ELSEVIER

Theoretical Computer Science 290 (2003) 937–973

Theoretical Computer Science

www.elsevier.com/locate/tcs

Hybrid action systems

Mauno Rönkkö[a,*], Anders P. Ravn[b], Kaisa Sere[a]

[a] *Department of Computer Science, Turku Centre for Computer Science, Åbo Akademi University, Lemminkäisenkatu 14A, FIN-20520 Turku, Finland*

[b] *Department of Computer Science, Aalborg University, Frederik Bajers Vej 7E, DK-9220 Aalborg, Denmark*

Received 23 March 2000; received in revised form 4 June 2002; accepted 12 June 2002
Communicated by D. Sannella

Abstract

In this paper we investigate the use of action systems with differential actions in the specification of hybrid systems. As the main contribution we generalize the definition of a differential action, allowing the use of arbitrary relations over model variables and their time derivatives in modelling continuous-time dynamics. The generalized differential action has an intuitively appealing predicate transformer semantics, which we show to be both conjunctive and monotonic. In addition, we show that differential actions blend smoothly with conventional actions in action systems even under parallel composition. Moreover, as the strength of the action system formalism is the support for stepwise development by refinement, we investigate refinement involving a differential action. We show that, due to the predicate transformer semantics, standard action refinement techniques apply also to the differential action, thus, allowing stepwise development of hybrid systems.

Keywords: Hybrid systems; Predicate transformers; Action systems; Refinement calculus; Concurrency

1. Introduction

Computers have found widespread use in the design and development of mechanical systems such as vehicles, robots, or instruments. Such embedded computer systems replace mechanical, electro-mechanical or electronic components and offer cost-effective solutions because they are constructed from mass-produced digital processors, highly

* Corresponding author.
E-mail addresses: mronkko@cs.utu.fi (Mauno Rönkkö), apr@cs.auc.dk (Anders P. Ravn), kaisa.sere@abo.fi (Kaisa Sere).

0304-3975/02/$ - see front matter
PII: S0304-3975(02)00547-9

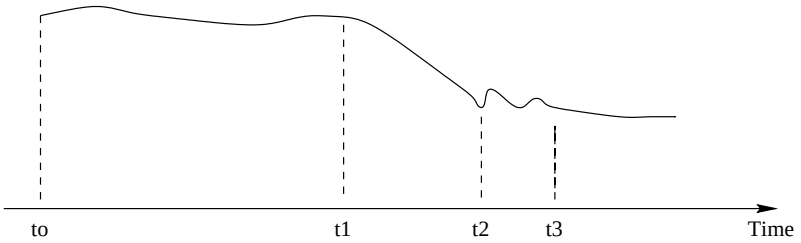

Fig. 1. A trajectory in a hybrid system. The velocity of a robot link is regulated to a normal speed (t0), when an object is detected (t1), the link has to decelerate in order to meet (t2) and follow (t3) the moving object.

standardized hardware interfaces and programs. They also allow more sophisticated and dependable solutions because digitalization allows precisions that are unattainable by analog devices. It is also possible to build highly complex mathematical models into the equipment, because computer storage for programs and data is inexpensive. The challenge in building such systems is no longer in finding ingenious mechanical or electronic solutions; it is to develop computer programs that interact with the mechanical system in such a way that the whole mechanism performs as expected.

Principles of program design have been studied intensively, and although industrial practice is more to build than to engineer programs, there are mature theories for program design, in particular reactive programs that interact with an environment. The basic model underlying these theories is the concept of a collection of state machines. Each state machine makes discrete transitions from state to state as determined by a current state and potential communication of events with other state machines. However, in embedded systems it is not enough to understand the interactions and state transitions of the computer program; one must also relate the computations to the continuous-time state change of the mechanism as a whole. A simple illustration is Fig. 1 which is a simplified graph illustrating how a robot link changes its velocity when approaching an object. It is clear that such changes may be controlled by a state machine which supervises the movement and changes between different modes. The mode changes are observed whenever the continuous-time state of the mechanism enters certain regions. This is an event that triggers a transition, leading to a computation that selects a new control algorithm. In recent years the fundamental models for such hybrid systems have been investigated [11,20]. These models extend the state machine framework by associating a continuous-time flow with at least some states. Typically, the flow is specified by a differential equation or relation and is enabled by a guard condition that determines when the flow is active. These models provide a framework for analyzing hybrid systems; but further work is needed to integrate control theory analysis and synthesis techniques with program design techniques in a unified framework for developing concrete hybrid systems. For such a framework to function in the large, some structuring devices and compositionality must also be considered. In this paper we consolidate our contributions to this research within the program design techniques associated with action systems.

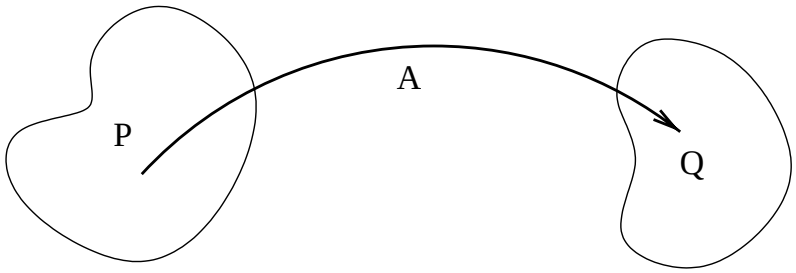

Fig. 2. Semantics of actions. The action A will terminate with a value in the region Q, defined by a predicate q over the model variables, provided that it is activated in the region P which is defined by $\mathrm{wp}(A, q)$.

Action systems were originally proposed by Back and Kurki-Suonio [4]. They have been successfully used in development of reactive and concurrent systems [5,13]. Action systems are iterated systems of actions based on Dijkstra's *guarded command language* [14], which is essentially a formal model of a general imperative programming language. Thus, concerning the syntax, action systems allow a smoother treatment of computer algorithms than state machine approaches. In addition, action systems support the structuring of a model by providing rules for variable scopes and parallel composition of models. All this brings a little additional complexity to the formalism, but it allows a systematic treatment of large-scale reactive and distributed systems. The meaning to *actions* (state transitions) of the language is given by defining a *predicate transformer* for them. The basic idea is illustrated in Fig. 2. Consider a final state characterized by a predicate q over the collection of model variables. The predicate transformer "wp" will for an action A define a weakest precondition predicate $\mathrm{wp}(A, q)$ that characterizes the initial model variable values for which the action terminates with a result in q.

In action systems, all the model variables denote values of functions of time, and there is no fixed variable denoting the global time. For specifying hybrid systems we have introduced a *differential action* [28] that maps continuous-time dynamics, *evolutions*, to the model variables as flows from an initial region to a final region. An evolution is atomic, i.e., the intermediate states are not observable. However, the differential action does allow the parametrization of an evolution region, which supports the simulation of an intermediate observation. We prefer this approach because it simplifies the reasoning in two ways: it localizes the analysis of evolutions entirely to individual differential actions, and it supports reasoning of hybrid dynamics on a higher level of transition sequences. In our initial work [28], we were influenced by existing models for hybrid systems and defined the differential action by an ordinary differential equation guarded by a condition—an evolution guard. It had quite an intuitive predicate transformer semantics that essentially establish the initial region for which a unique solution to the differential equation moves the state through a region with enabled guard to a final state satisfying the desired postcondition. This definition worked smoothly in our investigation of properties of hybrid action systems and their parallel composition [27]. However, action systems support *stepwise development* by *refinement* as formalized in

the *refinement calculus* [7]. Although some interesting results on the equivalence of differential actions under path transformations were established [29], the use of ordinary differential equations was somewhat restrictive; unique solutions are very deterministic, colliding with the refinement framework which uses non-determinism to capture abstractions. For this reason, we have reworked the concepts and generalized the differential action to allow use of differential relations, which are general predicates on model variables and their first-order time-derivatives.

Overview. The following sections introduce the differential action and its use in the action system framework. Throughout the text there are small examples. Most of these examples use simple differential relations which are easy to solve, because they serve to illustrate action systems and not mathematical analysis techniques.

We start in Section 2 by describing the state space of action systems and the use of predicates in reasoning about states and state changes. In Section 3 we define conventional actions. In Section 4 we introduce the generalized differential action and its semantics. In Section 5 we present action systems along withtheir parallel composition. In that section we explore concurrency of evolutions and introduce a class of action systems, hybrid action systems, for which we extend the parallel composition to support concurrency. In Section 6 we investigate foundations for refinement of hybrid action systems; and Section 7 compares action systems with related formalisms.

2. State space

In action systems, a model is defined over a fixed set of typed variables. A list $X = x_1, \ldots, x_n$ of variables with types $T = T_1, \ldots, T_n$ is introduced by a declaration of the form: $\mathbf{var}\, x_1, \ldots, x_n : T_1, \ldots, T_n$ or as a shorthand $\mathbf{var}\, X : T$.

We assume a selection of the usual discrete and numeric types, in particular the real numbers. A *state* of a model is a type consistent assignment of values to the variables, and the state space is the set of all such assignments. For a given model, a *predicate* is a first-order predicate, where the variables of X may occur free. A predicate denotes the possibly empty subset of the state space that satisfies it. We use $PRED(X)$ to denote the set of predicates for a model with variables X. A first-order predicate may also contain variables bound by quantifiers, these are logical variables and not part of the state space. Consider, for instance, a state space where x denotes a location (m) and v denotes a velocity (m/s), i.e., $\mathbf{var}\, x, v : \mathbb{R}, \mathbb{R}$. Then, $\exists \alpha : \mathbb{R}.0 \leqslant \alpha \leqslant 10 \wedge x = \alpha \Rightarrow v = 1$, where α is a bound variable, states that somewhere in an interval of 10 m the velocity is 1 m/s.

A predicate can be transformed by *textual substitution* [14] where a model variable is replaced with some expression throughout a predicate. Textual substitution is used, for instance, for reflecting a state change caused by the execution of an action. We denote the textual substitution of model variables X with expression E in a predicate p by

$$p[E/X]$$

Textual substitution is a postfix operator and, thus, affects only the term on its left. If such a term is a compound term, parenthesis are used for indicating it. For instance, the expression $x = 2 \vee (x = 3 \vee x = 4)[4/x] \vee x = 5$ unfolds to $x = 2 \vee 4 = 3 \vee 4 = 4 \vee x = 5$.

The state space is in the following used to give semantics to conventional actions, the new differential action, and action systems. For both kinds of actions, the weakest precondition semantics implicitly specifies a transition relation on the state space. For a differential action the transition relation includes also an additional constraint: a flow or an evolution satisfying the action must exist and relate two values of the state space. For action systems this implicitly defines a conventional hybrid systems model with a finite or infinite sequence of transitions that do not take time, and evolutions that take time. This is a hybrid systems model similar to the discrete model found in Back and von Wright [7].

3. Conventional actions

Conventional actions are used for capturing discrete-time dynamics, *jumps* [10], in hybrid systems. A conventional action is any statement in Dijkstra's *guarded command language* [14], including a pure guarded command. We shall also use Back's *non-deterministic assignment* [3] as a conventional action.

We define the meaning of an action with a *weakest precondition predicate transformer* [14]. A weakest precondition describes the largest set of states from which the execution of a given action terminates in a state satisfying a given postcondition. The weakest precondition predicate transformer returns a predicate, the weakest precondition, for a given action and a postcondition predicate. For model variables $\mathbf{var}\, X : T$, a postcondition $q : PRED(X)$, and an action A operating on X the weakest precondition is denoted

$$\mathrm{wp}(A, q)$$

An action can start executing only if it is *enabled*. Formally, the set of states in which an action is enabled is given by $\mathrm{g}(A) \mathrel{\hat{=}} \neg \mathrm{wp}(A, \mathit{false})$. Thus, an action is said to be disabled in states $\neg \mathrm{g}(A)$. An executing action may either terminate or continue indefinitely.The set of states from which an action terminates, i.e., reaches some final state, is given by $\mathrm{t}(A) \mathrel{\hat{=}} \mathrm{wp}(A, \mathit{true})$. An action is said to *abort* in all those states from where it does not terminate, i.e., $\neg \mathrm{t}(A)$.

We consider the execution of an action *atomic*, i.e., an action is executed to its completion before other actions are considered for execution. Therefore, if an action aborts, the other actions will not have a chance for execution.

The weakest precondition semantics of the conventional actions that we consider in this paper are given in Table 1 [3,7,14]. *Abort* halts the computation. *Skip* models an action that does nothing. *Assertion* models a condition that must hold in a given stage of computation. *Assignment* sets the values of the model variables to the values of given expressions. *Non-deterministic assignment* changes non-deterministically the values of the model variables so that a given condition holds. *Sequential composition* executes one action after another. *Non-deterministic choice* selects arbitrarily one enabled action

Table 1
Semantics of conventional actions

Action	Notation	wp(Action,q)
Abort	**abort**	*false*
Skip	**skip**	q
Assertion	$\{p\}$	$p \wedge q$
Assignment	$X := E$	$q[E/X]$
Non-deterministic assignment	$X := \chi.r$	$\forall\chi\colon\ r.q[\chi/X]$
Sequential composition	$A; B$	$\mathrm{wp}(A, \mathrm{wp}(B, q))$
Guarded command	$p \rightarrow A$	$p \Rightarrow \mathrm{wp}(A, q)$
Non-deterministic choice	$A \,[]\, B$	$\mathrm{wp}(A, q) \wedge \mathrm{wp}(B, q)$
Iteration	**do** A **od**	$\mathrm{wp}(\mu S.\, \mathrm{g}(A) \rightarrow A; S \,[]\, \neg \mathrm{g}(A) \rightarrow \mathbf{skip}, q)$

X are the model variables, and χ are bound variables disjoint from X. In addition, there are predicates $p : PRED(X)$, $q : PRED(X)$, and $r : PRED(X, \chi)$. Lastly, A and B denote some actions and, in the iteration, $S : PRED(X) \rightarrow PRED(X)$ is an action variable, and μS denotes the *least fixed point* [7].

Table 2
Healthiness conditions for actions

Property	Healthiness condition that holds
Strictness	$\mathrm{wp}(A, false) \equiv false$
Monotonicity	$(p \Rightarrow q) \Rightarrow (\mathrm{wp}(A, p) \Rightarrow \mathrm{wp}(A, q))$
Conjunctivity	$\mathrm{wp}(A, p) \wedge \mathrm{wp}(A, q) \equiv \mathrm{wp}(A, p \wedge q)$
Bounded non-determinism	$\mathrm{wp}(A, \exists i : N.q_i) \equiv \exists i : N.\mathrm{wp}(A, q_i)$

Here, A is an action, p and q are predicates, and q_i is a member of a chain $\{q_i | i \in \mathbb{N}\}$ of predicates such that $q_i \Rightarrow q_{i+1}$ for each i.

from the given actions and executes it. *Guarded command* executes its action provided that its guard holds in the current state. *Iteration* executes repeatedly an action till it becomes disabled.

3.1. Healthiness conditions

In addition to enabledness and termination there are also other general properties of actions that can be computed with the weakest precondition, for instance, the *healthiness conditions* [14]. All of these conditions must be met by an action that is used as a program statement. There are four healthiness conditions [14]: *strictness*, *monotonicity*, *conjunctivity*, and *bounded non-determinism*. Strictness captures the enabledness of an action. A non-strict action is also said to be *miraculous* [7], because it can establish the impossible. Monotonicity allows, for instance, context independent refinement of actions [7]. Another healthiness condition related to monotonicity is conjunctivity: any conjunctive action is also monotonic [7]. Finally, bounded non-determinism has to do with computability [7]: an action can be computed with a machine if and only if it is boundedly non-deterministic. A boundedly non-deterministic action consists of only a finite number of outcomes [7]. The healthiness conditions are presented in Table 2.

Note that even some of the conventional actions do not fulfil all the four healthiness conditions. For instance, a non-deterministic assignment is not necessarily boundedly non-deterministic. Still, all the conventional actions that we consider in this paper are conjunctive [7].

4. The differential action

Conventional actions like iteration and assignment can be used to give a discrete approximation to the *evolutions* [10] of a continuous-time dynamical system. However, this approach leads quickly to expressions that are impractical. So we must look for an action that captures continuous-time dynamics, the *differential action*. It describes how the values of certain specified model variables evolve starting from their initial values. When the evolution terminates the model variables are left with the reached values.

The differential action is a formal device that maps the effect of an evolution to model variables. Syntactically, it has two parts: an *evolution guard* which is a predicate over the model variables and describes the set of states where evolutions may start, and a *differential relation* which is a predicate over the model variables and their first order derivatives and, thus, captures the evolutions of interest. Note that higher order derivatives can always be reduced [21] to the first order by using additional model variables. For an evolution guard e and a differential relation d the differential action is written as

$$e :\rightarrow d$$

We consider the execution of a differential action atomic. Hence, an evolution proceeds uninterrupted within states captured by the evolution guard. If an evolution terminates, it does so only at the boundary of the evolution guard, never in an interior state.

In the differential relation, a model variable denotes a function of time that has as its initial value the value of the corresponding model variable in the current state. For instance, for $\mathbf{var}\, c : \mathbb{R}$, instead of $\phi(0) = c \wedge \dot{\phi}(\tau) = 1$ we write $\dot{c} = 1$. Thus, the condition $\phi(0) = c$ becomes implicitly added. Note that, a zero derivative forces the value of the corresponding function to remain constant over time. Also, by convention, we assume in the sequel that the derivative of a variable in a differential relation is zero unless it is explicitly given.This convention ensures that discrete variables remain unchanged during an evolution.

Functions that satisfy the differential relation by textual substitution [33] are said to form the *solution set*. There may be more than one solution function to a differential relation, in which case the differential relation is said to be non-deterministic. We consider as solution functions only functions of class C^1, i.e., smooth continuous functions of time whose first derivative is also continuous.[1] For instance, from an initial state

[1] Generally speaking, continuity of the solution function is required only on a region in which the evolution takes place; but in order to discuss the solution function independently of the evolution guard, we insist on general continuity.

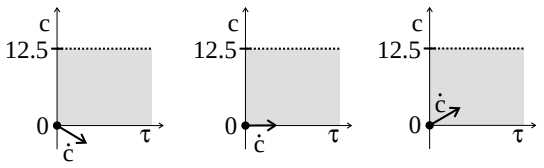

Fig. 3. Evolution at the boundary, $c=0$, of an evolution guard $0\leqslant c<12.5$. On the left: evolution terminates as it moves to the exterior of the evolution guard. In the middle: evolution remains at the boundary forever. On the right: evolution continues as it moves to the interior of the evolution guard.

$c=0$, the function $\phi:(\lambda\tau:\mathbb{R}.\tau)$ is a solution to the differential relation $\dot{c}=1$, because the textual substitution $\forall\tau.(\dot{c}=1)[\phi(\tau)/c,\ \dot{\phi}(\tau)/\dot{c}]$ simplifies to $1=1$ that holds.

Operationally, the differential action resembles the *guarded command* [7,14]. It is enabled only in a state from where there exists some *evolution*, a solution to the differential relation that also satisfies the evolution guard for some (non-zero) period of time. When an enabled differential action is executed, an evolution describes how the values of the corresponding model variables evolve. And, when an evolution terminates, if ever, it leaves the model variables with the reached values.

The termination of an evolution is determined by the evolution guard; an evolution cannot move outside the evolution guard, while it proceeds uninterrupted within the evolution guard. Therefore, in principle, an evolution terminates "as soon as" it reaches an *exterior* state, a state outside the evolution guard. However, due to use of real valued numbers the "first" exterior state does not always exist. Consider, for instance, an evolution guard $0\leqslant c<12.5$. Clearly, the "first" exterior state at the upper boundary is $c=12.5$; but, there is no "first" exterior state at the lower boundary, $c=0$, because no matter how near a state $c=\sigma$, $\sigma<0$, we choose there is always some state between the states $c=\sigma$ and $c=0$. Consequently, if an evolution terminates, it does so at the *boundary* of the evolution guard. Thus, the termination of an evolution depends also on the "direction" of the evolution. Consider the evolution guard above whose boundary states are $c\in\{0,12.5\}$ and *interior* states are $0<c<12.5$. If a boundary state is not part of the evolution guard, $c=12.5$ above, an evolution must terminate when it reaches such a state. However, if a boundary state is part of the evolution guard, $c=0$ above, an evolution must terminate in that state only if it is directed towards the exterior of the evolution guard. Consider, for instance, an evolution given by $\dot{c}=d$, where d is a constant. Such an evolution, for the evolution guard above, terminates in a state $c=0$ only if $d<0$, because only then the evolution moves to the exterior of the evolution guard. For $d=0$ the evolution remains in the state $c=0$ for ever, and for $d>0$ the evolution moves to the interior of the evolution guard.These facets are illustrated by Fig. 3.

As a summarizing example, consider the differential action $0\leqslant c\leqslant 10:\rightarrow\dot{c}=1$ modelling a delay of at most 10 s. Suppose, $c=0$ initially. Then, the differential action is enabled, because the evolution guard holds for a function $\phi:(\lambda\tau:\mathbb{R}.\tau)$ which is a solution to the differential relation as explained above. For ϕ, the evolution guard holds while $\tau\in[0,10]$. After 10 s the evolution reaches the state $\phi(10)=10$, i.e., $c=10$, which is a boundary state. Since, from this state, the evolution is headed for

an exterior state, it must terminate at this state. Hence, the duration of the evolution is 10 s, and the evolution leaves the model variable c with the reached value $c = 10$.

4.1. Weakest precondition semantics

We give for the differential action a weakest precondition semantics similar to the guarded command.This means that the weakest precondition captures also states from which there are no solutions satisfying the differential relation, or in which the evolution guard does not hold. However, in these states the differential action is considered disabled.

In the sequel we use the following notation. We denote a general type by T, and the class of continuous functions of time $\mathbb{R} \to \mathbb{R}^n$, whose first derivative is also continuous, by C^1. Let $X : \mathbb{R}^n$ and $Y : T$ be model variables with the intent that X are the evolved variables and Y are the remaining model variables. Also, let $e : PRED(X, Y)$, $d : PRED(X, \dot{X}, Y)$, and $q : PRED(X, Y)$ be predicates with the intent that e is an evolution guard, d is a differential relation, and q is a postcondition. Thus, the considered differential action is of form $e :\to d$. Finally, let $\phi : C^1$ be a function of time with $\phi(0)$ as its initial value.

We first formalize the set of solution functions. Informally, a solution function is of class C^1, it has as its initial value the current values of the variables, and, from the initial state, it satisfies the differential relation while it satisfies the evolution guard.

Definition 1. The function ϕ is a solution function of an evolution guard e and a differential relation d when it satisfies the predicate $SF(\phi, e, d)$. It is defined as

$$SF(\phi, e, d) \hat{=} \phi(0) = X \wedge \forall \tau : \mathbb{R} \cap [0, \infty).\ (e \Rightarrow d)[\phi(\tau)/X,\ \dot{\phi}(\tau)/\dot{X}]$$

Next, we define the duration of a function with respect to the evolution guard, i.e., how long a function satisfies the evolution guard. At any point τ of time, when a function ϕ satisfies the evolution guard, the condition $e[\phi(\tau)/X]$ holds. The "earliest" point of time when this condition does not hold, if such exists, determines the duration. By using infimum as the measure for the "earliest" we get precisely the termination behaviour at the boundary, as discussed.

Definition 2. We denote the duration of ϕ for e by $\Delta(\phi, e)$:

$$\Delta(\phi, e) \hat{=} \inf\{\tau : \mathbb{R} \cap [0, \infty) \mid \neg\, e[\phi(\tau)/X]\}$$

When ϕ satisfies e forever we define: $\Delta(\phi, e) \hat{=} \infty$.

Now, we can say precisely when a function describes an evolution of a differential action; such a function must be a solution function and, since an evolution takes time, it must satisfy the evolution guard for a non-zero period of time:

Definition 3. A function ϕ describes an evolution of $e :\to d$, if

$$SF(\phi, e, d) \wedge \Delta(\phi, e) > 0$$

Corollary 4. *When a function ϕ describes an evolution of $e :\rightarrow d$, the duration of the evolution is given by $\Delta(\phi, e)$.*

Corollary 5. *A function ϕ describes a terminating evolution of $e :\rightarrow d$, if and only if $\Delta(\phi, e) < \infty$.*

The weakest precondition for an evolution of a differential action is:

Lemma 6. *The largest set of states for $e :\rightarrow d$ from which an evolution ϕ terminates in a state satisfying q is given by $\Delta(\phi, e) < \infty \wedge q[\phi(\Delta(\phi, e))/X]$.*

Proof. By Corollary 5, the claim is trivial in the non-terminating case, so we assume termination and give a proof for the claim by contradiction. Let $X = X_0 \wedge Y = Y_0$ denote a state from which an evolution ϕ terminates in a state satisfying q, but does not satisfy the given condition. Since ϕ is an evolution, $\phi(0) = X_0$ must hold by Definition 3. Also, by Definition 3 and Corollary 5, the condition $0 < \Delta(\phi, e) < \infty$ holds. Therefore, the values of X at termination are given by $\phi(\Delta(\phi, e))$. Furthermore, the values of Y remain unchanged. Thus, since we assumed that q holds at termination, $q[\phi(\Delta(\phi, e))/X]$ must hold. But then, the given condition holds for the initial state $X = X_0 \wedge Y = Y_0$, which contradicts our initial assumption. Thus, the claim of Lemma 6 holds. □

Corollary 7. *All the states from which some evolution of $e :\rightarrow d$ does not terminate in a state satisfying q are captured by*

$$\exists \phi : C^1.SF(\phi, e, d) \wedge \Delta(\phi, e) > 0 \wedge \neg(\Delta(\phi, e) < \infty \wedge q[\phi(\Delta(\phi, e))/X])$$

The complement of this corollary provides the weakest precondition semantics for the differential action, because it only excludes those states from which some evolution does not terminate in a state satisfying a given postcondition:

Definition 8 (Semantics of the differential action).

$$\begin{aligned} \mathrm{wp}(e :\rightarrow d, q) \hat{=} \forall \phi : C^1.\ & SF(\phi, e, d) \wedge \Delta(\phi, e) > 0 \\ & \Rightarrow \Delta(\phi, e) < \infty \wedge q[\phi(\Delta(\phi, e))/X] \end{aligned}$$

Example 9. A delay of at most four seconds is given by a differential action $C \hat{=} 0 \leqslant c \leqslant 4 :\rightarrow \dot{c} = 1$. To illustrate the use of the weakest precondition semantics, we compute all the states from which all the evolutions of C terminate at $c = 4$. In the computation we use a general solution for the differential relation to simplify the weakest precondition expression. The computation goes:

$$\begin{aligned} & \mathrm{wp}(C, c = 4) \\ \equiv\ & \{\textit{unfolding } \mathrm{wp} \textit{ for } C\} \end{aligned}$$

$$
\begin{aligned}
&\forall \phi : C^1.\ SF(\phi, 0 \leqslant c \leqslant 4, \dot{c} = 1) \wedge \Delta(\phi, 0 \leqslant c \leqslant 4) > 0 \\
&\Rightarrow \Delta(\phi, 0 \leqslant c \leqslant 4) < \infty \wedge (c = 4)[\phi(\Delta(\phi, 0 \leqslant c \leqslant 4))/c] \\
\equiv\ &\{\textit{textual substitution}\} \\
&\forall \phi : C^1.\ SF(\phi, 0 \leqslant c \leqslant 4, \dot{c} = 1) \wedge \Delta(\phi, 0 \leqslant c \leqslant 4) > 0 \\
&\Rightarrow \Delta(\phi, 0 \leqslant c \leqslant 4) < \infty \wedge \phi(\Delta(\phi, 0 \leqslant c \leqslant 4)) = 4 \\
\equiv\ &\{\textit{inst. general solution for } \dot{c} = 1,\ \textit{i.e.},\ \phi : (\lambda c_0 : \mathbb{R}, \tau : \mathbb{R}.c_0 + \tau)\} \\
&\forall c_0 : \mathbb{R}.\ SF(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4,\ \dot{c} = 1) \\
&\wedge \Delta(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4) > 0 \\
&\Rightarrow \Delta(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4) < \infty \\
&\wedge c_0 + \Delta(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4) = 4 \\
\equiv\ &\{\textit{computing } SF(\cdot)\} \\
&\forall c_0 : \mathbb{R}.\ c_0 = c \wedge \Delta(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4) > 0 \\
&\Rightarrow \Delta(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4) < \infty \\
&\wedge c_0 + \Delta(\lambda\tau : \mathbb{R}.\ c_0 + \tau,\ 0 \leqslant c \leqslant 4) = 4 \\
\equiv\ &\{\textit{simplification}\} \\
&\Delta(\lambda\tau : \mathbb{R}.\ c + \tau, 0 \leqslant c \leqslant 4) > 0 \\
&\Rightarrow \Delta(\lambda\tau : \mathbb{R}.\ c + \tau,\ 0 \leqslant c \leqslant 4) < \infty \\
&\wedge c + \Delta(\lambda\tau : \mathbb{R}.\ c + \tau,\ 0 \leqslant c \leqslant 4) = 4 \\
\equiv\ &\{\Delta(\cdot) > 0\ \textit{when}\ 0 \leqslant c < 4\} \\
&0 \leqslant c < 4 \\
&\Rightarrow \Delta(\lambda\tau : \mathbb{R}.\ c + \tau,\ 0 \leqslant c \leqslant 4) < \infty \\
&\wedge c + \Delta(\lambda\tau : \mathbb{R}.\ c + \tau,\ 0 \leqslant c \leqslant 4) = 4 \\
\equiv\ &\{\textit{assuming}\ 0 \leqslant c < 4,\ \Delta(\cdot)\ \textit{is}\ 4 - c\} \\
&0 \leqslant c < 4 \Rightarrow 4 - c < \infty \wedge c + (4 - c) = 4 \\
\equiv\ &\{\textit{simplification}\} \\
&\neg(0 \leqslant c < 4) \vee\ \textit{true}
\end{aligned}
$$

Here, the left term captures all the states where C is disabled, and the right term captures all the states from where any evolution of C terminates in a state satisfying $c = 4$. Thus, we know that, *when enabled*, C terminates always in a state $c = 4$. Similarly, $\mathrm{wp}(C, c = 2)$, which evaluates to $\neg(0 \leqslant c < 4) \vee \mathit{false}$, tells us that an enabled C never terminates in a state $c = 2$.

In the same way, a delay of at most four seconds with at most 1% inaccuracy is modelled by $0 \leqslant c \leqslant 4 :\rightarrow 0.99 \leqslant \dot{c} \leqslant 1.01$.

It is important to realize that the differential action speaks only about observations of evolutions, not of time. For instance, if x denotes a location, $0 \leqslant x \leqslant 4 :\rightarrow \dot{x} = 2$

speaks only about the location, not of the velocity or of time. Therefore, we may only conclude from this differential action that the location of an object evolves, for instance, from $x=0$ to 4. To include an observation of the passage of time we need a clock variable, say, c. Then, $0 \leqslant x \leqslant 4 :\rightarrow \dot{x}=2 \wedge \dot{c}=1$ speaks simultaneous about the location *and* time. From this differential action we may conclude, for instance, that the location of an object evolves from $x=0$ to 4 and simultaneously time passes from $c=0$ to 2. From these two observations we may then conclude, indirectly, that the average velocity must have been 2 m/s.

4.2. Enabledness properties

We shall investigate next some of the properties of the differential action that will be useful later on in this paper. As an indicator that the given semantics satisfies our intuitions, we have:

Theorem 10. *A differential action is enabled only in states where there exists some evolution.*

Proof.

$$
\begin{aligned}
& \mathrm{g}(e :\rightarrow d) \\
\equiv & \{\textit{unfolding } \mathrm{wp} \textit{ for } \mathrm{g}\} \\
& \neg(\forall \phi : C^1 . \neg SF(\phi, e, d) \wedge \Delta(\phi, e) > 0 \\
& \Rightarrow \Delta(\phi, e) < \infty \wedge \textit{ false}[\phi(\Delta(\phi, e))/X]) \\
\equiv & \{\textit{strictness of substitution}\} \\
& \neg(\forall \phi : C^1 .\ \neg(SF(\phi, e, d) \wedge \Delta(\phi, e) > 0)) \\
\equiv & \{\textit{propagation of negation}\} \\
& \exists \phi : C^1 .\ SF(\phi, e, d) \wedge \Delta(\phi, e) > 0 \qquad \square
\end{aligned}
$$

Note, however, that the enabledness of the differential action is always restricted by the evolution guard:

Theorem 11. $\mathrm{g}(e :\rightarrow d) \Rightarrow e$.

Proof. Consider a state $X=X_0 \wedge Y=Y_0$ for which $\mathrm{g}(e :\rightarrow d)$ holds. Then, there exists a ϕ for which by definition $\phi(0)=X_0$. Also, by definition $\Delta(\phi, e)>0 \Rightarrow e[\phi(0)/X, Y_0/Y]$ Thus, for a state $X=X_0 \wedge Y=Y_0$, also e holds. $\square$

Remark 12. The condition $e \Rightarrow \mathrm{g}(e :\rightarrow d)$ does not hold in general. Consider, for instance, a differential action $0 \leqslant x \leqslant 1 :\rightarrow \dot{x}=1$. It is enabled in states $0 \leqslant x<1$, which is not implied by the evolution guard $0 \leqslant x \leqslant 1$. Another example is $-1 \leqslant x \leqslant 1 :\rightarrow \dot{x}=1/x$, for which there exists no evolution in the state $x=0$.

As the following useful theorem shows, we may use enabledness to simplify the evolution guard:

Theorem 13. $\mathrm{wp}(e :\rightarrow d, q) \equiv \mathrm{wp}(\mathrm{g}(e :\rightarrow d) :\rightarrow d, q)$.

Proof. By Theorems 10 and 11, the difference is that e may contain states from which there are no evolutions, whereas $\mathrm{g}(e :\rightarrow d)$ consists only of states from which there are evolutions. However, since the differential relation is the same in both differential actions, by Definition 3 and Theorem 10 the set of evolutions is the same. Hence, to prove the claim, we only need to show that the evolutions of $e :\rightarrow d$ terminate in the same states as the corresponding evolutions of $\mathrm{g}(e :\rightarrow d) :\rightarrow d$, i.e., $\Delta(\phi, e) = \Delta(\phi, \mathrm{g}(e :\rightarrow d))$ for a fixed evolution ϕ. We denote these durations:

$$\delta = \inf\{\tau \mid \tau \geqslant 0 \wedge \neg e[\phi(\tau)/X]\}$$

$$\gamma = \inf\{\tau \mid \tau \geqslant 0 \wedge \neg \mathrm{g}(e :\rightarrow d)[\phi(\tau)/X]\}$$

By Theorem 11 the predicate $\mathrm{g}(e :\rightarrow d) \Rightarrow e$ holds; thus, $\gamma \leqslant \delta$ must hold. Suppose now that $\gamma < \delta$. Then, we have an ε such that $\neg \mathrm{g}(e :\rightarrow d)[\phi(\varepsilon)/X]$ and $\gamma \leqslant \varepsilon < \delta$. This means that in the state $X = \phi(\varepsilon)$ the differential action $e :\rightarrow d$ is disabled. On the other hand, because we assumed $\gamma < \delta$, we know that for $\tau \in [\varepsilon, \delta)$ the function $\phi(\tau)$ does describe an evolution of $e :\rightarrow d$. Thus, $\psi(\tau) = \phi(\tau + \varepsilon)$ describes an evolution of $e :\rightarrow d$ for $\tau \in [0, \delta - \varepsilon)$. But ψ cannot be an evolution of $e :\rightarrow d$, because we just concluded that in the state $X = \phi(\varepsilon)$, which is $X = \psi(0)$, $e :\rightarrow d$ is disabled. Thus, $\delta \leqslant \gamma$. But then, $\gamma = \delta$ holds. This means that $\Delta(\phi, e) = \Delta(\phi, \mathrm{g}(e :\rightarrow d))$. Thus, we have shown that any fixed evolution ϕ has precisely the same duration in both of the differential actions. This, in turn, means that ϕ terminates precisely in the same state in both of the differential actions. Therefore, both of these actions capture precisely the same evolutions. Hence, $\mathrm{wp}(e :\rightarrow d, q) \equiv \mathrm{wp}(\mathrm{g}(e :\rightarrow d) :\rightarrow d, q)$ holds. □

To clarify the intuition of enabledness, we shall give one more example. For $\mathbf{var}\, x, v : \mathbb{R}, \mathbb{R}$, consider $0 \leqslant x \leqslant 100 :\rightarrow \dot{x} = v$. Its enabledness depends on the value of v. For $v > 0$, this differential action is enabled in states $0 \leqslant x < 100$, because all the possible evolutions tend to increase the value of x and that is not possible from the boundary $x = 100$. For $v < 0$, on the other hand, this differential action is enabled in states $0 < x \leqslant 100$, because all the possible evolutions tend to decrease the value of x and that is not possible from the boundary $x = 0$. However, for $v = 0$, this differential action is enabled in all the states $0 \leqslant x \leqslant 100$, because the only possible evolution, $\phi(\tau) = x$, satisfies the evolution guard for a period of time from all of these states. Note, however, that such a differential action never terminates. Still, if the differential action terminates in some state, it does not imply that the differential action is disabled in that state, as explained in the following.

4.3. Termination properties

As expected, termination is influenced by non-determinism:

Lemma 14. *From some initial state the differential action terminates if and only if all its evolutions terminate from that state.*

Proof.

$$
\begin{aligned}
& \mathrm{t}(e :\rightarrow d) \\
\equiv\ & \{\textit{unfolding } \mathrm{wp} \textit{ for } \mathrm{t}\} \\
& \forall \phi : C^1.\ SF(\phi, e, d) \wedge \Delta(\phi, e) > 0 \\
& \Rightarrow \Delta(\phi, e) < \infty \wedge \textit{true}[\phi(\Delta(\phi, e))/X] \\
\equiv\ & \{\textit{property of substitution}\} \\
& \forall \phi : C^1.\ SF(\phi, e, d) \wedge \Delta(\phi, e) > 0 \Rightarrow \Delta(\phi, e) < \infty \qquad \square
\end{aligned}
$$

And, as the following non-trivial theorem shows, termination may occur only at the boundary of an evolution guard:

Theorem 15. *All the terminating evolutions of a differential action terminate at the boundary of the evolution guard.*

Proof. $\Delta(\phi, e) > 0$ holds for an evolution. Also, due to termination the duration is finite, too. Let $\delta = \Delta(\phi, e)$ denote this value. Then, by Definition 2 the following holds:

$$\forall \tau : \mathbb{R} \cap [0, \delta).\ e[\phi(\tau)/X]$$

Furthermore, by Definition 2, as a property of infimum:

$$\neg e[\phi(\delta)/X] \vee \exists \gamma : \mathbb{R} \cap (\delta, \infty). \forall \tau : \mathbb{R} \cap (\delta, \gamma).\ \neg e[\phi(\tau)/X]$$

Thus, because the evolution terminates in the state $X = \phi(\delta)$, there are states in any neighborhood of the termination state that satisfy and do not satisfy e. Hence, the termination state belongs to the boundary of the evolution guard. $\square$

However, due to non-determinism the following remarks are important:

Remark 16. The differential action may well terminate in a state, where it is still enabled. Consider, for instance, $0 \leqslant x \leqslant 1 :\rightarrow \dot{x} \in \{-1, 1\}$. It is enabled in states $0 \leqslant x \leqslant 1$. And, its weakest precondition for a postcondition $x = 1$ computes to $\neg(0 < x \leqslant 1)$. Thus, from the state $x = 0$ this differential action is certain to terminate in the state $x = 1$ where it is still enabled.

Remark 17. The differential action may also terminate in a state from where it started executing. Consider, for instance, $0 \leqslant x \leqslant 2 :\rightarrow \dot{x} \in [-1, 1]$. It is enabled in states $0 \leqslant x \leqslant 2$. From a state $x = 0$ a possible evolution is given by a function $\phi : (\lambda \tau : \mathbb{R}.\ \tau - \tau^2)$, whose derivative is $\dot{\phi} : (\lambda \tau : \mathbb{R}.\ 1 - 2\tau)$. The duration of this evolution is 1. Clearly, during the period $[0, 1]$ both $0 \leqslant \phi(\tau) \leqslant 2$ and $\dot{\phi}(\tau) \in [-1, 1]$ hold. Still, at termination, a state $x = \phi(1)$ is reached, and $\phi(1) = 0$. Thus, $x = 0$ holds at termination, too.

4.4. Conjunctive normal form

The differential action belongs to the conjunctive predicate transformers. All the conjunctive predicate transformers can be written in a normal form [7]: $\{p\};X:=\chi.r$, where p and r are predicates, and X are the model variables.

The conjunctive normal form is obtained from the semantics of the differential action, $\mathrm{wp}(e:\rightarrow d,q)$, as follows. First, we rewrite

$$\forall\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\Rightarrow\Delta(\phi,e)<\infty\wedge q[\phi(\Delta(\phi,e))/X]$$

into

$$\forall\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\Rightarrow\Delta(\phi,e)<\infty$$

$$\wedge\forall\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\Rightarrow q[\phi(\Delta(\phi,e))/X]$$

Here, the latter conjunction is written in a longer, but equivalent, form

$$\forall\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\Rightarrow\Delta(\phi,e)<\infty$$

$$\wedge\forall\chi:(\exists\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\wedge\chi=\phi(\Delta(\phi,e))).\ q[\chi/X]$$

The equivalence is proven by unfolding the quantifications. The obtained expression above is the weakest precondition semantics of the statement:

$$\{\forall\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\Rightarrow\Delta(\phi,e)<\infty\};$$

$$X:=\chi.(\exists\phi:C^1.\ SF(\phi,e,d)\wedge\Delta(\phi,e)>0\wedge\chi=\phi(\Delta(\phi,e)))$$

This is the conjunctive normal form for the differential action.

To summarize, the differential action is conjunctive, and therefore also monotonic [7]. Furthermore, as discussed in [7], we can say directly from the conjunctive normal form of the differential action that it is not strict, disjunctive, or boundedly non-deterministic. Therefore, the differential action is more of a specification statement than a program statement.

5. Hybrid action systems

So far we have introduced individual actions; a complete system is modelled by an *action system* that contains a set of initialized variables and a collection of actions which are chosen repeatedly and non-deterministically for execution. Action systems can be composed in parallel, thus modelling *reactive systems* by their components.

Since the differential action has a semantics similar to the guarded command, we can use it in action systems. Such systems model both discrete-time and continuous-time dynamics, i.e., hybrid systems. However, the notion of parallelism in action systems is based on interleaving which is not useful for handling concurrent evolutions. Therefore, we shall investigate a special form of action systems called *hybrid action systems*, for which we extend the parallel composition to give a more suitable notion of concurrent evolutions.

5.1. Action systems

An action system is an initialized block of the form [2]

$$\mathscr{A} \hat{=} |[\mathbf{var}\, X : T \bullet X := E;\ \mathbf{do}\ A_1 \,[]\, .. \,[]\, A_n\ \mathbf{od}]| : V$$

The expression $\mathbf{var}\, X : T$ declares a set of variables X with types T. Some of the variables in X may be *observable* from outside. They are the *shared variables* and are decorated in the declaration with a superscript asterisk, for instance, x^*. The subset of shared variables is denoted X^*. The variables in X that are not shared, are called the *local variables*. They do not exist outside the action system where they are defined. Variables that do not belong to X, but are used by the actions $A_1..A_n$, must be members of the *imported variables*, V. These variables are assumed to be disjoint from X. Thus, the union of variables X and V forms the state space of $\mathscr{A}$.

The action $X := E$ initializes all the variables X by constants E. Hence, the shared variables are initialized in the action systemswhere they are declared. All action systems initialize synchronously before any other action is taken in one of them. This becomes apparent later, when the parallel composition for action systems is defined.

Typically, an action A_i of $A_1..A_n$ is a guarded command or a differential action. After initialization, an enabled action A_i is selected non-deterministically for execution. There are no fairness assumptions about the selection. The execution of A_i is always atomic, even for a composite or differential action. This means that when, for instance, a differential action is selected and executed, an evolution continues without any interruption till termination. Only then do the other enabled actions have a chance for execution.

The system $\mathscr{A}$ terminates, or converges, when all actions are disabled, $\neg \mathrm{g}(A_1 \,[]\, .. \,[]\, A_n)$. Similarly, $\mathscr{A}$ aborts,or diverges, if any executed action aborts.

Example 18. We give an example that is based on the temperature controller introduced by Jaffe et al. [19]. The task of the controller is to keep the (real valued) temperature t of a reactor tank within an interval $L \leqslant t \leqslant H$. At any time, t rises at rate R_t. When t reaches H, the tank is cooled with a rod which by itself causes t to decrease at rate R_u. The assumption here is that $R_t < R_u$. The tank is cooled till t reaches L. The used rod is then put aside, and reused after D time units. Should t reach H when the rod is unavailable, a complete shutdown of the system is required. The following action system specifies ideal dynamics of the reactor and its control program.

$$\begin{aligned}
\mathscr{R}eactor \hat{=}\ |[\ & \mathbf{var}\, t^*, c^*, u^* : \mathbb{R}, \mathbb{R}, \mathbb{R} \bullet t, c, u := L, D, 0; \\
& \mathbf{do}\ t = H \wedge D \leqslant c - u \rightarrow u := c \\
& \ []\ \ t < H \qquad\qquad :\rightarrow \dot{t} = R_t \wedge \dot{c} = 1 \\
& \ []\ \ u = c \wedge L < t \qquad :\rightarrow \dot{t} = R_t - R_u \wedge \dot{c} = 1 \wedge \dot{u} = 1 \\
& \mathbf{od} \\
]|\ &
\end{aligned}$$

In the action system above, we use auxiliary variables c and u. The variable c measures the elapsed idle time for the rod, and u records the last time when the rod was used.

Thus, if $c = u$ holds, the rod is in use. These variables are initialized so that the rod can be used right away, if necessary. The first action in $\mathscr{R}eactor$ models the decision when the rod is used for cooling. The second action models how the temperature of the reactor evolves without any cooling rods, while the third action models the cooling of the reactor. Note that when a complete shutdown of the system is required, i.e., $t = H \wedge D > c - u \wedge u \neq c$ holds, $\mathscr{R}eactor$ terminates.

5.2. Parallel composition of action systems

Typically, a hybrid system consists of several reactive components [10]: a controller, a plant, and several actuators and sensors. With the action system formalism we can model each of these components separately. The overall dynamics of the entire system is then obtained via parallel composition.

The parallel composition of action systems, as given by Back [2], is formalized as follows. Consider the two action systems, where X and Y are disjoint variables that can be either local or shared:

$$\mathscr{A} \hat{=} |[\mathbf{var}\, X : T \bullet X := E;\ \mathbf{do}\ A_1 [] .. [] A_m\ \mathbf{od}]| : V$$

$$\mathscr{B} \hat{=} |[\mathbf{var}\, Y : U \bullet Y := I;\ \mathbf{do}\ B_1 [] .. [] B_n\ \mathbf{od}]| : W$$

Then, the parallel composition is given as

$$\begin{aligned} \mathscr{A} \| \mathscr{B} = |[&\mathbf{var}\, X, Y : T, U \bullet X, Y := E, I; \\ &\mathbf{do}\ A_1 [] .. [] A_m [] B_1 [] .. [] B_n\ \mathbf{od} \\ &]| : (V \cup W) - (X \cup Y) \end{aligned}$$

Thus, the parallel composition combines the state spaces of the two action systems, merging the shared variables and keeping the local variables distinct. Imported variables in $V \cup W$, that are defined in either of the action systems, i.e., $(X \cup Y)$, are no longer imported. The actions from the action systems are merged with non-deterministic choice, modelling parallelism by interleaving.

The behavior of the parallel composition depends on how the individual action systems, the *reactive components*, interact with each other via the shared variables, $(X^* \cup Y^*)$. For instance, a reactive component does not terminate by itself. Termination is a global property of the composed action system [2].

Since the initializations are well defined, and the non-deterministic choice is both associative and commutative, the parallel composition is also associative and commutative. Therefore, the meaning of several parallel composed action systems can be unfolded in any order without affecting the result. Also, the definition of the parallel composition allows the use in reverse; a large action system can be decomposed into several smaller systems.

Example 19. This example extends Example 18 by adding a new rod to the reactor. The new rod is used when the other rod cannot be reused but the reactor tank needs to be cooled. The new rod causes t to decrease at rate R_v. The following action system

describes the ideal dynamics of the new rod. The variable v records theprevious time when the new rod was used:

$$
\begin{array}{l}
\mathscr{R}od \mathrel{\hat{=}} \\
|[\textbf{var } v^* : \mathbb{R} \bullet v{:=}0; \\
\quad \textbf{do } t = H \wedge c \neq u \wedge D > c-u \wedge D \leqslant c-v \rightarrow v{:=}c \\
\quad [] \ v = c \wedge L < t \qquad\qquad :\rightarrow \dot{t} = R_t - R_v \wedge \dot{c} = 1 \wedge \dot{v} = 1 \\
\quad \textbf{od} \\
]| : t, c, u
\end{array}
$$

The overall dynamics is obtained by unfolding the parallel composition:

$$
\begin{array}{l}
\mathscr{R}eactor \| \mathscr{R}od \equiv \\
|[\textbf{var } t^*, c^*, u^*, v^* : \mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R} \bullet t, c, u, v{:=}L, D, 0, 0; \\
\quad \textbf{do } t = H \wedge D \leqslant c-u \qquad\qquad \rightarrow u{:=}c \\
\quad [] \ t < H \qquad\qquad :\rightarrow \dot{t} = R_t \wedge \dot{c} = 1 \\
\quad [] \ u = c \wedge L < t \qquad\qquad :\rightarrow \dot{t} = R_t - R_u \wedge \dot{c} = 1 \wedge \dot{u} = 1 \\
\quad [] \ t = H \wedge c \neq u \wedge D > c-u \wedge D \leqslant c-v \rightarrow v{:=}c \\
\quad [] \ v = c \wedge L < t \qquad\qquad :\rightarrow \dot{t} = R_t - R_v \wedge \dot{c} = 1 \wedge \dot{v} = 1 \\
\quad \textbf{od} \\
]|
\end{array}
$$

However, as this example shows, an interleaving definition for parallel differential actions forces us to replicate some of the continuous-time dynamics of one component to all other components. The reason for this is that interleaving does not allow truly concurrent evolutions. For instance, in the example above we specify how the rod and the reactor tank behave together, $\dot{t} = R_t - R_u$, in order to describe the effect of the rod. Clearly, this is undesirable because now the specification for the new rod can only be used together with the reactor tank specification given in Example 18, not with other reactor tank specifications.

Motivated by this example we investigate an extension to the composition that allows concurrent evolutions.

5.3. Concurrent evolutions

In the following we shall consider only evolutions of differential actions with *closed evolution guards*, i.e., the evolution guards capture closed sets of states. We do so to ensure that sequentially composed differential actions specify seamless evolutions. Consider, for instance, sequentially composed differential actions $0 \leqslant x < 2 :\rightarrow \dot{x} = 1$; $2 < x < 4 :\rightarrow \dot{x} = 1$. From an initial state $x = 0$, the former differential action terminates in a state $x = 2$. However, as the latter differential action is enabled only in the states $2 < x < 4$, it cannot be executed, and such a composition is not reasonable. The problem does not appear with closed evolution guards $0 \leqslant x \leqslant 2 :\rightarrow \dot{x} = 1$; $2 \leqslant x \leqslant 4 :\rightarrow \dot{x} = 1$.

Here, from an initial state $x = 0$, the former differential action terminates in a state $x = 2$, where the latter differential action is enabled and, thus, evolves the value of x to 4. Note that the enabledness is always restricted by the evolution guard, as highlighted by Theorem 11.

For the formalization of concurrency, we first clarify, what is a closure of a predicate. Consider model variables $\mathbf{var}\, X : T$, where type T forms a metric space. Then, by convention, an *open sphere* around a state $\sigma : T$ with a positive radius $\delta : \mathbb{R} \cap (0, \infty)$ is defined as

$$B(\sigma, \delta) \hat{=} \{\gamma : T \mid \|\sigma - \gamma\| < \delta\}$$

A predicate $p : PRED(X)$ is said to be *open*, i.e., describe an open set, if every state σ belonging to p can be enlarged to an open sphere that is also fully contained in p, i.e.,

$$OPEN(p) \hat{=} \forall \sigma : T.\, p[\sigma/X] \Rightarrow \exists \delta : \mathbb{R} \cap (0, \infty).\ \forall \gamma : B(\sigma, \delta).\ p[\gamma/X]$$

Correspondingly, p is said to be *closed* if its complement describes an open set, i.e.,

$$CLOSED(p) \hat{=} OPEN(\neg p)$$

In addition, the *boundary* of p, denoted by ∂p, is the set of states σ for which any sphere $B(\sigma, \delta)$ contains states that both do and do not belong to p:

$$\partial p \hat{=} \exists \sigma : T.\ X = \sigma \\ \wedge \forall \delta : \mathbb{R} \cap (0, \infty).\ \exists \alpha : B(\sigma, \delta),\ \beta : B(\sigma, \delta).\ p[\alpha/X] \wedge \neg p[\beta/X]$$

With these standard definitions, the *closure* of p, denoted by $\bar{p}$, is the union of itself and its boundary:

$$\bar{p} \hat{=} p \vee \partial p$$

And, the *interior* of p, denoted by p°, is obtained by excluding its boundary from itself:

$$p^\circ = p \wedge \neg \partial p$$

Hence, $p^\circ \Rightarrow p \Rightarrow \bar{p}$ holds trivially. And, particularly, the following properties hold: $CLOSED(\bar{p})$, $CLOSED(\partial p)$, and $OPEN(p^\circ)$. Furthermore, we know that a closed predicate p is its own closure, i.e., $CLOSED(p) \Rightarrow (p \Leftrightarrow \bar{p})$, and an open predicate describes its own interior, i.e., $OPEN(p) \Rightarrow (p \Leftrightarrow p^\circ)$.

A key observation about evolutions is that the state is changed by a rate defined by the differential quotients. This corresponds to incrementing the current value by the (infinitesimal) differentials. Thus, it seems reasonable to view concurrent evolutions as similar to concurrent incrementing loops. Then, the semantics corresponds to addition of differentials, and the two rate changes are superimposed. An illustration of this idea is a person walking through a corridor, where a conveyor belt is the floor. Suppose that the person is walking with a constant velocity of 0.5 m/s and

the conveyor belt is moving to the same direction with 1 m/s. To an outside observer it seems that person is walking on the conveyor belt with a constant velocity of 1.5 m/s, and when the walker steps down from the conveyor belt, the walking velocity drops to 0.5 m/s. This phenomenon, concurrent evolutions, is caused by two independent factors that operate temporarily on the same variable: the velocity of the walker.

In order to define parallel composition of differential actions, cf. [28], we define *linear composition* of partially defined functions: The functions are added on their common domain, whereas in the remaining domains the functions retain their original value.

For a start, consider two deterministic differential equations, $\dot{x} = 0.5$ and 1. A superimposition that corresponds to the conveyor belt example above yields $\dot{x} = 0.5 + 1$. Suppose the former of these two differential equations is non-deterministic, e.g., $0.1 \leqslant \dot{x} \leqslant 0.6$. A similar kind of superimposition of this and $\dot{x} = 1$ should contain any possible deterministic superimposition of the two differential (in)equations. In other words, the superimposition should consist of all the superimpositions of $\dot{x} = \alpha$ and 1, where $0.1 \leqslant \alpha \leqslant 0.6$. This is given by $\exists\, \alpha, \beta : 0.1 \leqslant \alpha \leqslant 0.6 \wedge \beta = 1.\ \dot{x} = \alpha + \beta$ that yields $1.1 \leqslant \dot{x} \leqslant 1.6$. The example leads to the following generalization.

Definition 20 (Linear composition). Consider a general type T, model variables **var** X, $Y : \mathbb{R}^n, T$, predicates $e_i : PRED(X, Y)$ and $o_i : PRED(X, Y)$ modelling evolution guards, predicates $h_i : PRED(X, \dot{X}, Y)$ and $k_i : PRED(X, \dot{X}, Y)$ modelling differential relations, and non-deterministically composed differential actions:

$$DH \hat{=} \overline{e_1} :\!\rightarrow h_1 \,[]\, .. \,[]\, \overline{e_n} :\!\rightarrow h_n$$

$$DK \hat{=} \overline{o_1} :\!\rightarrow k_1 \,[]\, .. \,[]\, \overline{o_m} :\!\rightarrow k_m$$

Then, the linear composition denoted by $DH \oplus DK$ is defined as

$$\begin{array}{lll}
[]_{i=1..n,\ j=1..m}. & \overline{\mathrm{g}(DH_i) \wedge \mathrm{g}(DK_j)} & :\!\rightarrow \exists \alpha : \mathbb{R}^n,\ \beta : \mathbb{R}^n. \\
 & & h_i[\alpha/\dot{X}] \wedge k_j[\beta/\dot{X}] \wedge \dot{X} = \alpha + \beta \\
[]_{i=1..n}. & \overline{\mathrm{g}(DH_i) \wedge \neg \mathrm{g}(DK)} & :\!\rightarrow h_i \\
[]_{j=1..m}. & \overline{\neg \mathrm{g}(DH) \wedge \mathrm{g}(DK_j)} & :\!\rightarrow k_j
\end{array}$$

It is easy to check that linear composition is both associative and commutative.

Example 21. A person walking with a constant velocity of 1.5 m/s through a corridor that is 12 m long is modelled by $0 \leqslant x \leqslant 12 :\!\rightarrow \dot{x} = 1.5$, where x denotes the location in the corridor. An 8 m long conveyor belt residing in the middle of the corridor and moving objects with some velocity between 0.2 and 0.8 m/s against the walker's direction is modelled by a differential action $2 \leqslant x \leqslant 10 :\!\rightarrow\ -0.8 \leqslant \dot{x} \leqslant -0.2$. Walking across the corridor is given by a linear composition:

$$0 \leqslant x \leqslant 12 :\!\rightarrow \dot{x} = 1.5 \oplus 2 \leqslant x \leqslant 10 :\!\rightarrow -0.8 \leqslant \dot{x} \leqslant -0.2$$

It unfolds to

$$\begin{array}{ll}
\overline{0 \leqslant x \leqslant 12 \wedge 2 \leqslant x \leqslant 10} & :\rightarrow \exists \alpha : \mathbb{R},\ \beta : \mathbb{R}. \\
 & \quad \alpha = 1.5 \wedge -0.8 \leqslant \beta \leqslant -0.2 \wedge \dot{x} = \alpha + \beta \\
[]\overline{0 \leqslant x \leqslant 12 \wedge \neg(2 \leqslant x \leqslant 10)} & :\rightarrow \dot{x} = 1.5 \\
[]\overline{\neg(0 \leqslant x \leqslant 12) \wedge 2 \leqslant x \leqslant 10} & :\rightarrow -0.8 \leqslant \dot{x} \leqslant -0.2
\end{array}$$

This simplifies to

$$\begin{array}{ll}
2 \leqslant x \leqslant 10 & :\rightarrow 0.7 \leqslant \dot{x} \leqslant 1.3 \\
[]0 \leqslant x \leqslant 2 \vee 10 \leqslant x \leqslant 12 & :\rightarrow \dot{x} = 1.5 \\
[]\mathit{false} & :\rightarrow -0.8 \leqslant \dot{x} \leqslant -0.2
\end{array}$$

Here, the last action is never enabled, and therefore, it can be left out. Then, to clarify what happens in a state $x = 10$, we use Theorem 13 and strengthen the evolution guards of the two remaining differential actions with their enabledness. Thus, we obtain

$$2 \leqslant x < 10 :\rightarrow 0.7 \leqslant \dot{x} \leqslant 1.3 \quad [] \quad 0 \leqslant x < 2 \vee 10 \leqslant x < 12 :\rightarrow \dot{x} = 1.5$$

Here, the leftmost action models the movement of the walker on the conveyor belt, and the rightmost action models the movement of the walker outside the conveyor belt.

5.4. Hybrid parallelism

In order to allow concurrent evolutions together with interleaving, we identify a special class of action systems. In doing so we follow closely the approach proposed by Rönkkö and Ravn [28]. Basically, the dynamics of a hybrid system is an alternation of discrete- and continuous-time dynamics [28]:

Definition 22 (Hybrid alternation). The hybrid alternation of non-deterministically composed conventional actions H and non-deterministically composed differential actions DH is a prioritized iteration:

$$\textbf{alt}\, H\ \textbf{with}\, DH \mathrel{\hat{=}} \textbf{do}\ H[]\neg \mathrm{g}(H) \rightarrow DH\ \textbf{od}$$

Here, by convention, $\textbf{alt}\, H$ is dropped when $\mathrm{g}(H) \equiv \mathit{false}$, and $\textbf{with}\, DH$ is dropped when $\mathrm{g}(DH) \equiv \mathit{false}$.

Here, the evolutions of DH may proceed only after all the enabled jumps of H have been processed. However, once an evolution is engaged it continues without any interruption till the corresponding differential action terminates. Only after that jumps or other evolutions may occur. The case, when an evolution is followed by another evolution, is called a *switch* [10]. The alternation terminates when all the actions become disabled, i.e., when any state of $\neg \mathrm{g}(H\,[]\,DH)$ is reached. Hence, all the states from where the alternation terminates are given by $\mathrm{t}(\textbf{alt}\, H\ \textbf{with}\, DH)$. In summary, the hybrid alternation model inserts discrete-time computations between evolutions. Rönkkö and Ravn [27] have also discussed alternative definitions of hybrid alternation, their strengths and weaknesses.

When modelling hybrid dynamics, there are three important anomalies that may arise independently of the used formalism: *hibernation*, *timelock* [9], and *Zeno-dynamics* [1]. Note, however, that a model exhibiting some of these anomalies is not necessarily unsound. Nevertheless, such a model may behave unexpectedly. We shall now briefly describe these anomalies in detail.

In hibernation the model of continuous-time dynamics prevents the execution of the model of discrete-time dynamics. As a result, time progresses in the model, but some of the discrete-time dynamics may never become enabled. In our formalism, this is due to atomicity. The execution of a differential action is atomic, whereby it cannot be interrupted once engaged. This means that during an evolution some other actions may become temporarily enabled. Still, atomicity prevents their execution. Such actions are hibernating. Consider, for instance, $x:=0;\ \mathbf{alt}\, x=1\rightarrow x:=2\ \mathbf{with}\ 0\leqslant x<3:\rightarrow \dot{x}=1$. Here, $x=1\rightarrow x:=2$ hibernates during the evolution, and is never executed although the evolution of x passes through 1. Formally, hibernation in $\mathbf{alt}\, H\ \mathbf{with}\, DH$ may occur in states $\mathrm{g}(H)\wedge \mathrm{g}(DH)$.

Timelock is the opposite of hibernation; in timelock the model of discrete-time dynamics prevents altogether the execution of the model of continuous-time dynamics [9]. Thus, when timelock occurs, time in the model stops, and discrete-time dynamics remain enabled forever after. An example of timelock is $\mathbf{alt}\, x\leqslant 1\rightarrow x:=1\ \mathbf{with}\ 0\leqslant x\leqslant 3:\rightarrow \dot{x}=1$. When this hybrid alternation is executed in an initial state $x=0$, although both actions are enabled, only the guarded command is executed due to priority. After the execution, the state $x=1$ is reached. However, in this state, the guarded command remains enabled and is executed again. Hence, the differential action will never be executed and time stops causing the hybrid alternation to timelock. Formally, the states where timelock occurs are captured by $\neg\mathrm{t}(\mathbf{do}\, H\ \mathbf{od})$.

Zeno-dynamics is an intermediate of these two anomalies; in Zeno-dynamics, although time keeps on progressing during the execution of a model, there is an upper bound for the progress of time. For model variables $\mathbf{var}\, c,d:\mathbb{R},\mathbb{R}$, consider a hybrid alternation $\mathbf{alt}\ c=d\rightarrow d:=d/2\ \mathbf{with}\, d<c:\rightarrow \dot{c}=-1$. It has Zeno-dynamics from an initial state $c=1\wedge d=1/2$. Here, c measures the progress of time downwards. Clearly, c keeps on decreasing but there is a bound for it. Namely, c never reaches 0. Thus, we know that time keeps progressing but not beyond an upper bound, one time unit. Verifying the existence of Zeno-dynamics is not as easy as verifying the existence of the other two anomalies. Still, a necessary condition for a hybrid alternation $\mathbf{alt}\, H\ \mathbf{with}\, DH$ to have Zeno-dynamics is $\neg\mathrm{t}(\mathbf{alt}\, H\ \mathbf{with}\, DH)$. A more detailed proof of Zeno-dynamics requires to show that the value of an additional variable δ remains bounded in $\delta:=0;\ \mathbf{alt}\, H\ \mathbf{with}\, DH\oplus true:\rightarrow \dot{\delta}=1$.

Based on the hybrid alternation, we define a *hybrid action system* [28]:

Definition 23 (Hybrid action system). A hybrid action system is a block of the form

$$\mathscr{H}\ \hat{=}|[\mathbf{var}\, X:T\bullet X:=E;\ \mathbf{alt}\, H\ \mathbf{with}\, DH]|:V$$

Since the continuous-time dynamics is separated from the discrete-time dynamics in a hybrid action system, we can extend the parallel composition to

compose linearly the continuous-time dynamics while interleaving the discrete-time dynamics [28]:

Definition 24 (Parallel composition). Consider the two hybrid action systems:

$$\mathscr{H} \mathrel{\hat{=}} |[\mathbf{var}\, X : T \bullet X{:=}E;\ \mathbf{alt}\, H\ \mathbf{with}\, DH]| : V$$

$$\mathscr{K} \mathrel{\hat{=}} |[\mathbf{var}\, Y : U \bullet Y{:=}I;\ \mathbf{alt}\, K\ \mathbf{with}\ DK]| : W$$

The parallel composition of these systems is

$$\begin{aligned}\mathscr{H} \| \mathscr{K} = |[&\mathbf{var}\, X, Y : T, U \bullet X, Y{:=}E, I;\\ &\mathbf{alt}\, H\ []K\ \mathbf{with}\ DH \oplus DK\\ &]| : (V \cup W) - (X \cup Y)\end{aligned}$$

The extended parallel composition is associative and commutative, because both interleaving and linear composition are associative and commutative. Therefore, the parallel composition of several hybrid action systems can be unfolded in any order without affecting the result. However, note that the parallel composition of hybrid action systems may exhibit anomalies, although the individual component systems themselves would be free of anomalies.

Example 25. In this example we show that the reactor tank specification of Example 18 can be composed of two concurrent hybrid specifications. One of these specifications is for the reactor tank:

$$\begin{aligned}\mathscr{T}ank \mathrel{\hat{=}} |[&\mathbf{var}\, t^*, c^* : \mathbb{R}, \mathbb{R} \bullet t, c{:=}L, D;\\ &\mathbf{with}\ L \leqslant t \leqslant H :\rightarrow \dot{t} = R_t \wedge \dot{c} = 1\\ &]|\end{aligned}$$

And, the other is for the rod:

$$\begin{aligned}\mathscr{R}od \mathrel{\hat{=}} |[&\mathbf{var}\, u^* : \mathbb{R} \bullet u{:=}0;\\ &\mathbf{alt}\quad t = H \wedge D \leqslant c - u\ \rightarrow u{:=}c\\ &\mathbf{with}\ u = c \wedge L \leqslant t \leqslant H\ :\rightarrow \dot{t} = -R_u \wedge \dot{u} = 1\\ &]| : t, c\end{aligned}$$

The overall dynamics of the reactor tank is given by the parallel composition $\mathscr{R}eactor = \mathscr{T}ank \parallel \mathscr{R}od$:

$$\begin{aligned}\mathscr{R}eactor = |[&\mathbf{var}\, t^*, c^*, u^* : \mathbb{R}, \mathbb{R}, \mathbb{R} \bullet t, c, u{:=}L, D, 0;\\ &\mathbf{alt}\quad t = H \wedge D \leqslant c - u\ \rightarrow u{:=}c\\ &\mathbf{with}\ L \leqslant t \leqslant H \wedge u = c\ :\rightarrow \dot{t} = R_t - R_u \wedge \dot{c} = 1 \wedge \dot{u} = 1\\ &\quad []\quad L \leqslant t \leqslant H \wedge u \neq c\ :\rightarrow \dot{t} = R_t \wedge \dot{u} = 1\\ &]|\end{aligned}$$

As required by the informal description of the system, also the derived specification, $\mathscr{R}eactor$, terminates when the temperature of the tank reaches the upper limit and the rod cannot be used again, i.e., $t = H \wedge D > c - u \wedge u \neq c$ holds. Note that, unlike in Example 19, here the specification of $\mathscr{R}od$ has no replicated dynamics of $\mathscr{T}ank$. Thus, the $\mathscr{R}od$ specification can be used with other specifications for reactor tanks. The parallel composition gives then the overall system specification.

In this example all the hybrid action systems are free of anomalies. In particular, both $\mathscr{T}ank$ and $\mathscr{R}od$ are free of timelock and Zeno-dynamics, since their execution always terminates for $R_u > 0$ and $R_t > 0$.

It should be noted that the linear composition is by no means the only alternative for concurrent composition of evolutions. However, the linear composition exhibits simple, intuitive, and frequently needed properties. This was also the reason why we chose to investigate the linear composition as a form of concurrent composition in this paper. And, as for the algebraic properties, the composition operator does not necessarily have to be commutative, either. This is in particular the case, if the composition should be concurrently prioritized.

6. Stepwise development

In Example 25 above, we illustrated how a hybrid system is modelled by specifying its parallel components separately and then unfolding them into a single hybrid action system. However, when the system is made complex by many detailed and intricate discrete actions over shared variables, parallel composition does not offer any assistance. In order to manage such complexity one wishes to specify the overall dynamics in an abstract manner, and then add details while maintaining specific safety and liveness properties of the abstract system, i.e., *correctness properties*. This stepwise development is formalized for action systems in the *refinement calculus* [7]. In the following, we investigate how the refinement techniques carry over to hybrid action systems. For brevity, we limit the investigation to closed systems that have observable variables, but do not react with other systems. The stepwise development in a parallel setting follows then from that in a closed setting, because the meaning of a parallel composed system is given as an unfolding to a closed system.

6.1. Weak simulation of action systems

We shall consider in this paper a general development step where an abstract action system $\mathscr{A}$ is shown to be *weakly simulated* [2] by a concrete action system $\mathscr{C}$. Weak simulation allows increase of determinism while introducing details and, yet, guarantees preservation of both safety and liveness properties. Informally, $\mathscr{C}$ has the same jumps under the corresponding circumstances on the values of the shared variables as $\mathscr{A}$; however, $\mathscr{C}$ may have some additional, intermediate, jumps on the values of the local variables that are completely missing from $\mathscr{A}$. Also, since aborting is undesirable

behavior in a reactive setting, if $\mathscr{A}$ aborts from some state, $\mathscr{C}$ may do anything from that state. But, most importantly, if $\mathscr{A}$ terminates in some state, so does $\mathscr{C}$.

In general, $\mathscr{A}$ and $\mathscr{C}$ have different state spaces and a relation is used for describing the correspondence between the states. Then, the weak simulation proof basically amounts to showing that the correspondence relation holds between the initial states of the models, between all the states traversed by $\mathscr{C}$ and some states traversed by $\mathscr{A}$, and between all the termination states. When proving that the correspondence relation holds between the traversed states, we use the weakest precondition semantics of actions. It allows reasoning on the level of actions rather than on the level of states.

To show the correspondence between the dynamics of the abstract and the concrete actions we use *data refinement* [2,7]. It is a special form of refinement that allows the use of a *refinement relation* [2]. It is a predicate over all the model variables, and it is used for relating all the states of the concrete model to some states in the abstract model; thus, allowing increase of determinism. In addition, the refinement relation may also contain invariants over the values of any model variables. Such invariants capture the safety properties to be preserved.

Informally, a refinement relation r holds between an abstract action A and a concrete action C, if C can reach the same states, related by r, from at least the same initial states, related by r, as A. Then, we say that A is *refined* by C, and we denote it by $A \sqsubseteq_r C$. Thus, when A is refined by C, we may replace A with C in context of any other actions and still preserve the overall dynamics. In particular, this means that **abort** is refined by anything, since it does not reach any state. Data refinement of actions is defined [2]:

Definition 26. Let A be an action over local variables $X : T_X$ and shared variables $Z : T_Z$, and C be an action over Z and some local variables $Y : T_Y$. Then, the refinement relation is $r : PRED(X, Y, Z)$, and $A \sqsubseteq_r C$ is defined:

$$\forall q : PRED(X, Z).\ r \wedge \mathrm{wp}(A, q) \Rightarrow \mathrm{wp}(C, \exists X : T_X.\ r \wedge q)$$

In the definition, the existential quantification is used for relating every concrete state to some corresponding abstract state; thus, allowing increase of determinism. Note that in the absence of local variables X the existential quantification is superfluous and the postcondition is written as $r \wedge q$.

The presented data refinement condition for actions is used in the proof obligations for weak simulation. As already mentioned, such proof obligations must also consider the initialization, possible additional actions operating on newly introduced local variables, and the termination. The proof obligations for weak simulation are defined [2]:

Definition 27. Consider a refinement relation r, and the two action systems:

$$\mathscr{A} \hat{=} |[\,\mathbf{var}\ Z^*, X : ZT, XT \bullet Z, X := ZI, XI;\ \ \mathbf{do}\ A\ \mathbf{od}\,]|$$

$$\mathscr{C} \hat{=} |[\,\mathbf{var}\ Z^*, Y : ZT, YT \bullet Z, Y := ZI, YI;\ \ \mathbf{do}\ C [] B\ \mathbf{od}\,]|$$

Then, $\mathscr{A}$ is weakly simulated by $\mathscr{C}$, denoted by $\mathscr{A} \preccurlyeq_r \mathscr{C}$, if the following proof obligations are shown to hold:

(i) the initializations do not contradict the refinement relation, i.e., $r[ZI/Z, XI/X, YI/Y]$
(ii) the abstract actions A are refined by the concrete actions C, i.e., $A \sqsubseteq_r C$
(iii) the introduced intermediate actions B refine stuttering in $\mathscr{A}$, i.e., $\mathbf{skip} \sqsubseteq_r B$
(iv) the introduced actions are self-disabling, i.e., $r \Rightarrow \mathrm{t}(\mathbf{do}\, B\, \mathbf{od})$
(v) $\mathscr{C}$ does not terminate, unless $\mathscr{A}$ terminates as well, i.e., $r \wedge \mathrm{g}(A) \Rightarrow \mathrm{g}(C \,[]\, B)$

Note that the proof obligation (i) requires a *witness* to the refinement relation. Therefore, a refinement relation cannot be *false*. Also, the proof obligation (iv) ensures that the introduced intermediate dynamics eventually terminates and, hence, preserve liveness properties. The proof obligations above are *transitive* [2], whereby we may first prove that some intermediate action system simulates weakly an abstract action system and, then, continue to show that another action system simulates weakly the intermediate action system. At the end, when we have shown a sequence of weak simulations, we know by transitivity that also the final action system in this sequence preserves the safety and liveness properties of the original abstract model.

Although the proof obligations above are compact, the related proofs as such may well turn out be laborious. However, there are some basic proof techniques that help in manufacturing the proofs. In this paper we consider the following results that help in proving obligations (ii) and (iii) [2]:

Theorem 28. *Let A and C be non-deterministically composed actions, and r be a refinement relation. Then, to prove $A \sqsubseteq_r C$, we may alternatively prove that each of the non-deterministically composed actions in C refine some of the actions in A.*

Proof. This property is due to use of implication in Definition 26 and use of conjunction in the semantics of non-deterministic choice. □

Theorem 29. *Consider guarded commands $gA \to sA$ and $gC \to sC$. Then, $gA \to sA \sqsubseteq_r gC \to sC$ can be proven in two steps* [2]:

(i) *the guard is not weakened, i.e., $r \wedge gC \Rightarrow gA$*
(ii) *the action body is refined in the context of the new guard, i.e., $\{gC\}; sA \sqsubseteq_r sC$*

Proof. This property is due to use of implication in the refinement condition, and use of implication in the semantics of the guarded command. □

6.2. Weak simulation of hybrid action systems

Since data refinement, Definition 26, uses the weakest precondition semantics, the refinement may also involve differential actions. However, one should be clear about the meaning of such a refinement. In principle, the evolutions captured by a differential action are atomic transitions from an initial state to a final state. As the refinement only considers an atomic state change, the refinement involving a differential action does not directly speak of the evolutions. Consequently, the refinement involving a

differential action does not speak of the implicit time, either. However, the refinement does speak of the observations in the initial and final state of an evolution. Thus, if such an observation refers to passage of time via a clock variable, the refinement does also speak of time, indirectly. Hence, the following remark:

Remark 30. When some action is refined by a differential action, it does not mean that the refinement relation holds during an evolution. Rather, because an evolution is atomic, it means that the refinement relation holds at the termination of an evolution, if it held initially. In other words, only the input–output dynamics of actions is considered.

Since refinement relation is used for expressing safe states:

Remark 31. Data refinement in our framework preserves safety in the weak sense: an evolution starting from a safe state may traverse unsafe states, but it terminates always in a safe state.

Consequently, a differential action may refine either a conventional action or another differential action. In addition, a differential action may also be refined by a conventional action. However, the need for this latter case is arguable. When a conventional action is refined by a differential action, the refinement is used for inserting evolutions in between state changes. Then, jumps are transformed into evolutions, and the conventional action is said to be *continualized* [29] by the differential action, or, conversely, the conventional action is a *discretization* [29] of the differential action. When one differential action is refined by another, the refinement is used for introducing details to an existing evolution. Then, the abstract differential action is said to be *continuously transformed* by the concrete differential action.Continuous transformation is needed, for instance, in refining drifting timers of an abstract specification by more accurate timers. It is important to notice that refinement speaks only of observations about the evolutions. Therefore, continuous transformation does not preserve timing properties, unless the differential action speaks of the passage of time as well.

Since the semantics of a hybrid alternation is that of a prioritized iteration, a hybrid action system is just a special case of an action system. Moreover, the weak simulation proof obligations in Definition 27 require only conjunctivity from the iterated actions [2]. Since the differential action is conjunctive, the weak simulation proof obligations apply also to hybrid action systems. Thus, the weak simulation techniques of action systems as such carry over to the hybrid action systems. However, as pointed out in Remark 31, weak simulation in a hybrid setting preserves safety in the weak sense.

Due to the semantics of hybrid alternation a useful technique is to unfold the hybrid alternation before proceeding with the proofs. Also, due to the semantics of the differential action and the form of the proof obligations, another useful technique is to simplify the weakest precondition expression for a differential action under the assumption that the refinement relation holds. We shall illustrate both of these techniques with an example, next.

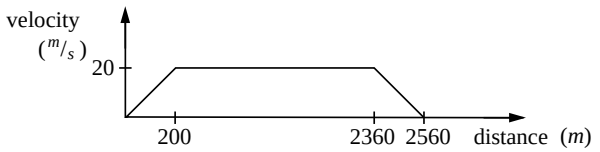

Fig. 4. Velocity during the journey.

6.3. Example: travelling train

We use a simple train example [29] to illustrate use of weak simulation proof obligations. In this example a train travels a distance of 2560 m. It starts by accelerating to travelling velocity, 20 m/s, which it then keeps for the most of the journey. At the end, the train decelerates to a full stop. Fig. 4 depicts how the velocity changes during the journey.

We develop now a hybrid action system modelling the travelling train. We start with a discrete model describing only the state changes, and add the details of continuous-time dynamics in a stepwise manner.

Initial specification: The hybrid action system below is the initial specification describing only the discrete state changes, the jumps from one event to another, during the journey.

$$
\begin{aligned}
&\mathscr{A}Train \mathrel{\hat{=}} |[\mathbf{var}\, x^* : \mathbb{R} \bullet x{:=}0;\\
&\quad \mathbf{alt}\ 0 \leqslant x < 200 \quad \rightarrow x{:=}200\\
&\quad [] \ \ 200 \leqslant x < 2360 \ \rightarrow x{:=}2360\\
&\quad [] \ \ 2360 \leqslant x < 2560 \rightarrow x{:=}2560\\
&\quad]|
\end{aligned}
$$

Here, x denotes the travelled distance, which is observable to outside. The initialization puts the train at the very beginning. The first action models the acceleration phase, the second action models the phase where the train travels with a constant velocity, and the last action models the deceleration phase. The hybrid action system terminates when the train has travelled the whole distance of 2560 m.

Continualization: As the first step we continualize the initial specification by connecting the discrete-time state changes with evolutions. In this step we take into consideration only the direction of the movement, not the velocity. Thus, we obtain the following hybrid action system, where ε is a fixed positive real valued constant:

$$
\begin{aligned}
&\mathscr{C}Train \mathrel{\hat{=}} |[\mathbf{var}\, x^* : \mathbb{R} \bullet x{:=}0;\\
&\quad \mathbf{with}\ 0 \leqslant x < 200 \quad :\rightarrow \varepsilon \leqslant \dot{x}\\
&\quad [] \ \ 200 \leqslant x < 2360 \ :\rightarrow \varepsilon \leqslant \dot{x}\\
&\quad [] \ \ 2360 \leqslant x < 2560 :\rightarrow \varepsilon \leqslant \dot{x}\\
&\quad]|
\end{aligned}
$$

Since $\mathscr{A}Train$ has only an **alt** part and $\mathscr{C}Train$ has only a **with** part, the corresponding hybrid alternations unfold to ordinary iterations without priorities. Therefore, the weak

simulation proof obligations of Definition 27 apply to these hybrid action systems as such. Moreover, as neither of these hybrid action systems have local variables, the refinement relation is *true*. Now we show that the proof obligations hold for the two hybrid action systems, i.e.,

$$\mathscr{A}Train \preceq_{true} \mathscr{C}Train$$

In this case the proof obligation (i) is $true[0/x]$, which holds trivially.

To show proof obligation (ii) we use Theorem 28, and show that it holds for each pair of actions. For brevity, we show here only the proof for the first action pair:

$$0 \leqslant x < 200 \rightarrow x{:=}200 \sqsubseteq_{true} 0 \leqslant x < 200 :\rightarrow \varepsilon \leqslant \dot{x}$$

We start by simplifying the weakest precondition for the differential action and a post-condition $q : PRED(x)$:

$$\forall \phi : C^1.\ SF(\phi, 0 \leqslant x < 200, \varepsilon \leqslant \dot{x}) \wedge \Delta(\phi, 0 \leqslant x < 200) > 0$$

$$\Rightarrow \Delta(\phi, 0 \leqslant x < 200) < \infty \wedge q[\phi(\Delta(\phi, 0 \leqslant x < 200))/x]$$

Here, the solution function condition, $SF(\cdot)$, is

$$\phi(0) = x \wedge \forall \tau : \mathbb{R} \cap [0, \infty).\ 0 \leqslant \phi(\tau) < 200 \Rightarrow \varepsilon \leqslant \dot{\phi}(\tau)$$

It states that a solution function must be a monotonically increasing function of class C^1 while the evolution guard holds. Therefore, $\Delta(\phi, 0 \leqslant x < 200) > 0$ holds only from initial states $0 \leqslant x < 200$. And, from such a state, the duration is finite and the expression $\phi(\Delta(\phi, 0 \leqslant x < 200))$ simplifies to 200. Therefore, the weakest precondition for the differential action, above, simplifies all in all to: $0 \leqslant x < 200 \Rightarrow q[200/x]$. However, this is precisely the semantics of $0 \leqslant x < 200 \rightarrow x := 200$, whereby the refinement of the first pair of actions holds trivially. The refinement of the other two pairs of actions are shown similarly.

The proof obligations (iii) and (iv) hold also trivially, because there are no new intermediate actions in $\mathscr{C}Train$.

Lastly, also the proof obligation (v) holds, because the actions in $\mathscr{A}Train$ and $\mathscr{C}Train$ are equally enabled.

Thus, we have shown that $\mathscr{C}Train$ is a weak simulation of $\mathscr{A}Train$, as it continualizes the conventional actions.

Introducing velocity: In this second and last step we introduce the notion of velocity by adding acceleration and deceleration to the specification. Such a hybrid action system, where v denotes the velocity, is

$$\begin{aligned}
\mathscr{V}Train \mathrel{\hat{=}} |[&\mathbf{var}\, x^*, v : \mathbb{R}, \mathbb{R} \bullet x, v{:=}0, 0;\\
&\mathbf{with}\ 0 \leqslant x < 200 \quad :\rightarrow \dot{x} = v \wedge \dot{v} = 1\\
&\ []\ \ 200 \leqslant x < 2360 :\rightarrow \dot{x} = v\\
&\ []\ \ 2360 \leqslant x < 2560 :\rightarrow \dot{x} = v \wedge \dot{v} = -1\\
&]|
\end{aligned}$$

Note that $\mathscr{V}\mathit{Train}$ reduces non-determinism of evolutions by allowing only some specific evolutions of $\mathscr{C}\mathit{Train}$. In particular, $\mathscr{V}\mathit{Train}$ preserves the timing properties of those specific evolutions, but not of the excluded evolutions.

The refinement relation used for proving the correctness must describe what we mean by acceleration and deceleration:

$$\begin{aligned} r \mathrel{\hat{=}} & (0 \leqslant x \leqslant 200 \Rightarrow v = \sqrt{2x}) \wedge \\ & (200 \leqslant x \leqslant 2360 \Rightarrow v = 20) \wedge \\ & (2360 \leqslant x \leqslant 2560 \Rightarrow v = \sqrt{5120 - 2x}). \end{aligned}$$

Here, the conjuncts describe, in order of appearance, how the velocity behaves during acceleration, travelling with a constant velocity, and deceleration. Since the hybrid alternations of the models have only a **with** part, they unfold to ordinary iterations and we may use Definition 27 to prove

$$\mathscr{C}\mathit{Train} \preceq_r \mathscr{V}\mathit{Train}$$

Proof obligation (i) is easily shown, because $r[0/x, 0/v]$ holds trivially.

To show proof obligation (ii) we use Theorem 28, and show that it holds for each pair of actions. For brevity, we show here only the proof for the first action pair:

$$0 \leqslant x < 200 :\rightarrow \varepsilon \leqslant \dot{x} \sqsubseteq_r 0 \leqslant x < 200 :\rightarrow \dot{x} = v \wedge \dot{v} = 1$$

We have already simplified the weakest precondition for the left-hand side differential action, which is $0 \leqslant x < 200 \Rightarrow q[200/x]$. Therefore, we simplify now the weakest precondition for the right-hand side differential action assuming that the refinement relation holds:

$$\begin{aligned} & \forall \phi_x : C^1, \phi_v : C^1. \\ & r \wedge \mathit{SF}((\phi_x, \phi_v),\ 0 \leqslant x < 200,\ \dot{x} = v \wedge \dot{v} = 1) \\ & \wedge \Delta((\phi_x, \phi_v),\ 0 \leqslant x < 200) > 0 \\ & \Rightarrow \Delta((\phi_x, \phi_v),\ 0 \leqslant x < 200) < \infty \\ & \wedge\ (r \wedge q)\ [\phi_x(\Delta((\phi_x, \phi_v),\ 0 \leqslant x < 200))/x, \\ & \qquad \phi_v(\Delta((\phi_x, \phi_v),\ 0 \leqslant x < 200))/v] \end{aligned}$$

Here, the general solution to the system of differential equations, $\dot{x} = v \wedge \dot{v} = 1$, is formed by the functions $\phi_x : (\lambda x_0 : \mathbb{R}, v_0 : \mathbb{R}, \tau : \mathbb{R}.\ x_0 + v_0 \cdot \tau + \frac{1}{2}\tau^2)$ and $\phi_v : (\lambda v_0 : \mathbb{R}, \tau : \mathbb{R}.\ v_0 + \tau)$. Therefore, the weakest precondition rewrites:

$$\begin{aligned} & \forall x_0 : \mathbb{R},\ v_0 : \mathbb{R}. \\ & r \wedge \mathit{SF}((\phi_x.x_0.v_0,\ \phi_v.v_0),\ 0 \leqslant x < 200,\ \dot{x} = v \wedge \dot{v} = 1) \\ & \wedge \Delta((\phi_x.x_0.v_0,\ \phi_v.v_0),\ 0 \leqslant x < 200) > 0 \\ & \Rightarrow \Delta((\phi_x.x_0.v_0,\ \phi_v.v_0),\ 0 \leqslant x < 200) < \infty \end{aligned}$$

$$\wedge\ (r \wedge q)\ [\phi_x.x_0.v_0.\Delta((\phi_x.x_0.v_0,\ \phi_v.v_0),\ 0 \leqslant x < 200)/x,$$
$$\phi_v.v_0.\ \Delta((\phi_x.x_0.v_0,\ \phi_v.v_0),\ 0 \leqslant x < 200)/v]$$

Here, the solution function condition, $SF(\cdot)$, simplifies to $x = x_0 \wedge v = v_0$, and we have

$$r \wedge \Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200) > 0$$
$$\Rightarrow \Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200) < \infty$$

$$\wedge\ (r \wedge q)\ [\phi_x.x.v.\ \Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200)/x,$$
$$\phi_v.v.\Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200)/v]$$

Now, $r \wedge \Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200) > 0$ holds only from initial states $0 \leqslant x < 200$ and, hence, we work with:

$$r \wedge 0 \leqslant x < 200 \Rightarrow \Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200) < \infty$$
$$\wedge\ (r \wedge q)[\phi_x.x.v.\Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200)/x,$$
$$\phi_v.v.\Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200)/v]$$

Next, the duration expression $\Delta((\phi_x.x.v,\ \phi_v.v),\ 0 \leqslant x < 200)$ unfolds to the expression $\inf\{\tau | \tau \geqslant 0 \wedge \neg(0 \leqslant x + v\tau + \frac{1}{2}\tau^2 < 200)\}$. By the refinement relation we know that $r \wedge 0 \leqslant x < 200 \Rightarrow v = \sqrt{2x}$. Therefore, from the initial states $0 \leqslant x < 200$ the infimum is the value of τ that satisfies $200 - x - v\tau - \frac{1}{2}\tau^2$, i.e., $\sqrt{v^2 + 400 - 2x} - v$. Thus, the duration of the evolution is $\sqrt{2x + 400 - 2x} - \sqrt{2x}$, i.e., $20 - \sqrt{2x}$. And, the weakest precondition expression simplifies to

$$r \wedge 0 \leqslant x < 200 \Rightarrow (r \wedge q)[\phi_x.x.v.(20 - \sqrt{2x})/x, \phi_v.v.(20 - v)/v]$$

And, that is $r \wedge 0 \leqslant x < 200 \Rightarrow (r \wedge q)[200/x,\ 20/v]$ with this expression the refinement condition of the two differential actions is written as

$$\forall q : PRED(x).$$
$$r \wedge (0 \leqslant x < 200 \Rightarrow q[200/x]) \Rightarrow (0 \leqslant x < 200 \Rightarrow (r \wedge q)[200/x,\ 20/v])$$

Because, here, q is a predicate over only the variable x and the textual substitution on the right trivially satisfies r, the expression simplifies further to

$$\forall q : PRED(x).$$
$$r \wedge (0 \leqslant x < 200 \Rightarrow q[200/x]) \Rightarrow (0 \leqslant x < 200 \Rightarrow q[200/x])$$

This expression holds, whereby the refinement of the first pair of actions holds. The refinement of the other two pairs of actions are shown similarly.

The proof obligations (iii) and (iv) hold also trivially, because there are no new intermediate actions in $\mathscr{V}Train$.

Lastly, also the proof obligation (v) holds, because the actions in $\mathscr{C}Train$ and $\mathscr{V}Train$ are equally enabled.

Thus, we have shown that $\mathscr{V}Train$ is a weak simulation of $\mathscr{C}Train$, as it continuously transforms the abstract differential actions.

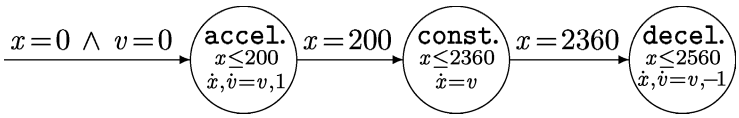

Fig. 5. A hybrid automaton of the travelling train.

7. Related work

In this section we place hybrid action systems in context and in contrast to several other hybrid system formalisms.

7.1. Hybrid automata

In the *hybrid automata* formalism [17], finite automata with continuous variables are used to model hybrid systems. The states of the automaton are called *locations*. At each location some dynamical laws describe how the values of the variables are continuously evolved. Thus, time advances at locations. Consequently, all the *transitions* between locations are considered to be instantaneous. Each transition is guarded and it may assign new values to the variables. There are also many other variants of hybrid automata, for instance, by Nicollin et al. [25,30]. Fig. 5 shows an example of a hybrid automaton that models the travelling train described in Section 6.3.

The hybrid automata formalism resembles the hybrid action systems formalism. For instance, both of these formalisms separate the continuous-time dynamics clearly from the discrete-time dynamics. Also, the meaning of several parallel hybrid automata can be unfolded into a single hybrid automaton by a parallel composition operator [17]. Furthermore, hybrid automata may be abstracted for easier analysis through bisimulation [16].

Recently, there has been some work by Rönkkö and Li [26] on relating these two formalisms. They have shown, for instance, that a class of hybrid action systems with constant rate differential actions and without non-deterministic assignment, called *linear hybrid action systems*, is a strict subclass of linear hybrid automata. Furthermore, they have given and proven correct a translation algorithm that translates a linear hybrid action system to a linear hybrid automata. Consequently, the tools for linear hybrid automata can also be used for the analysis of linear hybrid action systems.

7.2. Phase transition systems

Kesten et al. [20] have taken a more analytical approach to hybrid systems. They separate the concerns of how to model computations, specify requirements, and describe a system. Clearly, this allows the selection of the most appropriate model, logic, and description language for hybrid systems. Consequently, they use *hybrid statecharts* to describe the dynamics of the system, *temporal logic* for capturing the requirements, and *phase transition systems* as the computation model. In contrast, we use hybrid action systems both to describe a system, and as a computation model. For requirement specification we use predicate logic in our approach.

$$
\begin{array}{lcl}
V & = & D:\{\pi\}, I:\{x,v,T\} \\
\Theta & : & \pi = accel \wedge x = 0 \wedge v = 0 \wedge T = 0 \\
\mathscr{T} & : & accel\_const : \pi = accel \wedge x = 200 \wedge \pi' = const \wedge T = T' \\
 & & const\_decel : \pi = const \wedge x = 2360 \wedge \pi' = decel \wedge T = T' \\
\mathscr{A} & : & ac : \pi = accel \to v = v_0 + t \wedge x = x_0 + v_0 t + \tfrac{1}{2}t^2 \wedge T = t_0 + t \\
 & & co : \pi = const \to v = v_0 \wedge x = x_0 + v_0 t \wedge T = t_0 + t \\
 & & de : \pi = decel \to v = v_0 - t \wedge x = x_0 + v_0 t - \tfrac{1}{2}t^2 \wedge T = t_0 + t \\
\Pi & : & (\pi = accel \to 0 \leqslant x \leqslant 200) \\
 & & \wedge\ (\pi = const \to 200 \leqslant x \leqslant 2360) \\
 & & \wedge\ (\pi = decel \to 2360 \leqslant x \leqslant 2560)
\end{array}
$$

Fig. 6. A phase transition system of the travelling train.

A phase transition system [20] is a tuple $(V, \Theta, \mathscr{T}, \mathscr{A}, \Pi)$ consisting of system variables V, initial condition Θ, transitions $\mathscr{T}$, set of activities $\mathscr{A}$, and time-progress condition Π. The system variables are explicitly separated into discrete variables D, and continuous variables I, i.e., *integrators*. Among the integrators, there is a *master clock*, T, that can be used as a reference to see how long the system has been running. Initially the master clock is zero. The use of such an explicit time variable is a well-known concept also in real-time systems [1], and has been discussed in detail by Lamport [22] in a hybrid setting, too. The transitions describe the discrete zero-time changes in the system, whereas the activities describe the dynamical laws governing the continuous-time dynamics. The time-progress condition is used for stating when time advances in the model. Phase transition systems can also be seen as a basis for a compositional extension, *hybrid I/O automata*, that was introduced by Lynch, Segala, Vaandrager, and Weinberg [23]. Fig. 6 shows an example of a phase transition system modelling the travelling train described in Section 6.3.

7.3. CSP for hybrid systems

He Jifeng [15] has studied how to extend the *communicating sequential process* formalism, CSP [18], to accommodate the needs of hybrid systems. He gives a predicate semantics much like the weakest liberal precondition semantics for actions shown in this paper. However, the language of hybrid CSP is vastly richer, and more importantly, the conceptual approach of hybrid CSP is quite different from the one of hybrid action systems.

In hybrid CSP, the system consists of processes that communicate via *synchronized channels*. Consequently, parallel composition adheres to the notion of synchronous parallelism. Needless to say, such a parallel composition cannot accommodate concurrency of evolutions. However, it gives much more elegant tools for inspecting the resulting dynamics of synchronous parallelism than parallel composition of hybrid action systems does.

In hybrid CSP time is abstract, or implicit, like in hybrid action systems. However, the notion of atomicity is quite different in these two formalisms. In hybrid CSP

$$
\begin{aligned}
&\mathit{Accel} \;\hat{=}\; (\dot{x}, \dot{v}v = v, 1)_{(0,0)}\\
&\mathit{Const} \;\hat{=}\; (\dot{x} = v)\\
&\mathit{Decel} \;\hat{=}\; (\dot{x}, \dot{v} = v, -1)\\
&\mathit{Train} \;\hat{=}\; \mathit{Accel}[x = 200]\mathit{Const}[x = 2360]\ \mathit{Decel}\ [x = 2560]\ \texttt{STOP}
\end{aligned}
$$

Fig. 7. A hybrid CSP model of the travelling train.

everything can be interrupted, even the assignment statements which may, unlike the assignment action, span over time. Thus, in hybrid CSP, there is no "hard atomicity" as there is in (hybrid) action systems. This also justifies the semantics given for (algebraic) differential equations that has no bounds for the evolutions. Therefore, unlike the differential action, the differential equations in hybrid CSP are always enabled, and they never "terminate" unless they are interrupted. Fig. 7 shows an example of a hybrid CSP specification modelling the travelling train. In the specification $P[p]Q$ denotes the execution of process P until the predicate p evaluates to true when the execution is transferred from P to process Q. Furthermore, `STOP` is the standard *silent process*.

Reasoning about properties in hybrid CSP requires unfolding of the predicate semantics of the primitive operators for the given processes. After that, the obtained predicate is shown to cover the properties of interest which are also given as predicates. In this respect, the approach of hybrid CSP is similar to the one of hybrid action systems. An alternative *duration calculus* [32] semantics for hybrid CSP is given by Zhou et al. [31].

8. Conclusion

In this paper, we presented a conjunctive and monotonic generalization of a differential action [28,29]. The novelty of the generalized differential action is the use of arbitrary predicates over variables and their derivatives in modelling evolutions; thus, capturing non-deterministic evolutions. Moreover, we showed that such a differential action has an intuitively appealing predicate transformer semantics where the enabledness is determined by the evolution guard and the existence of solutions. This also subsumes the dangling problem of arithmetical undefinedness. For instance, a differential equation $\dot{x} = \frac{1}{x}$ is not defined in a state $x = 0$, and has no solution at that state. Therefore, the corresponding differential action is not enabled in the state $x = 0$, either. Hence, the differential action has a role of a guarded command due to the way its enabledness is treated. Consequently, a differential action blends smoothly with the conventional actions in action systems.

We also investigated the concurrency of evolutions, i.e., differential actions operating at the same time on the same variables. We formalized this as the linear composition operator. It is used in the parallel composition of hybrid action systems, where discrete changes are interleaved and continuous evolutions are composed linearly. The support for concurrency of evolutions helps in developing independent hybrid component specifications. As we showed with a reactor tank example, without concurrent evolutions,

one may have to encode in one component the continuous-time dynamics of all the other components. This raises the coupling between the specifications of components, which, in turn, lowers the clarity and reusability.

However, when specifying a system where the complexity is caused by many detailed and intricate sequential actions, parallel composition does not offer any assistance. In such a case we would like to use a stepwise development method. Then, details are added to an abstract specification in a provably correct manner. Motivated by this, we investigated how the weak simulation proof obligations of action systems generalize to hybrid action systems. We formalized weak simulation using data refinement, and showed how data refinement condition for actions applies also to the differential action. Due to the implicit notion of time, the data refinement condition also captures continualization and continuous transformation of differential actions. Consequently, we showed indirectly that the differential action is invariant under transformation of time domain, which is clearly a property that an action speaking of evolutions over implicit time must adhere. We then applied these results in an example where we illustrated the use of weak simulation in developing a hybrid specification for a travelling train. It should be noted that weak simulation proof obligations are also applicable in the abstraction of a specification. This could be useful when analyzing the stability of a given hybrid system, which is an interesting topic for future research.

The hybrid action systems formalism presented in this paper is complementary to other existing hybrid formalisms. For instance, the parallel composition of hybrid action systems accommodates concurrent evolutions that is typically not taken into consideration in other formalisms. On the other hand, other formalisms, like hybrid automata and hybrid CSP, give more elegant tools for inspecting synchronously coupled hybrid processes. Still, the most important feature of hybrid action systems complementary to other formalisms is the support for stepwise development of hybrid specifications. Use of refinement and abstraction provide the means for developing and analyzing non-trivial hybrid systems. Applications in specific settings with advanced control engineering concepts, like hierarchical control, is an important topic for future research.

Acknowledgements

We sincerely thank the anonymous referees for their instructive and careful comments.

References

[1] M. Abadi, L. Lamport, An old-fashioned recipe for real-time, ACM Trans. Programming Languages Systems 16(5) (1994) 1543–1571, also appeared in: J.W. de Bakker, C. Huizing, W.-P. de Roever, G. Rozenberg (Eds.), Lecture Notes in Computer Science, Vol. 600, Springer, Berlin, 1992.

[2] R.J.R. Back, Refinement calculus, part II: parallel and reactive programs, in: J.W. de Bakker, W.P. de Roever, G. Rozenberg (Eds.), Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. Proceedings, Lecture Notes in Computer Science, Vol. 430, Springer, Berlin, 1990, pp. 67–93.

[3] R.J.R. Back, Atomicity refinement in a refinement calculus framework, Reports on Computer Science and Mathematics, 141, Åbo Akademi, 1993.
[4] R.J.R. Back, R. Kurki-Suonio, Decentralization of process nets with centralized control, in: Proc. 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, 1983, pp. 131–142.
[5] R.J.R. Back, K. Sere, Stepwise refinement of action systems, Struct. Programming 12 (1991) 17–30.
[6] R.J.R. Back, J. von Wright, Trace refinement of action systems, in: J. Parrow (Ed.), CONCUR94, Lecture Notes in Computer Science, Vol. 666, Springer, Berlin, 1994.
[7] R.J.R. Back, J. von Wright, Refinement calculus: a systematic introduction, Graduate Texts in Computer Science, Springer, Berlin, 1998.
[8] M. Bonsangue, J.N. Kok, Semantics, orderings and recursion in the weakest precondition calculus, in: J.W. de Bakker, G. Rozenberg, W.P. de Roever (Eds.), Proceedings Rex Workshop on Semantics: Foundations and Applications, Lecture Notes in Computer Science, Vol. 666, Springer, Berlin, 1993, pp. 91–110.
[9] H. Bowman, Modelling timeout without timelocks, in: J.-P. Katoen (Ed.), Formal Methods for Real-Time and Probabilistic Systems, Lecture Notes in Computer Science, Vol. 1601, Springer, Berlin, 1999.
[10] M.S. Branicky, Studies in hybrid systems: modeling, analysis, and control, Ph.D. Thesis, EECS Department, Massachusetts Institute of Technology, 1995.
[11] M.S. Branicky, Analyzing and synthesizing hybrid control systems, in: G. Rozenberg, F. Vaandrager (Eds.), Lectures on Embedded Systems, Lecture Notes in Computer Science, Vol. 1494, Springer, Berlin, 1998, pp. 74–113.
[12] M. Broy, G. Nelson, Adding fair choice to Dijkstra's calculus, CM Transactions on Programming Languages Systems 16 (3) (1994) 924–938.
[13] M. Butler, E. Sekerinski, 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, Lecture Notes in Computer Science, Vol. 1165, Springer, Berlin, 1996.
[14] E.W. Dijkstra, A Discipline of Programming, Prentice-Hall International, Englewood Cliffs, NJ, 1976.
[15] J. He, From CSP to hybrid systems, in: A.W. Roscoe (Ed.), A Classical Mind, Essays in Honour of C.A.R. Hoare, Prentice-Hall, Englewood Cliffs, NJ, 1994, pp. 171–189.
[16] T.A. Henzinger, Hybrid automata with finite bisumaltions, in: Proc. 22nd Internat. Colloq. on Automata, Languages, and Programming (ICALP 1995), Lecture Notes in Computer Science, Vol. 994, Springer-Verlag, 1995, pp. 324–335.
[17] T.A. Henzinger, The theory of hybrid automata, in: Proc. 11th Ann. IEEE Symp. on Logic in Computer Science (LICS 1996), 1996, pp. 278–292.
[18] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall International, London, 1984.
[19] M. Jaffe, N. Leveson, M. Heimdahl, B. Melhart, Software requirements analysis for real-time process-control systems, IEEE Trans. Software Eng. 17 (3) (1991) 241–258.
[20] Y. Kesten, Z. Manna, A. Pnueli, Verification of clocked and hybrid systems, in: G. Rozenberg, F. Vaandrager (Eds.), Lectures on Embedded Systems, Lecture Notes in Computer Science, Vol. 1494, Springer, Berlin, 1998, pp. 4–73.
[21] J. La Salle, S. Lefschetz, Stability by Liapunov's Direct Method with Applications, Academic Press, New York, 1961.
[22] L. Lamport, Hybrid systems in TLA$^+$, in: R. Grossman, A. Nerode, A.P. Ravn, H. Rischel (Eds.), Hybrid Systems, Lecture Notes in Computer Science, Vol. 736, Springer, Berlin, 1993, pp. 149–178.
[23] N.A. Lynch, R. Segala, F.W. Vaandrager, H.B. Weinberg, Hybrid I/O automata, in: R. Alur, T.A. Henzinger, E.D. Sontag (Eds.), Hybrid Systems III, Lecture Notes in Computer Science, Vol. 1066, Springer, Berlin, 1996, pp. 496–510.
[24] M.H.A. Newman, Elements of the Topology of Plane Sets of Points, 1st Edition, 1939, Cambridge University Press, Cambridge, 1961.
[25] X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, An approach to the description and analysis of hybrid systems, in: R. Grossman, A. Nerode, A.P. Ravn, H. Rischel (Eds.), Hybrid Systems, Lecture Notes in Computer Science, Vol. 736, Springer, Berlin, 1993, pp. 149–178.
[26] M. Rönkkö, X. Li, Linear hybrid action systems, Nordic J. Comput. 8 (1) (2001) 159–177.

[27] M. Rönkkö, A.P. Ravn, Switches and jumps in hybrid action systems, Proc. Estonian Acad. Sci. Eng. 4 (2) (1998) 106–118.
[28] M. Rönkkö, A.P. Ravn, Action systems with continuous behavior, in: P.J. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, S. Sastry (Eds.), Proceedings of Hybrid Systems V, Lecture Notes in Computer Science, Vol. 1567, Springer, Berlin, 1999, pp. 304–323.
[29] M. Rönkkö, K. Sere, Refinement and continuous behavior, in: F.W. Vaandrager, J.H. van Schuppen (Eds.), Proceedings of Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, Vol. 1569, Springer, Berlin, 1999, pp. 223–237.
[30] S. Yovine, Model-checking timed automata, in: G. Rozenberg, F. Vaandrager (Eds.), Embedded Systems, Lecture Notes in Computer Science, Vol. 1494, Springer, Berlin, 1998.
[31] C. Zhou, Wang Ji, A.P. Ravn, A formal description of hybrid systems, in: R. Alur, T. Henzinger, E. Sontag (Eds.), Hybrid Systems III, Lecture Notes in Computer Science, Vol. 1066, Springer, Berlin, 1996, pp. 511–530.
[32] C. Zhou, A.P. Ravn, M.R. Hansen, An extended duration calculus for hybrid systems, in: R. Grossman, A. Nerode, A.P. Ravn, H. Rischel (Eds.), Hybrid Systems, Lecture Notes in Computer Science, Vol. 736, Springer, Berlin, 1993.
[33] D.G. Zill, M.R. Cullen, Differential Equations with Boundary-value problems, 4th Edition, Brooks/Cole Publishing Company, Pacific Grove, 1997.