Action Systems with Continuous Behaviour

Mauno Rönkkö

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

Anders P. Ravn

Department of Information Technology,
Technical University of Denmark,
Bldg. 344, DK-2800 Lyngby, Denmark,
`apr@it.dtu.dk`

Turku Centre for Computer Science
TUCS Technical Report No 151
December 1997

ISBN 952-12-0125-8
ISSN 1239-1891

Abstract

An action system framework is a predicate transformer based method for modelling and analysing distributed and reactive systems. The actions are statements in Dijkstra's guarded command language, and their semantics is given by predicate transformers. We extend conventional action systems with a *differential action* consisting of a differential equation and an evolution guard. The semantics is given by a weakest liberal precondition transformer, because it is not always desirable that differential actions terminate. It is shown that the proposed differential action has a semantics which corresponds to a discrete approximation when the discrete step size goes to zero.

The extension gives action systems the power to model real-time clocks and continuous evolutions within hybrid systems. In this paper we give a standard form for such a hybrid action system. We also extend parallel composition to hybrid action systems. This does not change the original meaning of the parallel composition, and therefore ordinary action systems compose in parallel with hybrid action systems.

Note This TUCS technical report combines and updates the two TUCS technical reports no.109 and no.110.

Keywords: weakest liberal precondition, actions, hybrid action systems, hybrid systems, parallel composition, HyTech

TUCS Research Group
Programming Methodology Research Group

1 Introduction

Action systems, originally proposed by Back and Kurki-Suonio [3], are predicate transformer based systems for modelling discrete computations. For a given program, the predicate has the program variables as free variables. The transformer uses Dijkstra's idea of describing the effect of a given program by calculating the precondition which guarantees that a given postcondition holds. Action systems have been extensively used in sequential program refinement, and also in reactive and concurrent programs, see e.g. [4, 9].

Previously action systems have been extended with explicit clock variables when reasoning about real-time systems, and with discrete approximations when modelling continuous phenomena in hybrid settings [13, 19]. However, since there is no technical reason why predicates over program variables should not speak of continuous entities, it seems possible to simplify reasoning in these applications by introducing the differential equations governing evolution of clocks and other continuous variables directly into the framework. Such an extension is the main result of this paper. A guideline in developing the differential action is a wish to preserve the connection between discrete accumulation and continuous evolution, such that the weakest liberal precondition semantics corresponds to the one found for the approximating discrete accumulation.

Inspired by the Branicky's unified model for hybrid systems [6], we identify a class of action systems suitable for modelling hybrid systems. These hybrid action systems alternate between discrete changes and continuous evolutions.

The parallel composition for ordinary action systems is described by interleaving of actions from the constituent systems. This model is based on the assumption that the effect of two simultaneous actions is the same as the effect of the two actions in some sequence. This assumption is not valid in a continuous parallel world, where the total effect of simultaneous differential actions is the sum of their individual effects. This form of composition is captured by a linear composition. We extend the parallel composition for hybrid action systems so, that it uses linear composition for combining the continuous evolutions, but interleaves discrete actions. We retain the usual meaning of the parallel composition for ordinary action systems. Hence, the mixed parallel composition between action systems and hybrid action systems remain also meaningful.

Related work.

Action systems are similar to the UNITY programs [10]. Especially, the refinement methodology is central in both of these approaches. In [11] is a discussion of how a program logic may be used to reason about continuous systems. That paper focuses on exploring the potentials of refinement reasoning for such systems, whereas we concentrate on a precise definition of the semantics.

Action systems have an operational flavour, and our extension is thus related to various hybrid automata models [1, 20, 18] and the general hybrid model surveyed in [7]. However, our interest is not in the basic model, but in an integration of specifications (predicates) and actions such that one can reason without reference to the semantics. A similar concept is found in the description language used in HyTech [15, 16], where the approximations formulated in [17] would be seen as action refinements. In applications, it seems promising to first use action refinement and standard mathematical analysis tools to weaken complex predicates with differential equations and then translate to HyTech and use its potential for automated checking.

A directly related piece of work is the duration calculus semantics for hybrid CSP [23, 14], which, however, does not include a proof theory. The current paper may be seen as an investigation of a proof theory analogous to the one investigated for duration calculus in [22]. However, that work does not include parallel composition.

Overview.

We start by defining action systems in Section 2. In Section 3 we derive from the weakest liberal precondition for a discrete accumulation a predicate describing continuous evolution. Based on this predicate we define the differential action. In Section 4 we study a gate example [21] to illustrate the similarities and the differences between formulations using discrete accumulation and continuous evolution. In Section 5 we identify a suitable form to describe hybrid systems. We call such action systems hybrid action systems. In Section 6 we investigate the parallel composition of action systems and extend it so that it is meaningful also when composing hybrid action systems. In Section 7 we use a water tank example to illustrate the extended parallel composition. We also use the example to outline, how HyTech can be used in analysing hybrid action systems. Section 8 concludes and discusses directions for further work.

2 Actions

An action is any statement in Dijkstra's guarded command language [12], including pure guarded commands. In a system, the actions operate on a fixed set of variables. The values of these variables identify a state in the system. Therefore, predicates or conditions on these variables provide a convenient way of specifying and reasoning about states in a system.

When we consider a given action A, we may speak of a *postcondition* q. It is a predicate specifying desirable states after the execution of the action.

By using the postcondition as a reference, we can define the meaning or the semantics for the action A. We can, for example, use the *weakest precondition* predicate transformer $\mathrm{wp}\,(A,\ q)$ for this purpose. It returns a predicate describing the largest set of states from which the action terminates and reaches the postcondition.

The weakest precondition can also be used for calculating two basic properties for any action. These properties are the termination set and the enabledness set. The set of states from which an action terminates, i.e. reaches an arbitrary state, is given by $\mathrm{t}A \ = \ \mathrm{wp}\,(A,\ true)$. Similarly, the set of states in which an action is enabled, i.e. the states from which an action starts to execute, is given by $\mathrm{g}A \ = \ \neg\mathrm{wp}\,(A,\ false)$.

In the following we give the semantics for the actions with the *weakest liberal precondition*, wlp. Unlike the weakest precondition, it does not require termination. The relation between these two predicate transformers is given by Dijkstra [12] as $\mathrm{wp}\,(A,\ q) \ = \ \mathrm{wlp}\,(A,\ q) \ \wedge \ \mathrm{t}A$ or equivalently when A is *conjunctive* $\mathrm{wlp}\,(A,\ q) \ = \ (\mathrm{t}\ A \ \Rightarrow \ \mathrm{wp}\,(A,\ q))$.

Furthermore, we consider two actions A and B to be the same when their semantics agree for any postcondition in the sense of weakest liberal precondition:

$$A \equiv B \quad \textbf{iff} \quad \forall q \ \cdot \ \mathrm{wlp}\,(A,\ q) = \mathrm{wlp}\,(B,\ q)$$

Any two actions for which this relation holds can be substituted for each other in any context without affecting the meaning.

2.1 Semantics for actions

In this paper we use capital letters such as X, Y, E, and F to denote a vector or list of variables and other components. For clarity, we do not use subscripts when referring to the components in a vector; we explicitly say which small letters denote vector components, whenever this is needed. Furthermore, let $q[E/X]$ denote textual substitution of expressions E for the free variables X in a predicate q.

Elementary actions are defined as in [5, 8].

$$\begin{array}{lcl}
\mathrm{wlp}\,(\textbf{abort},\ q) & \widehat{=} & true \\
\mathrm{wlp}\,(\textbf{skip},\ q) & \widehat{=} & q \\
\mathrm{wlp}\,(X := E,\ q) & \widehat{=} & q[E/X]
\end{array}$$

Note that *abort* never terminates, whereas a **skip** or an assignment are sure to terminate.

Let, as before, A and B be actions, and p be a predicate. The sequential composition, the composition by a non-deterministic choice, as well as prefixing an action with a guard are defined by

$$\begin{array}{lcl}
\mathrm{wlp}\,(A; B,\ q) & \widehat{=} & \mathrm{wlp}\,(A,\ \mathrm{wlp}\,(B,\ q)) \\
\mathrm{wlp}\,(A \ [\!]\ B,\ q) & \widehat{=} & \mathrm{wlp}\,(A,\ q) \ \wedge \ \mathrm{wlp}\,(B,\ q) \\
\mathrm{wlp}\,(p \rightarrow A,\ q) & \widehat{=} & p \ \Rightarrow \ \mathrm{wlp}\,(A,\ q)
\end{array}$$

An assertion ensures that a given condition p holds in the current state. If the given condition does not hold, the assertion aborts. Hence, the meaning for the assertion is $\{p\} \equiv p \rightarrow \textbf{skip} \ [\!]\ \neg p \rightarrow \textbf{abort}$ From which we calculate the semantics:

$$\begin{aligned} \mathrm{wlp}(\{p\},\ q) &= \mathrm{wlp}(p \rightarrow \textbf{skip},\ q) \ \wedge\ \mathrm{wlp}(\neg p \rightarrow \textbf{abort},\ q) \\ &= (p \ \Rightarrow\ \mathrm{wlp}(\textbf{skip},\ q)) \ \wedge\ \mathit{true} \end{aligned}$$

resulting in the semantics $\mathrm{wlp}(\{p\},\ q) \ = \ (p \ \Rightarrow\ q)$.

Iteration has the liberal semantics

$$\mathrm{wlp}(\ \textbf{do}\ \ A\ \ \textbf{od},\ q) \quad \hat{=} \quad \forall n \geq 0 \ \cdot\ \mathrm{wlp}(A^n,\ \mathrm{g}A \ \vee\ q)$$

where $A^0 \mathrel{\hat{=}} \textbf{skip}$ and $A^{n+1} \mathrel{\hat{=}} A; A^n$. The semantics for iteration is useful when proving some general properties related to the iteration of actions. For example, we can easily show that if p is an invariant for the iterated action, $p \ \Rightarrow\ \mathrm{wlp}(A,\ p)$, it is also an invariant throughout the entire iteration, $p \ \Rightarrow\ \mathrm{wlp}(\ \textbf{do}\ \ A\ \ \textbf{od},\ p)$. An iteration terminates when it is no longer enabled, $\mathrm{t}(\ \textbf{do}\ \ A\ \ \textbf{od}) = \exists\, n \geq 0 \ \cdot\ \neg\, \mathrm{g}A^n$.

2.2 Action systems

An action system $\mathcal{A}$ is an initialised block of the form

$$\begin{aligned} \mathcal{A} \ = \ &|[\ \textbf{var}\ \ X : T \\ &\quad X := E; \\ &\quad \textbf{do}\ \ A_1\ [\!]\ \ \ldots\ [\!]\ \ A_n\ \ \textbf{od} \\ &]| : Z \end{aligned}$$

The expression $\textbf{var}\ \ X : T$, declares the list of variables X with types T. Any of these variables can be *global*. This is denoted by a star (e.g. x^*) in the declaration. A global variable can be shared with other action systems. The global variables that do not belong to X, but are used by the actions A_1 ... A_n, are in the imported global variables Z. The variables in X that are not global, are *local* for the block. The local variables do not exist outside the action system where they are defined. All the local and global variables form the *state space* of an action system.

The action $X := E$ initialises the variables X by expressions E. Therefore, the global variables are initialised in the action system, where they are declared. Because the initialisation is a multiple assignment, the expression E may not refer to any variable in X or Z. For example, an initialisation $x, y := x + 3, y + 4$, where $\{x, y\} \ \in\ X$, is illegal, since the values of the variables are undefined before the initialisation. All action systems initialise before any action is taken in one of them. This becomes apparent later, when the parallel composition for the action systems is defined. After initialisation, the actions A_1 ... A_n are repeatedly executed when enabled. The selection of an enabled action for execution is non-deterministic. There are no fairness assumptions about selection of actions. The execution of an elementary action is always atomic. This ensures that during the execution, the values of the variables remain unchanged, unless the action itself changes them. If two enabled actions refer to disjoint variables, their execution can be in any order or in parallel. Hence, this models parallelism by interleaving.

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

3 Differential Action

We extend the conventional set of actions with a *differential action* that describes continuous behaviour. In the following we derive the semantics for the differential action from a discrete accumulation that approximates the continuous behaviour. We also show some of the useful properties related to the use of differential actions.

3.1 Discrete accumulation

In discrete accumulation a variable x is incremented from an initial value until a guard ceases to hold (if ever). When a discrete accumulation is used to approximate a continuous behaviour, the increment $f(x)$ is approximately the differential multiplied by a fixed, positive time step h. Both the increment $f(x)$ and the guard e may depend on h. The discrete accumulation is captured by

$$\mathbf{do}\quad e\ \rightarrow\ x := x + f(x)\cdot h\quad \mathbf{od}$$

The meaning of the discrete accumulation is given by the weakest liberal precondition for the iteration action. Let $p[v/x]^i$ denote the i times repeated substitution $p[v/x]...[v/x]$, and n and m be natural numbers. By unfolding the precondition for the action above we get

$$\begin{aligned}&\mathrm{wlp}\,(\,\mathbf{do}\quad e\ \rightarrow\ x := x + f(x)\cdot h\quad \mathbf{od},\ q) \\ &=\ \forall\, n \geq 0\ \cdot\ (e\ \vee\ q)[x + hf(x)/x]^n\ \vee \\ &\quad\ \exists\, 0 \leq m < n\ \cdot\ \neg e[x + hf(x)/x]^m\end{aligned} \tag{1}$$

Analysis of this predicate is laborious, because it requires the calculation of a predicate after repeated substitutions. Fortunately, we can express the repeated substitution as a single substitution with the help of a recursive function, Φ_h, which for the fixed $h > 0$ is given by:

$$\Phi_h(n\cdot h)\ =\ x + h\cdot \textstyle\sum_{i=h}^{n\cdot h} f(\Phi_h(i-h))$$

This function has the property that $\Phi_h(n\cdot h)$ equals $x[x + hf(x)/x]^n$. The function Φ_h also has an implicit form:

$$\Phi_h(0) = x,\quad \forall\, i \geq 0\ \cdot\ \frac{\Phi_h(i\cdot h + h) - \Phi_h(i\cdot h)}{h} = f(\Phi_h(i\cdot h)) \tag{2}$$

It is this form that we use in the concise semantics for the discrete accumulation. From the precondition expression (1) above we obtain

$$
\begin{aligned}
&\mathrm{wlp}\,(\,\mathbf{do}\quad e \;\rightarrow\; x := x + f(x)\cdot h \quad \mathbf{od},\ q) \qquad\qquad (3)\\
&= \;\exists\,\Phi_h \;\cdot\; \Phi_h(0) = x \;\wedge\; \forall\, i > 0 \;\cdot\; \tfrac{\Phi_h(i\cdot h + h) - \Phi_h(i\cdot h)}{h} = f(\Phi_h(i\cdot h)) \;\wedge\\
&\qquad\quad (\,\forall\, n \geq 0 \;\cdot\; (e \;\vee\; q)[\Phi_h(n\cdot h)/x] \;\vee\\
&\qquad\quad\;\; \exists\, 0 \leq m < n \;\cdot\; \neg e[\Phi_h(m\cdot h)/x]\,)
\end{aligned}
$$

3.2 Evolution

In a continuous evolution, the variable x is assumed to be a differentiable function of time. If a solution exists to the differential equation $\dot{x} = f(x)$, then it is the limit of the step functions $(\Phi_h \mid h > 0)$ as h approaches zero. We denote this limit by Φ, and by applying the definition of differentiation to the formula (2), we have

$$\Phi(0) = x, \quad \dot{\Phi} = f(\Phi)$$

This observation leads us to define a new action for continuously evolving variables, called a *differential action*. The differential action evolves continuously the values of given variables X as long as a guard e holds, this guard condition may use the variables X. The evolution is controlled by a system of differential equations. We use the standard convention of writing a system of differential equations as $\dot{X} = F(X)$. The $\dot{X}$ is a component-wise first derivative of X.

The exploration above is summarised in the following definition.

Definition 1. Let e be a guard and q be a postcondition, both may speak of the variables X. The differential action $e :\ \dot{X} = F(X)$ has the semantics

$$
\begin{aligned}
&\mathrm{wlp}\,(e :\ \dot{X} = F(X),\ q) \;\widehat{=}\\
&\;\exists\,\Phi \;\cdot\; \Phi(0) = X \;\wedge\; \dot{\Phi} = F(\Phi) \;\wedge\\
&\quad (\forall\,\tau \geq 0 \;\cdot\; (e \;\vee\; q)[\Phi(\tau)/X] \;\vee\; \exists\, 0 \leq \delta < \tau \;\cdot\; \neg e[\Phi(\delta)/X])
\end{aligned}
$$

In the semantics, the first two conjuncts require the existence of the solution Φ for the differential equation $\dot{X} = F(X)$. The last term isolates the first point in time, where the guard ceases to hold.

It is difficult to capture the exact link between discrete accumulation and the differential action in the general case; and one cannot expect it to be easy, because it would answer many questions of numerical analysis. However, as an indication of the kind of result one may have, we give the following theorem.

Theorem 2. *If the discrete accumulation function Φ_h, given by*

$$\Phi_h(n \cdot h) \; = \; x + h \cdot \textstyle\sum_{i=h}^{n \cdot h} f(\Phi_h(i-h))$$

converges to a solution Φ for the differential equation $\dot{x} = f(x)$ from above, i.e., $\|\Phi_h\| \leq \|\Phi\|$ for all h, and if the evolution guard e is upward closed, i.e, $\|a\| \leq \|b\|$ implies $e[a/x] \geq e[b/x]$; that is if e is true for some value, then it is true for any larger value.

Then differential termination: $\neg wlp(e : \dot{x} = f(x), false)$ implies accumulation termination: $\neg wlp(\,\mathbf{do}\quad e \to x := x + f(x) \cdot h \;\; \mathbf{od}, false)$.

The implication holds in the reverse direction if 'from below' and 'downward closed' are substituted for 'from above' and 'upward closed'.

3.3 Stability

If the evolution guard e remains true in a differential action, it never terminates, as seen from wlp $(e : \; \dot{X} = F(X), \; false) \; = true$; but the behaviour is bounded by the guard. In this case, the system becomes *stable* with respect to the guard. When something is known about the stability of the differential equation, it may be used in the analysis of the differential action.

Theorem 3. *Assume that the differential equation $\dot{X} = F(X)$ has solutions. Let O be a point in the domain of X. Let a sphere around O with radius ϵ, be denoted by $S(\epsilon)$.*

Assuming that a solution with initial value X_0 in $S(\rho)$ is Ljapunov stable in $S(\epsilon)$, then we have

$$\mathrm{wlp}\,(\;\{X \in S(\rho)\};\; X \in S(\epsilon) : \; \dot{X} = F(X), \; false \;) \;\; = \;\; true$$

Furthermore, if the solution is asymptotically stable, then for any $\gamma > 0$:

$$\mathrm{wlp}\,(\;\{X \in S(\rho)\};\; X \in S(\epsilon) \setminus S(\gamma) : \; \dot{X} = F(X), \; X \in S(\gamma) \;) \;\; = \;\; true$$

Also, if the solution for $\dot{X} = F(X)$ is periodic and O is any point on the solution curve,

$$\mathrm{wlp}\,(\;\{X \neq O\};\; X \neq O : \; \dot{X} = F(X), \; X = O \;) \;\; = \;\; true$$

3.4 Clocks

A clock or timer that measures the passage of time is specified with a differential action. An accurately running timer c is given by $e : \; \dot{c} = 1$. If it had a skew of at most 1 %, it would be $\{0.99 < \delta < 1.01\}$; $e : \; \dot{c} = 1 \cdot \delta$. The assertion in front of the differential action ensures that the skew is within the given limits.

A delay for four time units is given by $c := 1;\quad c < 5 : \ \dot{c} = 1$. The corresponding discrete accumulation would have $f = 1$:

$$\textbf{do}\quad c < 5 \ \rightarrow \ c := c + 1 \cdot h \quad \textbf{od}$$

and have the accumulation function: $\Phi(n \cdot h) \ = \ c + h \cdot \sum_{i=h}^{n \cdot h} 1$ which reduces to $\Phi(n \cdot h) \ = \ c + h \cdot n$; as one would expect. It will terminate when $n = [\frac{5-c}{h}]$, i.e., not later than the corresponding differential action, as predicted by the second part of Theorem 2.

3.5 Composite evolutions

The following is an example of an evolution of several continuous variables. There are two variables x and v that evolve. The x variable is dependent on the v.

$$\begin{array}{ll} & \mathrm{wlp}\,(e :\ \dot{x}, \dot{v} = v, 1\ ,\ \ q) \\ = & \exists\, \Phi_x, \Phi_v \ \cdot\ \Phi_x(0) = x \ \wedge\ \Phi_v(0) = v \ \wedge\ \dot{\Phi}_x = \Phi_v \ \wedge\ \dot{\Phi}_v = 1 \ \wedge \\ & (\ \forall \tau \geq 0 \ \cdot\ (e \ \vee\ q)[\Phi_x(\tau), \Phi_v(\tau)/x, v] \ \vee \\ & \ \ \exists\, 0 \leq \delta < \tau \ \cdot\ \neg e[\Phi_x(\delta), \Phi_v(\delta)/x, v]\ ) \end{array}$$

If we assume that the action has been preceded by an initialisation, say $x, v := 0, 0$, the clock example allows us to simplify the result to

$$\begin{array}{l} \exists\, \Phi_x \ \cdot\ \Phi_x(0) = 0 \ \wedge\ \dot{\Phi}_x = v \ \wedge \\ \qquad (\forall \tau \geq 0 \ \cdot\ (e \ \vee\ q)[\Phi_x(\tau), \tau/x, v] \ \vee\ \exists\, 0 \leq \delta < \tau \ \cdot\ \neg e[\Phi_x(\delta), \delta/x, v]) \end{array}$$

In an application, the timer v is probably there to limit the evolution of the time varying system x. This could result in an guard $e = (v < 5)$, for which we would get:

$$\exists\, \Phi_x \ \cdot\ \Phi_x(0) = 0 \ \wedge\ \dot{\Phi}_x = v \ \wedge\ q[\Phi_x(5), 5/x, v]$$

Notice that $q[\Phi_x(5), 5/x, v]$ specifies all those states, from which x evolves to values $\Phi_x(5)$, when v evolves linearly to value 5.

3.6 Algebraic Properties

In the following we assume that the systems of differential equations used in the differential actions have solutions. Also, the equivalences are in the sense of weakest liberal precondition as defined in Section 2.

A differential action acts as a skip action when the evolution guard does not hold initially. This is captured by the following equivalence, which can eliminate a redundant differential action.

$$\{\neg e\};\ e :\ \dot{X} = F(x) \quad \equiv \quad \{\neg e\};\ \textbf{skip} \tag{4}$$

At termination of a differential action the evolution guard does not hold anymore. This is captured by the following equivalence, which is used for introducing a termination assertion.

$$e: \dot{X} = F(x) \quad \equiv \quad e: \dot{X} = F(x);\ \{\neg e\} \tag{5}$$

With the help of the two properties above, it is easy to prove that sequential composition of a differential action with itself has the same effect as the differential action by itself.

$$\begin{array}{ll} & e: \dot{X} = F(x);\ \ e: \dot{X} = F(x) \\ \equiv & \{\ \textit{termination assertion introduction}\ (5)\ \} \\ & e: \dot{X} = F(x);\ \ \{\neg e\};\ \ e: \dot{X} = F(x) \\ \equiv & \{\ \textit{redundancy elimination}\ (4)\ \} \\ & e: \dot{X} = F(x);\ \ \{\neg e\};\ \mathbf{skip} \\ \equiv & \{\ \mathbf{skip}\ \textit{elimination}\ \} \\ & e: \dot{X} = F(x);\ \ \{\neg e\} \\ \equiv & \{\ \textit{termination assertion removal}\ (5)\ \} \\ & e: \dot{X} = F(x) \end{array}$$

This illustrates that sequential composition takes no time; there is no evolution over the composition.

A differential action does not change the values of variables that do not belong to X. Therefore, we can we can weaken or strengthen the evolution guard by moving parts to or from an assertion. Let g be a predicate not containing variables of X, then the following holds for a differential action

$$\{g\};\ \ g \wedge e: \dot{X} = F(x) \quad \equiv \quad \{g\};\ \ e: \dot{X} = F(x)$$

4 A Gate Example

The following example illustrates the differences and similarities between a discrete and a continuous approach in modelling and analysing a system. We use the lowering of a gate as an example. The gate is lowered with a speed proportional to its current position, that is $\dot{x} = -x$. The task is to model the movement, and measure the time for the gate to become closed. The gate is started 1 meter above the stop and it is considered to be closed, when it is at most 1 cm away from the stop. Figure 1 illustrates the setting.

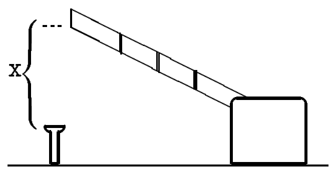

Figure 1: The gate.

4.1 With discrete accumulation

Let, as before, h be the time step. With discrete accumulation the model is

$$x, c := 1, 0; \quad \textbf{do} \quad x > 0.01 \rightarrow x, c := x - x \cdot h, c + 1 \cdot h \quad \textbf{od}$$

The time consumed is given by c, which is $n \cdot h$, where n is the number of iterations; but we cannot use this directly, because n depends on h.

We therefore analyse the series of substitutions taking place in the weakest liberal precondition for the discrete accumulation:

$$\begin{array}{l} x, \quad x - x \cdot h, \quad x - x \cdot h - (x - x \cdot h) \cdot h, \\ (x - x \cdot h - (x - x \cdot h) \cdot h) - (x - x \cdot h - (x - x \cdot h) \cdot h) \cdot h, \quad \ldots \end{array}$$

After taking common factors and performing series simplification, we obtain an accumulation function $\Phi(n \cdot h) = x(1-h)^n$. Next we solve the h from the accumulation guard $1 \cdot (1-h)^n = 0.01$ yielding $h = 1 - e^{\frac{1}{n} \ln 0.01}$. Replacing the h in expression for c and simplifying it gives $c = n(1 - e^{\frac{1}{n} \ln 0.01})$. This expresses the time consumed by the lowering of the gate in proportion to the iteration accuracy n. Taking the limit of the Taylor polynomial of this expression when n approaches infinity gives an approximation for the total time, $-\ln 0.01 \ \approx 4.61$.

4.2 With continuous evolution

The same problem is expressed with continuous evolution by

$$x, c := 1, 0; \quad x > 0.01 : \ \dot{x}, \dot{c} = -x, 1$$

The total time is analysed as follows:

$$\begin{array}{ll} & \mathrm{wlp}\,(\, x, c := 1, 0; \quad x > 0.01 : \ \dot{x}, \dot{c} = -x, 1 \ , \quad x \leq 0.01 \ \wedge \ r = c \,) \\ = & \{ \ \textit{definitions, instantiation of } \Phi_x(t) = xe^{-t} \textit{ and } \Phi_c(t) = c + t \ \} \\ & \forall \tau \geq 0 \ \cdot \ e^{-\tau} > 0.01 \ \vee \ r = \tau \ \vee \ \exists\, 0 \leq \delta < \tau \ \cdot \ e^{-\delta} \leq 0.01 \\ = & \{ \ \textit{case analysis around } \tau = -\ln 0.01 \ \} \\ & r = -\ln 0.01 \end{array}$$

This tells us directly that after $-\ln 0.01$ time units the gate becomes closed.

5 Hybrid Action System

A hybrid system starts from some initial state and alternates between continuous evolution and discrete changes of the variables in the system. The various circumstances, under which this alternation takes place, are studied more closely by Branicky [6].

5.1 Stutter free evolution

A differential action describes how the variables evolve when the guard is enabled. It also states that when the guard does not hold, the differential action has no effect on the variables. This is called stuttering. The equivalence below shows the decomposition of a differential action.

$$e: \dot{X} = F(X) \quad \equiv \quad e \rightarrow e: \dot{X} = F(X) \ [\!] \ \neg e \rightarrow \textbf{skip}$$

When thinking of how a hybrid system behaves, we are mainly interested in the stutter free behaviour. An infinite stuttering is seen as termination. Therefore, we introduce a notational short form for a stutter free differential action, which is $e :\rightarrow \dot{X} = F(X)$ with the meaning of $e \rightarrow e: \dot{X} = F(X)$. The stutter free differential action has the property that it is enabled only when the evolution guard holds, that is $\mathrm{g}(e :\rightarrow \dot{X} = F(X)) \ = \ e$. Also, the enabledness set of a non-deterministic composition of stutter free differential actions is the disjunction of the evolution guards, that is,

$$\mathrm{g}(e_1 :\rightarrow \dot{X} = F_1(X) \ [\!] \ \ldots \ [\!] \ e_n :\rightarrow \dot{X} = F_n(X)) \quad = \quad e_1 \ \vee \ \ldots \ \vee \ e_n$$

5.2 Hybrid alternation

The alternation between discrete and continuous behaviour is expressed by iteration of ordinary and (stutter free) differential actions. Assuming that the cause for the discrete behaviour D is calculated to be $\mathrm{g}D$, and the stutter free evolution is known as DA, the alternation takes a normal form

$$\begin{array}{lllll} \textbf{do} & \mathrm{g}D & \rightarrow & D \\ [\!] & \neg\mathrm{g}D & \rightarrow & DA \\ \textbf{od} \end{array} \tag{6}$$

The D in the model may be a non-deterministic composition of several discrete actions. Similarly, the stutter free evolution DA may be a non-deterministic composition of several stutter free differential actions.

For the form (6) of alternation above we introduce a short form

$$\textbf{alt } D \textbf{ with } DA$$

This construct gives discrete changes priority over the continuous evolution. The continuous evolution may proceed only after all enabled discrete changes have been processed. However, once the continuous evolution is engaged it may continue without any interruption till the selected differential action terminates. After termination, the discrete changes may occur again. Thus, the model inserts discrete computations in between any continuous evolution described by a differential action. The alternation terminates, when it reaches a state in which no discrete changes and no continuous evolution is enabled, that is $\neg\mathrm{g}D \wedge \neg\mathrm{g}DA$. Hence, all the initial states from which the system terminates are given by $\mathrm{t}(\textbf{ alt } D \textbf{ with } DA \ )$.

The time in the model is assumed to progress during the continuous evolution, but not during the discrete changes. Therefore, if the discrete changes do not disable themselves, $\mathrm{wp}\,(D,\ \neg\,\mathrm{g}D)\ =\ \mathit{false}$, there is no continuous evolution nor a termination after the discrete changes become enabled. This corresponds to the system being *aborted*.

With the given form of alternation (6) we can identify a special class of action systems which have the desired hybrid behaviour.

Definition 4. Let X be local variables and Z be imported global variables. The actions in D and DA may refer to all of these variables X or Z. Any action system of the following form is called a hybrid action system.

$$
\begin{array}{lll}
\mathcal{H} & \widehat{=} & |[\ \mathbf{var}\quad X : T \\
 & & \quad X := I; \\
 & & \quad \mathbf{alt}\ D\ \mathbf{with}\ DA \\
 & &]| : Z
\end{array}
$$

When there are no discrete actions, i.e., $\mathrm{g}D = \mathit{false}$, the **alt** part is dropped. Similarly, the **with** part is dropped, when there are no continuous evolutions, i.e., $\mathrm{g}DA = \mathit{false}$.

A hybrid action system may speak also of the continuous evolution of imported global variables. The meaning of a hybrid action system is obtained by unfolding the hybrid alternation into an ordinary iteration (6). This is important for the definition of a non-trivial parallel composition.

6 Parallel Composition

Parallel composition is defined between action systems in [2]. Consider two action systems $\mathcal{A}$ and $\mathcal{B}$

$$
\begin{array}{lll}
\mathcal{A} & \widehat{=} & |[\ \mathbf{var}\quad X : T \\
 & & \quad X := I; \\
 & & \quad \mathbf{do}\ A_1\ [\!]\ \ldots\ [\!]\ A_m\quad \mathbf{od} \\
 & &]| : Z
\end{array}
\qquad
\begin{array}{lll}
\mathcal{B} & \widehat{=} & |[\ \mathbf{var}\quad Y : S \\
 & & \quad Y := J; \\
 & & \quad \mathbf{do}\ B_1\ [\!]\ \ldots\ [\!]\ B_n\quad \mathbf{od} \\
 & &]| : U
\end{array}
$$

where Z are imported global variables to $\mathcal{A}$ and U are imported global variables to $\mathcal{B}$, and the local variables X and Y are assumed to be disjoint. The parallel composition $\mathcal{A} \,||\, \mathcal{B}$ is defined to be the action system

$$
\begin{array}{lll}
\mathcal{A} \,||\, \mathcal{B} & = & |[\ \mathbf{var}\quad X, Y : T, S \\
 & & \quad X, Y := I, J; \\
 & & \quad \mathbf{do}\ A_1\ [\!]\ \ldots\ [\!]\ A_m\ [\!]\ B_1\ [\!]\ \ldots\ [\!]\ B_n\quad \mathbf{od} \\
 & &]| : (Z \cup U) - (X \cup Y)
\end{array}
$$

The parallel composition combines the state spaces of the two action systems, merging the global variables and keeping the local variables distinct. Those imported global variables in $Z \cup U$, that are defined in either of the actions system, i.e. $(X \cup Y)$, are now locally declared.

The behaviour of the parallel composition depends on how the individual action systems, the *reactive components*, interact with each other via the global variables that they use. For instance, a reactive component does not terminate by itself. Termination is a global property of the composed action system [2].

Since the initialisations are well defined, and non-deterministic choice between actions 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.

6.1 Hybrid Action Systems

The parallel composition for action systems uses interleaving for parallelism. This works well for a discrete behaviour, but not for a continuous behaviour. Consider the previously presented gate example. Suppose, we try to compose another force lifting the gate with a constant velocity $0.5\,{}^{m}/_{s}$ whenever the gate is at least half a meter away from being closed. Composing this force by interleaving the earlier non-linear closing force does not produce the desired result. By interleaving, the gate would non-deterministically either close normally, or it would start to open. In reality, we would expect that the gate stops half closed. The problem with the interleaving composition is that it does not really allow a simultaneous effect of the components, which is what we would like to have with the continuous behaviour. Motivated by this observation we explore an alternative way of composing continuous behaviour.

Linear Composition. In linear composition two partially defined functions are added on their common domain, whereas in the remaining domains the functions retain their original meaning. Linear composition thus assumes that a function is zero outside the domain where it is defined. The linear composition is formalised as follows. Let f be a partially defined function in the domain $\mathbf{dom}\ f$. Also, let g be a partially defined function in the domain $\mathbf{dom}\ g$. The linear composition of these functions is a set of partial functions

$$f(x) \oplus g(x) \quad = \quad \begin{cases} f(x) + g(x), & x \in \ \mathbf{dom}\ f \cap \mathbf{dom}\ g \\ f(x) + 0, & x \in \ \mathbf{dom}\ f \cap \overline{\mathbf{dom}\ g} \\ 0 + g(x), & x \in \ \overline{\mathbf{dom}\ f} \cap \mathbf{dom}\ g \end{cases} \tag{7}$$

The linear composition is both associative and commutative. It is defined in the whole range of $\mathbf{dom}\ f \cup \mathbf{dom}\ g$, whereas ordinary addition is defined only in the range $\mathbf{dom}\ f \cap \mathbf{dom}\ g$.

Obviously, the linear composition can be generalised for partially defined systems of differential equations. Let $\mathcal{F}(X)$ and $\mathcal{G}(X)$ be sets of vectors

$$\mathcal{F}(X) \;=\; \begin{cases} F_1(X), & X \in \mathbf{dom}\ F_1 \\ \vdots & \\ F_m(X), & X \in \mathbf{dom}\ F_m \end{cases} \qquad \mathcal{G}(X) \;=\; \begin{cases} G_1(X), & X \in \mathbf{dom}\ G_1 \\ \vdots & \\ G_n(X), & X \in \mathbf{dom}\ G_n \end{cases}$$

where the domains $\mathbf{dom}\ F_1 \ldots \mathbf{dom}\ F_m$ are disjoint, as well as the domains $\mathbf{dom}\ G_1 \ldots \mathbf{dom}\ G_n$. A linearly composed system of differential equations

$$\dot{X} \;=\; \mathcal{F}(X) \;\oplus\; \mathcal{G}(X)$$

unfolds to a set of partially defined systems of differential equations

$$\begin{cases} \dot{X} \;=\; \mathcal{F}(X) + \mathcal{G}(X), & X \in \bigcup_{i=1}^{m} \mathbf{dom}\ F_i \;\cap\; \bigcup_{j=1}^{n} \mathbf{dom}\ G_j \\ \dot{X} \;=\; \mathcal{F}(X), & X \in \bigcup_{i=1}^{m} \mathbf{dom}\ F_i \;\cap\; \bigcap_{j=1}^{n} \overline{\mathbf{dom}\ G_j} \\ \dot{X} \;=\; \mathcal{G}(X), & X \in \bigcap_{i=1}^{m} \overline{\mathbf{dom}\ F_i} \;\cap\; \bigcup_{j=1}^{n} \mathbf{dom}\ G_j \end{cases}$$

Linear composition applies to stutter free differential actions. A stutter free differential action $e :\rightarrow\ \dot{X} = F(x)$ denotes a partially defined system of differential equations in the domain where e holds, that is $\{t \mid e[\phi(t)/X]\}$. A set of such systems with disjoint domains is a set of differential actions combined with non-deterministic choice.

Let DF be a composition $f_1 :\rightarrow \dot{X} = F_1(X)\ [\!]\ \ldots\ [\!]\ f_m :\rightarrow \dot{X} = F_m(X)$, and DG be a composition $g_1 :\rightarrow \dot{X} = G_1(X)\ [\!]\ \ldots\ [\!]\ g_n :\rightarrow \dot{X} = G_m(X)$. The DF and DG are analogous to $\dot{X} = \mathcal{F}(X)$ and $\dot{X} = \mathcal{G}(X)$. The linear composition of these functions presented above, is expressed in terms of actions as

$$\begin{array}{lll} [\!]\ i = 1..m, j = 1..n\ \cdot & f_i \;\wedge\; g_j & :\rightarrow \dot{X} = F_i(X) + G_j(X) \\ [\!]\ i = 1..m\ \cdot & f_i \;\wedge\; \neg\,\mathrm{g}DG & :\rightarrow \dot{X} = F_i(X) \\ [\!]\ j = 1..n\ \cdot & \neg\,\mathrm{g}DF \;\wedge\; g_j & :\rightarrow \dot{X} = G_j(X) \end{array}$$

where $\mathrm{g}DF$ is $f_1 \vee \ldots \vee f_m$ and $\mathrm{g}DG$ is $g_1 \vee \ldots \vee g_m$.

Example. Let X be a vector of variables x, y, and z. Let $F(X)$ be a vector of functions $(\,f(X), g(X), 0\,)$, and $G(X)$ be a vector of functions $(\,0, h(X), u(X)\,)$. Consider the following linear composed system of differential equations where e and i are predicates over X used for describing the domains.

$$\dot{X} \;=\; F(X),\ \{X \mid e\} \;\oplus\; G(X),\ \{X \mid i\}$$

This unfolds to

$$\begin{cases} \dot{X} \;=\; (\,f(X),\, g(X) + h(X),\, u(X)\,), & \{X \mid e \;\wedge\; i\} \\ \dot{X} \;=\; (\,f(X),\, g(X),\, 0\,), & \{X \mid e \;\wedge\; \neg i\} \\ \dot{X} \;=\; (\,0,\, h(X),\, u(X)\,), & \{X \mid \neg e \;\wedge\; i\} \end{cases}$$

We use the differential actions to denote the same linear composition of systems of differential equations as follows.

$$e :\rightarrow \dot{X} = F(x) \quad \oplus \quad i :\rightarrow \dot{X} = G(X)$$

This unfolds into the following non-deterministic composition

$$\begin{array}{lll} & e \wedge i & :\rightarrow \dot{x}, \dot{y}, \dot{z} = f(X), g(X) + h(X), u(X) \\ [] & e \wedge \neg i & :\rightarrow \dot{x}, \dot{y}, \dot{z} = f(X), g(X), 0 \\ [] & \neg e \wedge i & :\rightarrow \dot{x}, \dot{y}, \dot{z} = 0, h(X), u(X) \end{array}$$

6.2 Parallel composition of hybrid action systems

The separation of the continuous evolution and the discrete changes in a hybrid action system allows an extension of the parallel composition. We extend the parallel operator so, that it composes linearly the continuous evolution, but interleaves the discrete changes.

Let X and Y be disjoint local variables X, and Z and U be imported global variables. The ordinary actions H and stutter free differential actions DH may speak of variables X and Z, and similarly the ordinary actions K and stutter free differential actions DK may speak of variables Y and U. Consider the two hybrid action systems $\mathcal{H}$ and $\mathcal{K}$

$$\begin{array}{lll} \mathcal{H} & \hat{=} & |[\ \textbf{var}\ \ X : T \\ & & \quad X := I; \\ & & \quad \textbf{alt}\ H\ \textbf{with}\ DH \\ & &]| : Z \end{array} \qquad \begin{array}{lll} \mathcal{K} & \hat{=} & |[\ \textbf{var}\ \ Y : S \\ & & \quad Y := J; \\ & & \quad \textbf{alt}\ K\ \textbf{with}\ DK \\ & &]| : U \end{array}$$

where X and Y are disjoint. The parallel composition of these hybrid action systems has the following meaning.

$$\begin{array}{lll} \mathcal{H} \parallel \mathcal{K} & = & |[\ \textbf{var}\ \ X, Y : T, S \\ & & \quad X, Y := I, J; \\ & & \quad \textbf{alt}\ H\ []\ K\ \textbf{with}\ DH \oplus DK \\ & &]| : Z \cup U - (X \cup Y) \end{array}$$

The meaning is obtained by first unfolding the linear composition for the continuous evolution and then by unfolding the hybrid alternation as before.

Since both interleaving composition and linear composition are associative and commutative, this extended parallel composition is also associative and commutative. This means that the composition of several hybrid action systems can be unfolded in any order without affecting the result.

A very simple example of parallel composition occurs, when we take an autonomous system, given by:

$$\begin{array}{lll} \mathcal{S} & \hat{=} & |[\ \textbf{var}\ \ X^* : T \\ & & \quad X := I; \\ & & \quad \textbf{with}\ true :\rightarrow \dot{X} = F(X) \\ & &]| : \end{array}$$

and compose it with a control:

$$
\begin{array}{lll}
\mathcal{C} & \widehat{=} & |[\ \ \textbf{var}\ \ V : S \\
 & & \quad V := J; \\
 & & \quad \textbf{alt}\ \ \ \mathrm{g}K \rightarrow K \\
 & & \quad \textbf{with}\ e :\rightarrow \dot{X} = U(X, V) \\
 & &]| : X
\end{array}
$$

The parallel composition gives:

$$
\begin{array}{lll}
\mathcal{S} \parallel \mathcal{C} & = & |[\ \ \textbf{var}\ \ X^*, V : T, S \\
 & & \quad X, V := I, J; \\
 & & \quad \textbf{alt}\ \ \ \mathrm{g}K \rightarrow K \\
 & & \quad \textbf{with}\ \mathit{true} :\rightarrow \dot{X} = F(X) \quad \oplus \quad e :\rightarrow \dot{X} = U(X, V) \\
 & &]| :
\end{array}
$$

which during evolutions satisfies $\dot{X} = F(X) + U(X, V)$. Control has been added.

7 A Water Tank Example

Parallel composition is illustrated by a water tank example. The water tank is a leaking container that is being filled, see Figure 2. Both leak and filling are continuous processes within the physical limitations of the tank. In this example, the task is to model these two different continuous processes as separate components and to obtain a model for the composed behaviour.

At the beginning, the container is empty. It starts leaking as soon as the water level denoted by y is at or above the lower point of an outlet. This threshold is denoted by min. Filling starts right from the beginning, and stops when the water level reaches the lower point of the inlet. This water level is denoted by max. The water flows in at rate $1\ {}^{l}/_{s}$ and it flows out at rate $\frac{1}{2}\ {}^{l}/_{s}$.

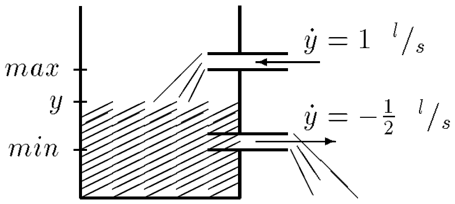

Figure 2: The water tank with threshold water levels (min and max) and flow rates.

7.1 Leaking

The water level y is a global variable, so that the filling process can share it. The hybrid system for the leaking consists only of an evolution describing the water flow from the tank whenever the water level is at or above min.

$$
\begin{array}{lll}
\mathcal{L}eak & \hat{=} & |[\ \mathbf{var}\quad y^* : I\!\!R \\
& & \quad y := 0; \\
& & \quad \mathbf{with}\ min \leq y :\to \dot{y} = -\frac{1}{2} \\
& &]| :
\end{array}
$$

7.2 Filling

We use the global variable y from the previous action system to describe the water level. A global variable $fill$ records whether the tank is being filled or not.

The hybrid filling system alternates between a discrete decision to stop the filling when the water level hits the max threshold and a continuous evolution describing the water flow into the tank during the filling.

$$
\begin{array}{lll}
\mathcal{F}ill & \hat{=} & |[\ \mathbf{var}\quad fill^* : I\!\!B \\
& & \quad fill := true; \\
& & \quad \mathbf{alt}\quad fill\ \wedge\ y \geq max \to fill := false \\
& & \quad \mathbf{with}\ fill\ \wedge\ y < max :\to \dot{y} = 1 \\
& &]| : y
\end{array}
$$

The difference between this and a continuous evolution **with** $y < max :\to \dot{y} = 1$ is that the hybrid action system above fills the tank only once.

7.3 Tank

The behaviour for the water tank is obtained through a parallel composition of the models for filling and leaking.

$$
\mathcal{T}ank \quad \hat{=} \quad \mathcal{F}ill \parallel \mathcal{L}eak
$$

Unfolding the meaning of the water tank into a hybrid action system we get

$$
\begin{array}{lll}
\mathcal{T}ank & = & |[\ \mathbf{var}\quad fill^*, y^* : I\!\!B, I\!\!R \\
& & \quad fill, y := true, 0; \\
& & \quad \mathbf{alt}\quad fill\ \wedge\ y \geq max \to fill := false \\
& & \quad \mathbf{with}\ fill\ \wedge\ y < max :\to \dot{y} = 1\ \oplus\ min \leq y :\to \dot{y} = -\frac{1}{2} \\
& &]| :
\end{array}
$$

Unfolding the linear composition yields

$$
\begin{array}{lllll}
\mathcal{T}ank & = & |[\ \mathbf{var} & fill^*, y^* : I\!\!B, I\!\!R & \\
& & \quad fill, y := true, 0; & & \\
& & \quad \mathbf{alt} & fill\ \wedge\ y \geq max & \to fill := false \\
& & \quad \mathbf{with} & fill\ \wedge\ y < min & :\to \dot{y} = 1 \\
& & \quad [\!] & fill\ \wedge\ min \leq y < max & :\to \dot{y} = \frac{1}{2} \\
& & \quad [\!] & (\neg fill\ \wedge\ min \leq y)\ \vee\ y \geq max & :\to \dot{y} = -\frac{1}{2} \\
& &]| : & &
\end{array}
$$

Finally, we translate the hybrid action system above into an ordinary action system. During the translation we use the properties for differential actions

(Section 3.6) to simplify the result. The behaviour of the water tank is given by the action system

$$
\begin{array}{lll}
\mathcal{T}ank & = & |[\ \textbf{var}\ \ fill^*, y^* : \mathbb{B}, \mathbb{R} \\
& & \quad fill, y := true, 0; \\
& & \quad \textbf{do}\ \ fill\ \wedge\ y \geq max \qquad\qquad \rightarrow\ fill := false \qquad\qquad\qquad (1) \\
& & \quad\ \ [\!]\ \ fill\ \wedge\ y < min \qquad\qquad \rightarrow\ y < min:\ \dot{y} = 1 \qquad\qquad (2) \\
& & \quad\ \ [\!]\ \ fill\ \wedge\ min \leq y < max\ \ \rightarrow\ min \leq y < max:\ \dot{y} = \frac{1}{2} \quad (3) \\
& & \quad\ \ [\!]\ \ \neg fill\ \wedge\ min \leq y \qquad\quad \rightarrow\ min \leq y:\ \dot{y} = -\frac{1}{2} \qquad\quad (4) \\
& & \quad \textbf{od} \\
& &]|:
\end{array}
$$

When we analyse the behaviour of this action system, we notice that after initialisation only action (2) is enabled. This means that the tank first fills up to the min level at full rate. Then the action (3) becomes enabled; this action fills the tank up to the max level with half filling rate. When the tank is full action (1) becomes enabled; terminating filling. Immediately, action (4) becomes enabled letting the tank leak until the water goes below the min level. At this point, there are no more actions enabled, so the system terminates.

A reusable filling system might be modelled by adding a control that resets $fill$.

7.4 A step towards HyTech

The action system for the water tank above can also be translated to the language of HyTech. The translation is merely a representation of the behaviour described above. As a first step we rewrite the verbal presentation into a hybrid automaton yielding Figure 3. In this automaton each state represents a differential action in the system. The boolean variable $fill$ is implicitly encoded in the names of the states. The discrete actions are seen as transition actions in between the states, and hence the discrete action (1) vanishes from the resulting automaton.

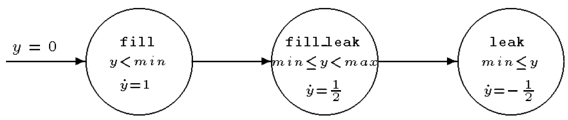

Figure 3: A hybrid automaton describing the behaviour of the water tank.

The automaton in Figure 3 is a linear hybrid automaton, and therefore it can be written in the language of HyTech. The following is the HyTech input for the water tank automaton.

```
-- a water tank
var y :  analog;  -- the water level

automaton tank
synclabs:;
initially fill & y=0;                               -- initialization
loc fill:  while y<min wait {y'=1}                      -- action (2)
    when True goto fill_leak;
loc fill_leak:  while min<=y & y<max wait {y'=1/2}  -- action (3)
    when True goto leak;
loc leak:  while min<=y wait {y'=-1/2}          -- action (1),(4)
end -- tank
```

This idea of translating a hybrid action system into HyTech works well, as long as the continuous evolutions are described by differential equations with constant right-hand side (Not to be confused with linear differential equations, where the right hand side is a linear expression in the variables). This limitation comes from the theoretical model of linear hybrid automata, which is used as a basis in the tool. For the non-linear evolutions a linearisation must be performed before they can be expressed in the language of HyTech.

8 Conclusions

In this paper, we presented differential equations as primitive actions. Its weakest liberal precondition semantics was derived from the one for a discrete accumulation that approximates the continuous evolution. Differential actions provide a compact way of describing and analysing hybrid behaviours fusing standard mathematical analysis with discrete mathematics in the framework of weakest liberal preconditions as seen in the gate example.

We also identified a class of hybrid action systems. They alternate between discrete actions and evolutions as typical for hybrid systems. Since the hybrid action systems are based on the use of differential actions where the time is implicitly present, we avoid the complication of a global time variable and an associated tick action.

Parallel composition of action systems is extended to cover hybrid action systems. The extension is based on the mathematical properties of a linear composition. Since the use of hybrid action systems and the extended parallel composition does not impose any further constraints, all the existing results for action systems are directly usable. Moreover, the parallel composition between hybrid action systems and ordinary action systems remain meaningful with the extended parallel operator.

The analysis of the hybrid behaviour in a hybrid action system is obtained by translating it into an ordinary action system. As illustrated with the water tank example, the translation is mechanical and based on the given general model. An existing tool, HyTech, can well be used for detailed analysis of a hybrid action system where the evolutions are described by differential equations with constant right-hand side or by approximating, cf. Theorem 3. This possibility was also illustrated with the water tank example.

These results are a foundation for further work. In particular, we shall investigate translations to HyTech of simple action systems. Furthermore, there is much remaining work in transferring results from control theory to the framework. Here, however, we consider it most fruitful to link this with specific classes of applications. A starting point could be a reformulation of Branicky's classification.

A very interesting idea is to study a variant of parallel composition, where the interaction between evolutions is through the differentiation variable. An example would be relative motion, where a person is walking with a speed $\frac{dp}{dx} = 1$ relative to a conveyor belt, satisfying $\dot{x} = f(x)$. Here parallel composition would be conjunction of the two equations. This is already included in the current definition of parallel composition for a single evolution; but what will happen during transitions?

Acknowledgments

We thank R. Back, E. Sekerinski, K. Sere, and the referees for their reading and comments for the earlier versions of this paper. This research was partially supported by the *Saastamoisen Säätiö* foundation. It has also been supported by the European Commission under Grant EC-US 043 Hybrid NSF. We would also like to thank the organisers of HS'97 for support.

References

[1] R. Alur, C. Couroubetis, T. A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel (eds.) *Hybrid Systems*, LNCS 736, Springer-Verlag, 1993.

[2] R. J. R. Back. *Refinement calculus, part II: Parallel and reactive programs.* In J. W. de Bakker, W.–P. de Roever, and G. Rozenberg, (eds.), *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. Proceedings. 1989*, LNCS 430, Springer-Verlag, 1990.

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

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

[5] M. Bonsangue and 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*, LNCS 666, pp. 91-110, Springer-Verlag, 1993.

[6] M. S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control.* PhD Thesis. Massachusetts Institute of Technology, EECS Dept, 1995.

[7] M. S. Branicky. General hybrid dynamical systems: modeling, analysis, and control. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, p. 186–200, Springer-Verlag, 1996.

[8] M. Broy and G. Nelson. Adding Fair Choice to Dijkstra's Calculus. *ACM Transactions on Prog. Languages and Systems*, **16**,3, pp. 924-938, 1994.

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

[10] K. Chandy, and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[11] K. M. Chandy. Reasoning about continuous systems. *Science of Computer Programming*, **14**, pp. 117–132, 1990.

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

[13] C.J. Fidge, A.J.Wellings. *An action-based formal model for concurrent, real-time systems.* Technical Report 95-1, SVR Center, January 1995.

[14] He Jifeng. From CSP to Hybrid Systems. In *A Classical Mind, Essays in Honour of C.A.R. Hoare*, pp. 171 - 189, Prentice Hall, 1994.

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

[16] T. A. Henzinger, P-H. Ho, and H. Wong-Toi. *A User Guide to HyTech.* `http://www-cad.eecs.berkeley.edu/~tah/Publications.`

[17] T. A. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pp. 377–388, 1996.

[18] Y. Kesten, Z. Manna, and A. Pnuelli. Verifying Clocked Transition Systems. In R. Alur, T. A. Henzinger, and E. D. Sontag (eds.) *Hybrid Systems III*, LNCS 1066, pp. 13-40, Springer-Verlag, 1996.

[19] R. Kurki-Suonio. Stepwise Design of Real Time Systems. *IEEE Transactions on Software Engineering*, **19**, 1, pp. 56-69, 1993.

[20] N. Lynch, R. Segala, F. Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pp. 496–510, 1996.

[21] M. Sintzoff. Abstract verification of structured dynamical systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pp. 126–137, Springer-Verlag, 1996.

[22] Xu Quiwen. Semantics and verification of extended phase transition systems in duration calculus. In O. Maler, editor, *Workshop on Hybrid and Real-Time Systems*, LNCS 1201, pp. 301–315, 1997.

[23] Chaochen Zhou, Wang Ji, and A. P. Ravn. A formal description of hybrid systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pp. 511–530. Springer-Verlag, 1996.

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

http://www.tucs.abo.fi

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

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

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