leq y \leq 12 \Rightarrow \textit{true} \\
\Leftrightarrow & \{\textit{simplification}\} \\
& \textit{true}
\end{array}
$$

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi

University of Turku
- **Department of Mathematical Sciences**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**