Differential Equations as Actions

M. Rönkkö

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

A. P. Ravn

Department of Computer Science, Technical University of Denmark,
bldg. 344, DK 2800 Lyngby, Denmark,
apr@it.dtu.dk

Turku Centre for Computer Science
TUCS Technical Report No 109
April 1997

ISBN 951-650-997-5
ISSN 1239-1891

Abstract

Differential equations are used in control theory to describe a system and its behaviour. They define explicitly the relationships between continuous variables during the evolution of the system. In this paper, we introduce the use of differential equations as primitive actions. The semantics is given with the weakest liberal precondition predicate transformer.

Keywords: actions, differential equations, weakest liberal precondition, continuous variables, hybrid systems

1 Introduction

Reasoning about the continuous evolution of variables is cumbersome with a traditional action. Consider the following example. A gate is lowered with a speed that is proportional to the current position of the gate. The further away the gate is from being closed, the quicker the gate closes. The task is to model this movement and to measure the time for the gate to become closed from a given initial position. This gate system can be analysed and the time spend by the movement can be presented as a function. However, to model the movement, one has to resort to discrete accumulation, the iteration of a $x := x - x \cdot \Delta t$ action. But how should the Δt be chosen if x represents the position the gate? Obviously, Δt should be a very small number, but how small number is small enough?

When analysing the continuous evolutions, the topic of interest is the relationships between the continuous variables. A well established way of denoting this relationship explicitly in control theory, is to use differential equations along with boundary expressions. The differential equations define implicitly all possible evolution paths of the variables from an initial value up to the boundary expressions.

The use of differential equations makes a design more robust, since similiar kind of systems are described by a single design. The differential equations can be modified to adjust the design with little need for modifying the overall design. In addition, the design becomes time independent. Time is implicitly present in the design, but to measure the passage of the time, one has to introduce a clock. The advantage of this is that the same design can be used to analyse the effect of the clock accuracy to the behaviour of the system.

In hybrid systems, the differential equations are also used for describing, how continuous time dependent variables behave in a state of a system, and what are the conditions for state transitions. For example, various hybrid automata frameworks are used for this purpose.

In this paper, we use action from the action systems framework to study the connection between discrete accumulation and continuous evolution. The purpose is to introduce an autonomous differential equation as a primitive action. We define it with the weakest liberal precondition semantics that corresponds to the one found for the approximative discrete accumulation.

1.1 Overview

We start by defining the conventional set of actions with the weakest liberal precondition in the chapter 2. The actions are then used in the chapter 3 to analyse the discrete accumulation and to obtain a weakest liberal precondition expression describing the accumulation process. In the chapter 4, we add the notion of implicitness to this expression. In the chapter 5, we apply the definition of differentiation to obtain an expression, where the discrete accumulation becomes continuous evolution. Finally, in the chapter 6, we return to the gate example. We study the problem both with discrete accumulation and with differential equations to illustrate the similarities and the differencies of these approaches.

2 Actions

Action systems, originally proposed by Back and Kurki-Suonio [2], are predicate transformer based systems that operate on some state base. Action systems have been applied in many non-trivial applications when modeling reactive and concurrent behaviour, see e.g.[3, 7]. In this paper, we restrict ourselves to individual actions instead of studying the parallel behaviour several action systems.

An action is any statement in an extended version of Dijkstra's guarded command language [8], where also the pure guarded commands are allowed as actions. The actions are defined with the *weakest precondition* predicate transformer $\mathrm{wp}(A, q)$ that returns the weakest precondition, from which the action A terminates and establishes the postcondition q. The *termination* of an action can also be given with the weakest precondition as a function

$$\mathrm{t}A \quad = \quad \mathrm{wp}(A, true)$$

In this paper, we give the semantics for the actions with the *weakest liberal precondition* (wlp). The motivation for this comes from reactive systems, where non-termination is often considered as a correct behaviour. The relation between the weakest precondition and the weakest liberal precondition is given by Dijkstra [8] as

$$\mathrm{wp}(A, q) \quad = \quad \mathrm{wlp}(A, q) \ \wedge \ \mathrm{t}A$$

We also need the definition of the *guard condition*. It returns all those states, in which the action A is *enabled*. An action is said to be enabled, if its guard condition is different from $false$. Therefore, an action is always enabled, if its guard condition equals to $true$. The guard condition is the function

$$\mathrm{g}A \quad = \quad \neg\mathrm{wp}(A, false)$$

Let the $q[e/x]$ denote the textual substitution of the free occurrences of variables x with the expressions e in the predicate q. Furthermore, let A and B be actions, and p be a predicate. The actions are defined as [4, 6]

$$
\begin{array}{lcl}
\mathrm{wlp}(abort, q) & = & true \\
\mathrm{wlp}(skip, q) & = & q \\
\mathrm{wlp}(x := e, q) & = & q[e/x] \\
\mathrm{wlp}(A; B, q) & = & \mathrm{wlp}(A, \mathrm{wlp}(B, q)) \\
\mathrm{wlp}(A \mathbin{[\!]} B, q) & = & \mathrm{wlp}(A, q) \ \wedge \ \mathrm{wlp}(B, q) \\
\mathrm{wlp}(p \to A, q) & = & p \ \Rightarrow \ \mathrm{wlp}(A, q)
\end{array}
$$

The iteration is given by the (liberal) partial correctness as

$$\mathrm{wlp}(\mathbf{do}\ A\ \mathbf{od}, q) \ = \ \forall_{\,n \geq 0} \,.\, \mathrm{wlp}(A^n, \mathrm{g}A \ \vee \ q)$$

3 Discrete accumulation

The value of a variable is considered to be accumulated, when it starts from an initial value and steps through a sequence of values. The accumulation stops, when the value is outside a given boundary. This discrete accumulation is modeled with an iteration action. Let x be an accumulated variable, $f(x)$ be an accumulation step, and e be a boundary expression. The fixed value h is an accumulation coefficiant in respect to the linear progress modeled by the iteration itself. The accumulation process is captured by

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

We can study the behaviour of the accumulation by analysing its weakest liberal precondition. The weakest liberal precondition for the iteration given earlier is not very informative. Therefore, we introduce a bit of a notation. Let $p[{}^{v}/_{x}]^{i}$ denote the sequential substitution of free variables x in predicate p with values v, where no substitution takes place when $i = 0$. Also, let q stand for a postcondition. We obtain an expression

$$
\begin{array}{cl}
 & \mathrm{wlp}(\mathbf{do}\ \ e \ \to \ x := x + f(x) \cdot h \ \ \mathbf{od}, \ q) \\
= & \{ \ \textit{definition of } \mathrm{wlp} \textit{ and series simplification} \ \} \\
 & \forall_{\,n \geq 0} \,.\, (e \ \vee \ q)[{}^{x+hf(x)}/_{x}]^{n} \ \vee \ \exists_{\,0 \leq m < n} \,.\, \neg e[{}^{x+hf(x)}/_{x}]^{m}
\end{array}
$$

where n and m correspond to the iteration rounds and, thus, belong to natural numbers. The derivation of this expression is in more detail in appendix A.1.

From the expression above, we can see that for any iteration round, either the iteration guard e is enabled, or the postcondition holds, or the iteration guard was disabled during the previous round. This closure form guarantees that the postcondition is evaluated only during the first occurance of the guard disabledness. During the other iteration rounds the expression evaluates to true, because the iteration guard can only be either enabled or disabled. The figure 1 illustrates this closure form, where the iteration guard becomes disabled after the third iteration round.

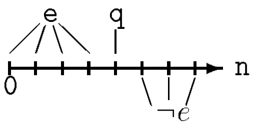

Figure 1: Partial correctness for accumulative iteration.

4 Implicitness

The weakest liberal precondition for the discrete accumulation expresses explicitly, how the value of x is modified with the step $f(x)$ during each iteration round. Although the notation for the sequential substitution allows a compact representation, it does not provide much help in the evaluation of the expression. Therefore, we change the sequential substitution into a single substitution.

One way of accomplishing this is to introduce a recursive accumulation function Φ, such that for a fixed $h > 0$, the value $\Phi(n \cdot h)$ equals to the expression $[{}^{x+hf(x)}/_{x}]^n$. Let i and n belong to natural numbers. The function Φ is

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

However, the evaluation of a sum of a recursive function is not very practical, either. Fortunately, this particular function can be expressed implicitly as

$$\Phi(0) = x, \quad \forall_{i \geq 0} \, . \; \frac{\Phi(i \cdot h + h) - \Phi(i \cdot h)}{h} = f(\Phi(i \cdot h)) \tag{1}$$

This implicit form is also more useful, as it states the difference between two successive accumulation rounds. The weakest liberal precondition for the discrete accumulation can now be expressed without a sequential substitution as

$$
\begin{array}{rl}
& \mathrm{wlp}(\mathbf{do}\ \ e \ \rightarrow\ x := x + f(x)\cdot h \ \ \mathbf{od},\ q) \\
= & \{\ \textit{previous form and implicitness}\ \} \\
& \exists_{\Phi}\ .\ \Phi(0) = x \ \wedge\ \forall_{i>0}\ .\ \frac{\Phi(i\cdot h+h)-\Phi(i\cdot h)}{h} = f(\Phi(i\cdot h)) \ \wedge \\
& (\forall_{n\geq 0}\ .\ (e \ \vee\ q)[{}^{\Phi(n)}/_{x}] \ \vee\ \exists_{0\leq m<n}\ .\ \neg e[{}^{\Phi(m)}/_{x}])
\end{array}
$$

5 Continuous evolution

The iteration of the accumulation is performed in a discrete domain of natural numbers. The variable x can, of course, belong to real numbers, but the finiteness of the iteration domain limits the values of x to be discrete. Therefore, it is impossible to express an iteration, which totally covers an interval of real numbers. This makes it difficult to detect exactly, when the boundary expression e changes. To overcome these problems, we define the iteration in a continuous domain of real numbers.

The continuous evolution is described by a continuous function. The iteration step of the evolution corresponds to the first derivative of the function. By applying the definition of differentiation to the implicit formula for the accumulation (1), we obtain the evolution condition. This yields a differential equation with an initial value problem

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

With this implicit formula, we can define the weakest liberal precondition semantics for a new action, that allows continuous evolution of variables. The differential action is defined as

$$
\begin{array}{rcl}
\mathrm{wlp}(\dot{x} = f(x)\ \mathbf{inv}\ e,\ q) & = & \exists_{\Phi}\ .\ \Phi(0) = x \ \wedge\ \dot{\Phi} = f(\Phi) \ \wedge \\
& & (\forall_{\tau\geq 0}\ .\ (e \ \vee\ q)[{}^{\Phi(\tau)}/_{x}] \ \vee \\
& & \exists_{0\leq\delta<\tau}\ .\ \neg e[{}^{\Phi(\delta)}/_{x}])
\end{array}
$$

where τ and δ belong to real numbers. In this action, the variable x is continuously evolved as long as the invariant (boundary expression) e holds. The evolution is determined by the differential equation $\dot{x} = f(x)$. Therefore, the calculation of the weakest liberal precondition requires the existence of the solution for the differential equation, which is stated by the first two conjuncts.

5.1 Multiple variables

In order to capture the dependencies between continuous variables, multiple differential equations are allowed to appear. In this case, the substitution takes place similarly to the one used in multiple assignment statement. However, care should be taken in writing out the dependent form of the weakest liberal precondition. The following is an example of the weakest liberal precondition for the evolution of two continuous variables x and y. Here, x is dependent on y.

$$
\begin{array}{ll}
 & \mathrm{wlp}(\dot{x}, \dot{y} = y, 1 \ \mathbf{inv}\ e, \quad q) \\
= & \{ \ \mathit{definition\ of}\ \mathrm{wlp} \ \} \\
 & \exists_{\Phi_x, \Phi_y} \,.\, \Phi_x(0) = x \ \wedge \ \dot{\Phi}_x = \Phi_y \ \wedge \ \Phi_y(0) = y \ \wedge \ \dot{\Phi}_y = 1 \ \wedge \\
 & \qquad (\forall_{\tau \geq 0} \,.\, (e \ \vee \ q)[^{\Phi_x(\tau), \Phi_y(\tau)}/_{x,y}] \\
 & \qquad \ \vee \ \exists_{0 \leq \delta < \tau} \,.\, \neg e \ [^{\Phi_x(\delta), \Phi_y(\delta)}/_{x,y}])
\end{array}
$$

6 Clocks

The passage of the time is measured with a clock. A discrete clock is an iterated action $c := c + \Delta$, where c is the current value of the clock and Δ is the size of one tick. A discrete clock suffers from the granularity of a tick. It never reaches all of the values in any continuous interval. Therefore, all the variables that depent on this clock are also defined only at some discrete points of the time. This causes problems with accuracy, in particular when comparing the values of these variables.

An accurate clock is modeled with the differential action as $\dot{c} = 1$ **inv** e. It is a continuous variable that evolves at the same rate as the time. Simultaneous evolution of continuous time dependent variables is modeled as multiple variables in a differential action. Since the differential action translates the variables into piecewise continuous functions, there are no such granularity problems as with the discrete clocks.

The following is an example of an accurate clock. It is a simple alarm clock that is switched on at one o'clock, and stops at five o'clock, sharp.

$$c := 1; \quad \dot{c} = 1 \ \mathbf{inv}\ c < 5$$

One might replace this alarm clock with a single action assigning the value 5 to the variable c. However, that would not model the passage of the time. It would be just an assignment statement in a discrete point of the time.

7 The gate problem

We return now back to the gate problem described in the introduction of this paper. The gate is lowered with a speed that is proportional to the current position of the gate, that is $\dot{x} = -x$. The task is to model the movement of the gate, and to measure the time for the gate to become closed. Figure 2 illustrates the problem graphically. We will now analyse a case, where the initial position of the gate is 1 m away. The gate in considered to be closed, when it is at most 1 cm away.

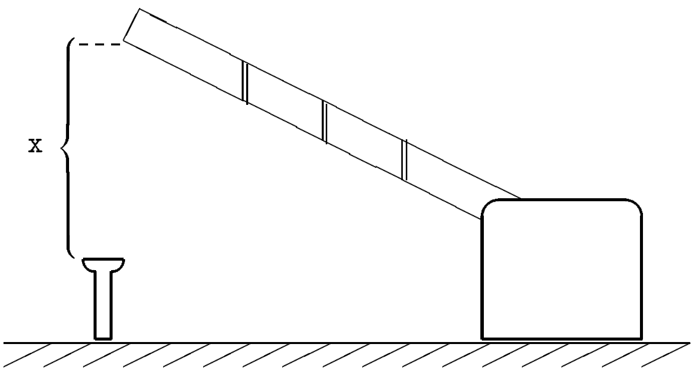

Figure 2: The gate.

7.1 With discrete accumulation

The gate problem is described with the discrete accumulation as a sequential composition of actions

$$x := 1;\ \mathbf{do}\ x > 0.01 \rightarrow x := x - x \cdot \Delta\ \mathbf{od}$$

In this, the Δ models the linear progression of the time. Thus, the total time spent by the iteration is a sum $\sum_{i=1}^{n} \Delta$, where n is the number of performed iterations.

We can analyse this with the weakest liberal precondition.

$$
\begin{array}{rl}
& \mathrm{wlp}(x := 1;\ \mathbf{do}\ x > 0.01 \rightarrow x := x - x \cdot \Delta\ \mathbf{od}, x \leq 0.01) \\
= & \{\ \textit{definition of}\ \mathrm{wlp}\ (\textit{implicit}\ \Phi)\ \ \} \\
& \mathrm{wlp}(\ x := 1,\ \exists_{\Phi}\ .\ \Phi(0) = x\ \wedge \\
& \qquad\qquad \forall_{i>0}\ .\ \frac{\Phi(i\cdot\Delta+\Delta)-\Phi(i\cdot\Delta)}{\Delta} = -\Phi(i\cdot\Delta)\ \wedge \\
& \qquad\qquad (\forall_{n\geq 0}\ .\ (x > 0.01\ \ \vee\ \ x \leq 0.01)[{}^{\Phi(n)}/_{x}]\ \vee \\
& \qquad\qquad \exists_{0\leq m<n}\ .\ \neg x > 0.01[{}^{\Phi(m)}/_{x}])\ \ ) \\
= & \{\ \textit{instantiation of}\ \Phi\ \textit{and simplification}\ \ \} \\
& \mathrm{wlp}(\ x := 1,\ \textit{true}\ ) \\
= & \{\ \textit{definition of}\ \mathrm{wlp}\ \ \} \\
& \textit{true}
\end{array}
$$

Essentially, this tells us that the construct is usable, only if there exists the accumulation function Φ. We can prove this by analysing the series of substitutions that take place in the iteration.

$$
\begin{array}{rl}
& x,\ x - x\cdot\Delta,\ x - x\cdot\Delta - (x - x\cdot\Delta)\cdot\Delta, \\
& (x - x\cdot\Delta - (x - x\cdot\Delta)\cdot\Delta) - (x - x\cdot\Delta - (x - x\cdot\Delta)\cdot\Delta)\cdot\Delta, \\
& \ldots \\
= & \{\ \textit{common factor}\ \ \} \\
& x,\ x(1-\Delta),\ x(1-\Delta) - (x(1-\Delta))\cdot\Delta, \\
& (x(1-\Delta) - (x(1-\Delta))\cdot\Delta) - (x(1-\Delta) - (x(1-\Delta))\cdot\Delta)\cdot\Delta, \\
& \ldots \\
= & \{\ \textit{common factor}\ \ \} \\
& x,\ x(1-\Delta),\ x(1-\Delta)(1-\Delta), \\
& x(1-\Delta)(1-\Delta) - (x(1-\Delta)(1-\Delta))\cdot\Delta,\ \ldots \\
= & \{\ \textit{series simplification}\ \ \} \\
& x(1-\Delta)^0,\ x(1-\Delta)^1,\ x(1-\Delta)^2,\ x(1-\Delta)^3,\ \ldots
\end{array}
$$

From this it is possible to see that

$$\Phi(n\cdot\Delta) = x(1-\Delta)^n$$

generates this series. It also satisfies the progress condition stated in the weakest liberal precondition, and thus, is the accumulation function Φ in question.

Next, we verify that with this design, the gate eventually becomes closed, i.e. the termination of the discrete accumulative iteration. The derivation of the weakest precondition for the termination is given in appendix A.2.

$$
\begin{aligned}
& \mathrm{wp}(x := 1;\ \mathbf{do}\ x > 0.01 \rightarrow x := x - x \cdot \Delta\ \mathbf{od}, \mathit{true}) \\
= \ & \{\ \mathit{definition\ of}\ \mathrm{wp}\ (\mathit{implicit}\ \Phi)\ \} \\
& \mathrm{wp}(\ x := 1,\ \exists_{\Phi}\ .\ \Phi(0) = x\ \wedge \\
& \qquad\qquad \forall_{i>0}\ .\ \tfrac{\Phi(i\cdot\Delta+\Delta)-\Phi(i\cdot\Delta)}{\Delta} = -\Phi(i\cdot\Delta)\ \wedge \\
& \qquad\qquad \exists_{n\geq 0}\ .\ \neg x > 0.01[\,{}^{\Phi(n)}/_{x}]\ \ ) \\
= \ & \{\ \mathit{instantiation\ of}\ \Phi\ \mathit{and\ simplification}\ \} \\
& \mathrm{wp}(\ x := 1,\ \exists_{n\geq 0}\ .\ x \leq 0.01[\,{}^{x(1-\Delta)^n}/_{x}]\ \ ) \\
= \ & \{\ \mathit{substitution\ and\ definition\ of}\ \mathrm{wp}\ \} \\
& \exists_{n\geq 0}\ .\ (1-\Delta)^n \leq 0.01
\end{aligned}
$$

This holds, only if $\Delta > 0$. However, this condition is a bit misleading. For example, when modeling the movement of the gate with $\Delta = 2$ the system terminates and establishes the condition $x \leq 0.01$. But, the value of the x becomes -1 at the termination. This is interpreted as the bar of the gate hitting 1 m below the wanted position after 2 seconds. Clearly, the use of big Δ values produces very erroneous results.

We can approach this problem by analysing the inequation $(1-\Delta)^n \leq 0.01$ numerically. By increasing the number of required iterations, i.e. the value of n, we obtain smaller and smaller values for Δ. However, the sum $\sum_{i=1}^{n} \Delta$ increases less and less, and it seems to approach approximately a number of 4.6 . This indicates that the value of the Δ should be minimal, but bigger than zero.

In the case of the gate problem, it is possible to calculate exactly the time spent by the lowering of the gate. We start by solving the Δ from the boundary expression

$$
\begin{aligned}
& (1-\Delta)^n = 0.01 \\
= \ & \{\ \mathit{applying}\ e^x\ \} \\
& e^{n\cdot\ln(1-\Delta)} = e^{\ln 0.01} \\
= \ & \{\ \mathit{applying}\ \ln\ \mathit{and\ division\ by}\ n\ \} \\
& \ln(1-\Delta) = \tfrac{1}{n}\ln 0.01 \\
= \ & \{\ \mathit{applying}\ e^x\ \mathit{and\ manipulation}\ \} \\
& \Delta = 1 - e^{\frac{1}{n}\ln 0.01}
\end{aligned}
$$

Next, we insert this into the expression stating the total time spent by the lowering process, and obtain

$$
\begin{aligned}
& \textstyle\sum_{i=1}^{n} 1 - e^{\frac{1}{n}\ln 0.01} \\
= \ & \{\ \mathit{series\ simplification}\ \} \\
& n(1 - e^{\frac{1}{n}\ln 0.01})
\end{aligned}
$$

where n represents the number of iterations needed to reach the value 0.01 from the initial value of 1.

Before we can analyse the limit of this expression, when n approaches infinity, we replace the exponential function with a tailor ploynomial approximation

$$e^{\frac{1}{n}\ln 0.01} \quad = \quad 1 + \tfrac{1}{n}\ln 0.01 + c \cdot (\tfrac{1}{n})^2$$

where c is some constant value representing the remainder of the polynomial. This modifies the expression into

$$\begin{array}{ll} & n(1 - 1 - \frac{1}{n}\ln 0.01 - c \cdot (\frac{1}{n})^2) \\ = & \{ \ \mathit{simplification} \ \} \\ & -\ln 0.01 - \frac{c}{n} \\ = & \{ \ n \to \infty \ \} \\ & -\ln 0.01 \end{array}$$

Thus, the total time spent by the lowering of the gate is $-\ln 0.01 \ \approx 4.61$.

7.2 With continuous evolution

The gate problem is expressed with the continuous evolution as

$$x := 1; \ \dot{x} = -x \ \textbf{inv} \ x > 0.01$$

We also introduce a clock repesented by the variable c. The clock measures the time spent by the process. Thus, we obtain the following

$$
\begin{array}{ll}
& \mathrm{wlp}(x, c := 1, 0;\ \dot{x}, \dot{c} = -x, 1\ \mathbf{inv}\ x > 0.01,\quad x \le 0.01\ \wedge\ r = c) \\
= & \{\ \mathit{definition\ of}\ \mathrm{wlp}\ \ \} \\
& \mathrm{wlp}(x, c := 1, 0; \\
& \qquad \exists_{\Phi_x, \Phi_c} \,.\, \Phi_x(0) = x\ \wedge\ \dot{\Phi}_x = -\Phi_x\ \wedge \\
& \qquad\qquad\qquad \Phi_c(0) = c\ \wedge\ \dot{\Phi}_c = 1\ \wedge \\
& \qquad (\forall_{\tau \ge 0} \,.\, (x > 0.01\ \vee\ (x \le 0.01\ \wedge\ r = c))[\,{}^{\Phi_x(\tau), \Phi_c(\tau)}/_{x,c}] \\
& \qquad\quad \vee\ \exists_{0 \le \delta < \tau} \,.\, \neg(x > 0.01)[\,{}^{\Phi_x(\delta)}/_{x}])\ ) \\
= & \{\ \mathit{simplification\ and\ instantiation\ of}\ \Phi_x(t) = xe^{-t}\ \ \} \\
& \mathrm{wlp}(x, c := 1, 0; \\
& \qquad \exists_{\Phi_c} \,.\, \Phi_c(0) = c\ \wedge\ \dot{\Phi}_c = 1\ \wedge \\
& \qquad (\forall_{\tau \ge 0} \,.\, (x > 0.01\ \vee\ r = c)[\,{}^{xe^{-\tau}, \Phi_c(\tau)}/_{x,c}] \\
& \qquad\quad \vee\ \exists_{0 \le \delta < \tau} \,.\, \neg(x > 0.01)[\,{}^{xe^{-\delta}}/_{x}])\ ) \\
= & \{\ \mathit{substitution\ and\ instantiation\ of}\ \Phi_c(t) = c + t\ \ \} \\
& \mathrm{wlp}(x, c := 1, 0; \\
& \qquad \forall_{\tau \ge 0} \,.\, (xe^{-\tau} > 0.01\ \vee\ r = c)[\,{}^{c+\tau}/_{c}] \\
& \qquad\quad \vee\ \exists_{0 \le \delta < \tau} \,.\, \neg(xe^{-\delta} > 0.01)\ ) \\
= & \{\ \mathit{substitution\ and\ simplification}\ \ \} \\
& \mathrm{wlp}(x, c := 1, 0; \\
& \qquad \forall_{\tau \ge 0} \,.\, (xe^{-\tau} > 0.01\ \vee\ r = c + \tau) \\
& \qquad\quad \vee\ \exists_{0 \le \delta < \tau} \,.\, xe^{-\delta} \le 0.01) \\
= & \{\ \mathit{definition\ of}\ \mathrm{wlp}\ \ \} \\
& \forall_{\tau \ge 0} \,.\, e^{-\tau} > 0.01\ \vee\ r = \tau\ \vee\ \exists_{0 \le \delta < \tau} \,.\, e^{-\delta} \le 0.01 \\
= & \{\ \mathit{case\ analysis\ around}\ \tau = -\ln 0.01\ \ \} \\
& (\forall_{0 \le \tau < -\ln 0.01} \,.\, e^{-\tau} > 0.01\ \vee\ r = \tau\ \vee\ \exists_{0 \le \delta < \tau} \,.\, e^{-\delta} \le 0.01)\ \wedge \\
& (\exists_{\tau = -\ln 0.01} \,.\, e^{-\tau} > 0.01\ \vee\ r = \tau\ \vee\ \exists_{0 \le \delta < \tau} \,.\, e^{-\delta} \le 0.01)\ \wedge \\
& (\forall_{-\ln 0.01 < \tau} \,.\, e^{-\tau} > 0.01\ \vee\ r = \tau\ \vee\ \exists_{0 \le \delta < \tau} \,.\, e^{-\delta} \le 0.01) \\
= & \{\ \mathit{simplification}\ \ \} \\
& e^{\ln 0.01} > 0.01\ \vee\ r = -\ln 0.01\ \vee\ \exists_{0 \le \delta < -\ln 0.01} \,.\, e^{-\delta} \le 0.01 \\
= & \{\ \mathit{simplification}\ \ \} \\
& \mathit{false}\ \vee\ r = -\ln 0.01\ \vee\ \mathit{false} \\
= & \{\ \mathit{simplification}\ \ \} \\
& r = -\ln 0.01
\end{array}
$$

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

8 Conclusions

In this paper, we presented a way of using differential equations as primitive actions. The weakest liberal precondition semantics was derived from the discrete accumulation that approximates the continuous evolution described by the differential equations implicitly. We also presented an implicit version of the weakest liberal precondition for the discrete accumulation.

Differential equations as actions provide a compact way of describing continuous behaviour. Also, the analysis of continuous systems is more direct with differential actions than with the discrete accumulation. In a way, the differential action is an abstract design of a mathematical model. The weakest liberal preconditions are then used as a tool to extract this model. The gate example presented in this paper illustrates this approach.

One application for the differential actions is to model the clocks in real-time systems. In comparison to the clocked transition system (CTS) by [9], the clocks are presented as ordinary variables. The transitions are sequences of actions modifying the variables in the system. The master clock is a global variable that is modified only by the tick action. The tick action is a differential action that updates all the clocks including the master clock, and thus, represents the tick transition in the CTS. The time progress condition stated in the CTS is seen as a global invariance property in an action system.

However, the hybrid systems offers a better application field for differential actions. Autonomous switching and autonomous jumps in Branicky's unified model [5] are presented as a sequence differential actions and ordinary assignments respectively. Similarly, the controlled switching and jumps in the unified model are presented as nondeterministic conditional statements with differential actions and ordinary assignments.

There exists also various hybrid automata models for the hybrid systems. For example, in the hybrid autotmata model by [1], the locations with continuous variable updates are presented as differential actions. In this model, the transitions between the locations are presented as sequence of actions between the differential actions.

As future work, the aim is to study the parallellism in hybrid systems with the differential actions. The action systems themselves provide a good framework for this purpose. Naturally, more time will be spent on studying the similarities and the differencies between other existing approaches and our new approach.

Acknowledgments

We thank R. Back, E. Sekerinski, and K. Sere for their reading and comments. This research was partially supported by the *Saastamoisen Säätiö* foundation.

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 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*, pages 131–142, 1983.

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

[4] 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, p. 91-110, Springer-Verlag, 1993.

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

[6] M. Broy and G. Nelson. Adding Fair Choice to Dijkstra's Calculus. *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, p. 924-938, May 1994.

[7] M. Butler, E. Sekerinski, and K. Sere. Steam boiler specification and refinement. To be published, preliminary version presented at the Seminar on *Methods for Semantics and Specification*, Schloss Dagstuhl, June 1995.

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

[9] 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, Springer-Verlag, 1996.

A Derivations

A.1 Accumulative iteration

$$
\begin{array}{cl}
& \mathrm{wlp}(\mathbf{do}\ \ e\ \rightarrow\ x := x + f(x)\cdot h\ \ \mathbf{od}, q) \\
= & \{\ \ definition\ of\ \mathrm{wlp}\ \ \} \\
& \forall_{\,n\geq 0}\ .\ \mathrm{wlp}((e\ \rightarrow\ x := x + f(x)\cdot h)^n, e\ \vee\ q) \\
= & \{\ \ unfolding\ \mathrm{wlp}\ and\ \mathrm{t}(e\ \rightarrow\ x := x + f(x)\cdot h)\ \ \} \\
& (e\ \vee\ q)\ \wedge \\
& (\neg e\ \vee\ (e\ \vee\ q)[{}^{x+f(x)\cdot h}/_{x}])\ \wedge \\
& (\neg e\ \vee\ (\neg e\ \vee\ (e\ \vee\ q)[{}^{x+f(x)\cdot h}/_{x}])[{}^{x+f(x)\cdot h}/_{x}])\ \wedge \\
& \ldots \\
= & \{\ \ series\ simplification\ \ \} \\
& \forall_{\,n\geq 0}\ .\ (e\ \vee\ q)[{}^{x+hf(x)}/_{x}]^n\ \ \vee\ \ \exists_{\,0\leq m<n}\ .\ \neg e[{}^{x+hf(x)}/_{x}]^m
\end{array}
$$

where an expression $p[v/x]^i$ denotes the sequential substitution of free variables x in predicate p with values v. No substitution takes place, when $i = 0$.

A.2 Derivation of termination

The weakest precondition for the iteration actions is defined as

$$
\mathrm{wp}(\mathbf{do}\ A\ \mathbf{od}, q)\ \ =\ \ (\forall_{\,n\geq 0}\ .\ \mathrm{wp}(A^n, \mathrm{g}A\ \vee\ q))\ \wedge\ (\exists_{\,n\geq 0}\ .\ \neg\mathrm{g}A^n)
$$

In the case of an accumulative iteration we can simplify this as follows

$$
\begin{array}{cl}
& \mathrm{wp}(\mathbf{do}\ \ e\ \rightarrow\ x := x + f(x)\cdot h\ \ \mathbf{od},\ true) \\
= & \{\ \ definition\ of\ \mathrm{wp}\ \ \} \\
& \forall_{\,n\geq 0}\ .\ \mathrm{wp}((e\ \rightarrow\ x := x + f(x)\cdot h)^n, e\ \vee\ q)\ \vee \\
& (\exists_{\,n\geq 0}\ .\ \neg\mathrm{g}(e\ \rightarrow\ x := x + f(x)\cdot h)^n) \\
= & \{\ \ unfolding\ \mathrm{wp}\ and\ \mathrm{g}\ \ \} \\
& true\ \wedge \\
& (\neg e\ \vee\ (true)[{}^{x+f(x)\cdot h}/_{x}])\ \wedge \\
& (\neg e\ \vee\ (\neg e\ \vee\ (true)[{}^{x+f(x)\cdot h}/_{x}])[{}^{x+f(x)\cdot h}/_{x}])\ \wedge \\
& \ldots\ \wedge \\
& (\neg e\ \vee \\
& \ \neg e\ \vee\ \neg e[{}^{x+f(x)\cdot h}/_{x}]\ \vee \\
& \ \neg e\ \vee\ (\neg e\ \vee\ \neg e[{}^{x+f(x)\cdot h}/_{x}])[{}^{x+f(x)\cdot h}/_{x}]\ \ \vee\ \ \ldots) \\
= & \{\ \ series\ simplification\ \ \} \\
& \exists_{\,n\geq 0}\ .\ \neg e[{}^{x+hf(x)}/_{x}]^n
\end{array}
$$

By using implicit notation presented in the chapter 4, the termination condition gets the form

$$
\begin{array}{ll}
 & \mathrm{wp}(\mathbf{do}\ \ e\ \rightarrow\ x := x + f(x)\cdot h\ \ \mathbf{od},\ \mathit{true}) \\
= & \{\ \mathit{implicitness}\ \} \\
 & \exists_{\,\Phi}\ .\ \Phi(0) = x\ \wedge\ \forall_{\,i>0}\ .\ \frac{\Phi(i\cdot h+h)-\Phi(i\cdot h)}{h} = f(\Phi(i\cdot h))\ \wedge \\
 & \exists_{\,n\geq 0}\ .\ \neg e[\,{}^{\Phi(n)}/_{x}]
\end{array}
$$

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

http://www.tucs.abo.fi

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

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

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