Universität Augsburg

Non-Smooth and Zeno Trajectories for Hybrid System Algebra

Peter Höfner Bernhard Möller

Report 2006-07 March 2006

Institut für Informatik
D-86135 Augsburg

Non-Smooth and Zeno Trajectories for Hybrid System Algebra

Peter Höfner* and Bernhard Möller

Institut für Informatik, Universität Augsburg
D-86135 Augsburg, Germany
{hoefner,moeller}@informatik.uni-augsburg.de

Abstract. Hybrid systems are heterogeneous systems characterised by the interaction of discrete and continuous dynamics. In this paper we compare a slightly extended version of our earlier algebraic approach to hybrid systems to other approaches. We show that hybrid automata, which are probably the standard tool for describing hybrid systems, can conveniently be embedded into our algebra. But we allow general transition functions, not only smooth ones. Moreover we embed other models and point out some important advantages of the algebraic approach. In particular, we show how to easily handle Zeno effects, which are excluded by most other authors. The development of the theory is illustrated by a running example and a larger case study.

1 Introduction

Hybrid systems are dynamical as well as heterogeneous systems characterised by the interaction of discrete and continuous dynamics. Many applications, such as automated highway systems, air-traffic control, automotive controllers robotics, chemical and biological processes, are known. Hybrid automata (HA) [2, 11] are widely popular for designing and modelling hybrid systems. They are based on timed automata [4], a variation of finite state machines. After a short introduction we give in Section 2 the exact definition of HA. Afterwards, in Section 3, we present a slightly extended version of our algebraic model of hybrid systems [12]. Section 4 shows that this model forms a left semiring. Furthermore, it contains a very brief overview of the algebraic theory necessary for the paper. Section 5 presents one of the main results, viz. that Hybrid automata can be conveniently embedded into our algebra. But we allow general transition functions, not only smooth ones. Therefore our model is more expressive as HA. We further show how to embed other models and point out some important advantages of the algebraic approach. The following two sections discuss composition of hybrid systems as well as how to handle Zeno effects. The latter can easily be handled in the algebra, whereas in other models they are excluded by most other authors. To round off the paper and to illustrate our theory in an explicit example, in Section 8 we present a case study and also discuss some aspects of safety.

* This research was supported by DFG (German Research Foundation)

2 Hybrid Automata

In [2], hybrid systems are modeled by hybrid traces and hybrid automata. As we will show later on, hybrid traces are related to an algebra of hybrid systems. First, we want to recapitulate the definition of hybrid automata.

A *hybrid automaton* (HA) H [2, 1, 8, 11] consists of the following components.

Variables A finite set $X = \{x_1, \dots, x_n\}$ of real-valued variables. The number n is called the *dimension* of H. We write $\dot{X}$ for the set $\{\dot{x}_1, \dots, \dot{x}_n\}$ of dotted variables, which represent the first derivatives of the x_i during continuous change. We write X' for the set $\{x'_1, \dots, x'_n\}$ of primed variables, which represent the values of the x_i at the end of a continuous period, just before occurrence of a discrete change.

Control graph A finite directed multigraph (V, E). The vertices in V are called *control modes*. The edges in E are called *control switches*.

Initial, invariant and flow conditions Three vertex labelling functions *init*, *inv*, and *flow* assign to each control mode $v \in V$ three predicates. Each initial condition $init(v)$ and invariant condition $inv(v)$ is a predicate with free variables from X. Each flow condition $flow(v)$ is a predicate with free variables from $X \cup \dot{X}$.

Jump conditions An edge labelling function *jump* assigns to each control switch $e \in E$ a predicate $jump(e)$ with free variables from $X \cup X'$.

Events A finite set Σ of events, and an edge labelling function $event : E \to \Sigma$ that assigns to each control switch an event.

To illustrate this definition and to show a graphical representation we give one of the standard examples of [2, 11]. This example has no events. In Section 8 we will see an example that contains an event.

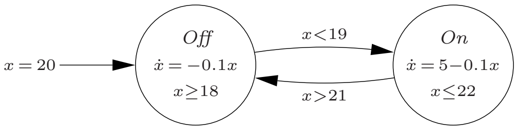

Fig. 1. Thermostat automaton

Example 2.1 (Temperature Control)
The hybrid automaton of Figure 2 models a thermostat. It is adapted from [11]. The variable x represents the temperature. Initially, the temperature is equal to 20 degrees and the heater is off (control mode *Off*). The temperature falls according to the flow condition $\dot{x} = -0.1x$. According to the jump condition $x < 19$, the heater may start as soon the temperature is below 19 degrees. The invariant condition $x \geq 18$ ensures that the heater will go on at least when the temperature is equal to 18 degrees. In control mode *On*, the heater is on and the temperature rises according to the flow condition $\dot{x} = 5 - 0.1x$. If the

temperature reaches the second jump condition, the heater is switched off and the procedure starts again (maybe with another initial value). □

3 Algebra of Hybrid Systems

We aim at an algebraic view of hybrid systems, in particular of hybrid automata. A first approach to this is given in [12]. For this model we use trajectories (cf. e.g. [18]) that reflect the variation of the values of the variables over time. We briefly present the main definitions and ideas and recapitulate the basic algebraic concepts which are in line with our approach.

Let V be a set of *values* and D a set of *durations* (e.g. $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, $\ldots$). We assume a cancellative addition $+$ on D and an element $0 \in D$ such that $(D, +, 0)$ is a commutative monoid and the relation $x \leq y \Leftrightarrow_{df} \exists\, z \,.\, x + z = y$ is a linear order on D. Then 0 is the least element and $+$ is isotone w.r.t. $\leq$. Moreover, 0 is indivisible, i.e., $x + y = 0 \Leftrightarrow x = y = 0$. D may include the special value ∞. If so, ∞ is required to be an annihilator w.r.t. $+$ and hence the greatest element of D (and cancellativity of $+$ is restricted to elements in $D - \{\infty\}$). For $d \in D$ we define the interval $\mathsf{intv}\, d$ of admissible times as

$$\mathsf{intv}\, d =_{df} \begin{cases} [0, d] \text{ if } d \neq \infty \\ [0, d[\text{ otherwise .} \end{cases}$$

A *trajectory* t is a pair (d, g), where $d \in D$ and $g : \mathsf{intv}\, d \to V$. Then d is the *duration* of the trajectory, the image of $\mathsf{intv}\, d$ under g is its *range* $\mathsf{ran}\ (d, g)$. This view models *oblivious* systems in which the evolution of a trajectory is independent of the history before the starting time. Therefore we do not need a notion of fusion-closedness of sets of trajectories.

The set of all trajectories is denoted by TRA. Note that we do not impose any restriction on the function g. If we want to deal with a special kind of functions, e.g. continuous ones, we can restrict g to this property. We only have to ensure that the desired property is closed under sequential composition.

This is defined as follows. The idea is to extend one trajectory (d_1, g_1) at the right end, i.e., at time d_1, with another one (d_2, g_2) to a trajectory $(d_1 + d_2, g)$, if reasonable. Since g needs to be a function, one needs to decide how to handle the time-point d_1.

$$(d_1, g_1) \cdot (d_2, g_2) =_{df} \begin{cases} (d_1 + d_2, g) \text{ if } d_1 \neq \infty \wedge g_1(d_1) = g_2(0) \\ (d_1, g_1) \qquad\quad \text{if } d_1 = \infty \\ \text{undefined} \quad\ \ \text{otherwise} \end{cases}$$

with $g(x) = g_1(x)$ for all $x \in [0, d_1]$ and $g(x + d_1) = g_2(x)$ for all $x \in \mathsf{intv}\, d_2$.

For a zero-length trajectory $(0, g_1)$ we have $(0, g_1) \cdot (d_2, g_2) = (d_2, g_2)$ if $g_1(0) = g_2(0)$. Similarly, $(d_2, g_2) \cdot (0, g_1) = (d_2, g_2)$ if $g_1(0) = g_2(d_2)$ or $d_2 = \infty$. For a value $v \in V$, let $\underline{v} =_{df} (0, g)$ with $g(0) = v$ be the corresponding zero-length trajectory.

A *process* is a set of trajectories, consisting of possible behaviours of a hybrid system. The set of all processes is denoted by PRO. The finite and infinite parts of a process A are defined as

$$\mathsf{inf}\, A \;=_{df}\; \{(d,g) \in A \,|\, d = \infty\} \;, \qquad \mathsf{fin}\, A \;=_{df}\; A - \mathsf{inf}\, A \;.$$

In Section 7 we give a general definition of finite and infinite elements.

Composition is lifted to processes as follows:

$$A \cdot B \;=_{df}\; \mathsf{inf}\, A \cup \{a \cdot b \,|\, a \in \mathsf{fin}\, A, b \in B\} \;.$$

The constraint $g_1(d_1) = g_2(0)$ for composability of trajectories $T_1 = (d_1, g_1)$ and $T_2 = (d_2, g_2)$ is very restrictive in a number of situations. Hence we will introduce a *compatibility relation* $\asymp \subseteq V \times V$, which describes the behaviour at the point of composition. The relation allows 'jumps' at the connection point between T_1 and T_2. This is meaningful, since 'jumps' in the interior of a trajectory ~~(such as the composition of g_1 and g_2)~~ are already allowed by our definition. Note that we do not postulate any condition for $\asymp$. But, in most cases $\asymp$ will be at least reflexive to accommodate the case of equal values $g_1(d_1)$ and $g_2(0)$. If one wants to *enforce* 'jumps' at any composition point, $\asymp$ has to be irreflexive.

To model a more liberal form of composition that takes $\asymp$ into account, we extend a trajectory (d_1, g_1) at the right end, i.e., at time d_1, using the compatibility relation. To this end we express that, up to $\asymp$, we do not care about the exact value $g_1(d_1)$. Therefore we inflate the original trajectory to a process that before time d_1 agrees with the original trajectory, but shows all values admitted by $\asymp$ at time d_1:

$$(d_1, g_1)_{\asymp} \;=\; \{(d_1, g) \,|\, g(x) \;=\; g_1(x),\, x \in [0, d_1[,\, g_1(d_1) \asymp g(d_1)\} \;.$$

The composition of (d_1, g_1) and (d_2, g_2) considering the compatibility relation $\asymp$ is then the composition

$$(d_1, g_1)_{\asymp} \cdot \{(d_2, g_2)\}$$

over PRO. The extension operation is lifted pointwise to processes.

Symmetrically, we can also employ the compatibility relation at the left end or even at both ends of a trajectory.

Example 3.1 We now show how to model hybrid automata using these concepts. As value set we choose $V \;=_{df}\; Q \times \mathbb{R}^n$. The behaviour of the automaton in a given state q in the interval $[0, d]$ is given by a trajectory (d, g) such that $g(t) = (q, f(t))$ for some function $f : [0, d] \to \mathbb{R}^n$ that satisfies the invariant and flow condition of q. This corresponds to Henzinger's relation $(q, f(0)) \xrightarrow{d} (q, f(d))$.

The compatibility relation is given by

$$(p, x) \asymp (q, y) \;\Leftrightarrow\; (p = q \,\wedge\, x = y) \,\vee\, (\exists\, \sigma \in \Sigma : (p, x) \xrightarrow{\sigma} (q, y)) \;,$$

where the first part $(p = q \wedge x = y)$ deals with compositions that do not leave a control mode and the second part models the event belonging to the edge (p, q) (if the edge is present).

Iterated composition of such trajectories then gives the behaviour of the automaton over longer periods of time. □

We apply this encoding for hybrid automata to our running example 2.1.

Example 3.2 We define two sets of trajectories.

$$\begin{aligned} A^{\mathrm{Off}} &=_{df} \ \{(d, (\mathrm{Off}, g(t))) \,|\, d \in \mathbb{R}^{+},\ \dot{g}(t) \ = \ -0.1t\}\,, \\ A^{\mathrm{On}} &=_{df} \ \{(d, (\mathrm{On}, g(t))) \,|\, d \in \mathbb{R}^{+},\ \dot{g}(t) \ = \ 5 - 0.1t\}\,. \end{aligned}$$

Using the compability relation given above, the overall behaviour of the automaton is represented by the algebraic expression

$$(A^{\mathrm{Off}}_{\asymp} \cdot A^{\mathrm{On}}_{\asymp})^{\omega}\ ,$$

where ω is an operator for infinite iteration. The existence of such an operator will be shown in the next section. Note that we do not yet consider the initial value of the temperature. This can easily be done by pre-composing a zero-length trajectory {20}. □

4 Algebraic Background

Now, let us have a closer look at the algebraic structure of the trajectory-based model.

A *left semiring* is a quintuple $(S, +, 0, \cdot, 1)$ where $(S, +, 0)$ is a commutative monoid and $(S, \cdot, 1)$ is a monoid such that $\cdot$ is left-distributive over $+$ and *left-strict*, i.e., $0 \cdot a \ = \ 0$. The left semiring is *idempotent* if $+$ is idempotent and $\cdot$ is right-isotone, i.e., $b \leq c \ \Rightarrow \ a \cdot b \leq a \cdot c$, where the *natural order* $\leq$ on S is given by $a \ \leq \ b \ \Leftrightarrow_{df} \ a + b \ = \ b$. Left-isotony of $\cdot$ follows from its left-distributivity. Moreover, 0 is the $\leq$-least element. A *semiring* is a left semiring in which $\cdot$ is also right-distributive and right-strict.

A left idempotent semiring S is called a *left quantale* if S is a complete lattice under the natural order and $\cdot$ is universally disjunctive in its left argument. Following [7], one might also call a left quantale a *left standard Kleene algebra*. A left quantale is *Boolean* if its underlying lattice is a completely distributive Boolean algebra.

An important left semiring (that is even a semiring and a left quantale) is REL, the algebra of binary relations over a set under relational composition.

Checking all the axioms for the case of processes, we get

Lemma 4.1

1. *The processes form a Boolean left quantale* $\mathrm{PRO} =_{df} (\mathcal{P}(\mathrm{TRA}), \cup, \emptyset, \cdot, \mathrm{I})$.
2. *Additionally, $\cdot$ is positively disjunctive in its right argument.*

As in [16], we can extend an idempotent left semiring by finite and infinite iteration. A *left Kleene algebra* is a structure $(S, {}^*)$ consisting of an idempotent semiring S and an operation * that satisfies the left *unfold* and *induction* axioms

$$1 + a \cdot a^* \leq a^*\ , \qquad b + a \cdot c \leq c \Rightarrow a^* \cdot b \leq c\ .$$

To express infinite iteration we axiomatise an ω-operator over a left Kleene algebra. A *left ω algebra* [6] is a pair $(S, {}^\omega)$ such that S is a left Kleene algebra and ${}^\omega$ satisfies the *unfold* and *coinduction* axioms

$$a^\omega = a \cdot a^\omega\ , \qquad c \leq a \cdot c + b \Rightarrow c \leq a^\omega + a^* \cdot b\,.$$

Lemma nur umformatiert.

Lemma 4.2

1. *Every left quantale can be extended to a left Kleene algebra by defining*
$$a^* =_{df} \mu x\,.\ a \cdot x + 1.$$
2. *If the left quantale is a completely distributive lattice then it can be extended to a left ω algebra by setting $a^\omega =_{df} \nu x\,.\ a \cdot x$. In this case,*
$$\nu x\,.\ a \cdot x + b \;=\; a^\omega + a^* \cdot b.$$

The proof of 2. (see e.g. [5]) uses fixpoint fusion.

Since by Lemma 4.1 PRO forms a left quantale, we also have finite iteration * and infinite iteration ${}^\omega$ with all their laws available.

5 Embeddings of Other Approaches

In [2] Alur et al. show that Hybrid automata are equally expressive as piecewise-smooth hybrid traces. In Example 3.1 we already showed how to embed hybrid automata (and hence piecewise-smooth hybrid traces). However, it is obvious that hybrid traces are more expressive than hybrid automata. In this section we show how to adapt the hybrid traces to the algebra of hybrid systems and hence to the theory of semirings. Furthermore, we discuss the embedding of other models for hybrid systems.

Alur et al. use for their approach any intervals, i.e., intervals can be open, half-open, or closed, as well as bounded or unbounded. Let V' be a set of values (similar to V of Section 3). A *trace* is a function from $\mathbb{R}^+$ to V'.

A set T of traces is *fusion-closed* if for all traces $\tau_1, \tau_2 \in T$ and $t_1, t_2 \in \mathbb{R}^+$, if $\tau_1(t_1) = \tau_2(t_2)$, then $\tau \in T$ for the trace τ with $\tau(t) = \tau_1(t)$ for all $t \leq t_1$ and $\tau(t) = \tau_2(t + t_2 - t_1)$ for all $t > t_1$. Now we show the connection between the set of fusion-closed, piecewise-smooth hybrid traces and PRO. Every trace can be split into a sequence of intervals $I_1, I_2, \ldots$ such that for any i, j with $i \neq j$

$$I_i \cap I_j = \emptyset\ , \qquad \bigcup_i I_i = \{x \mid \exists\, i : x \in I_i\} = \mathbb{R}\ .$$

τ is *piecewise-smooth* iff there is a sequence of intervals such that all $\tau|_{I_i}$ are C^∞-functions. Here, $\tau|_{I_i}$ denotes the restriction of the function τ to I_i, i.e.,

$\tau|_{I_i} : I_i \to V$, $\tau|_{I_i}(x) = \tau(x)$ for all $x \in I_i$. Note that all intervals of the sequence except the first and the last one, can be open, half-open or closed as well as bounded or unbounded. Since the sequence of intervals covers $\mathbb{R}$, the first interval has to be $[0, x[$ or $[0, x]$ and the last one $[x, \infty[$ or $]x, \infty[$. Now we can formulate and prove the desired connection.

Lemma 5.1 *There exists a function α from the fusion set T of piecewise-smooth traces to* PRO, i.e, T can be embedded into PRO.

Proof. We will give a constructive proof by defining α', which connects each trace τ of T to a process. The construction is close to the don't-care approach of Section 3 using the compatibility relation $\asymp$. The main idea is to extend the (half-)open intervals to closed intervals. α is then given by lifting α' pointwise. First, we need a function β which works on an interval of the implicitly given interval sequence. Let I be such an interval. For the construction, we assume $D = \mathbb{R}^+$. We start with the case, where $I = [a, b]$ is closed and bounded, then

$$\beta(\tau|_{[a,b]}) = \{(d, f) \mid d = b - a, f(x) = \tau(x + a), x \in [0, d]\} .$$

In this case, where I is closed, we have a one-to-one relation between a part of the sequence $\tau|_I$ and a trajectory (strictly speaking, and a process with one element). If I is (half-)open, the case is a bit more complicated. We demonstrate this when $I = [a, b[$. The other two cases ($]a, b]$, $]a, b[$) are similar.

$$\beta(\tau|_{[a,b[}) = \{(d, f) \mid d = b - a, f(x) = \tau(x + a), x \in [0, d[, f(d) = v, v \in V\} .$$

We do not have a one-to-one relation anymore. In fact we 'complete' the half-open interval to a closed interval including all possibilities. Therefore, we get a process as result. Two cases are missing, if $I = [a, \infty[$ and if $I =]a, \infty[$. We only show the first one. The second one is similar if we consider again all possibilities for the 'missing point a'.

$$\beta(\tau|_{[a,\infty[}) = \{(d, f) \mid d = \infty, f(x) = \tau(x + a), x \in [0, d[\} .$$

Overall, we have defined a function $\beta : I \to \text{PRO}$. To get $\alpha' : \tau \to \text{PRO}$ we just use the composition of trajectories: $\alpha'(\tau) = \beta(\tau|_{I_1}) \cdot \beta(\tau|_{I_2}) \cdots$ □

Since we add all values of V at those points where the interval is (half-)open, the composition cannot have the empty set as result, and so 'jumps' in the behaviour do not matter and fit well with our model. Obviously, the proof can be generalised to any traces, since we do not make any usage of piecewise-smoothness.

In the case where we restrict PRO to such trajectories where the function f is piecewise-smooth, we can easily formulate a dual lemma.

Lemma 5.2 *There is a function that maps trajectories with piecewise-smooth functions and infinite duration in $D = \mathbb{R}$ to traces.*

Proof. We have to restrict P to its infinite parts since traces have to cover all of $\mathbb{R}^+$. The construction of the function is similar to the one of Lemma 5.1. □

Summarising, we can formulate our main theorem of this section.

Theorem 5.3 *The fusion set S of hybrid traces can be embedded into* PRO. More precisely, hybrid automata, the fusion set of piecewise-smooth traces and PRO restricted to piecewith-smooth trajectories are isomorphic.

Proof. This is a simple consequence of Lemma 5.1 and Lemma 5.2 as well as the fact that Hybrid automata and the fusion set T of piecewise-smooth traces are isomorphic, as shown in [2]. □

So far, we have shown that Hybrid automata (hybrid traces), which are probably the standard tool for describing hybrid systems, can conveniently embedded into our algebra. But in contrast to hybrid automata we can also handle non-smooth trajectories. Other models like timed automata [3] or hybrid I/O automata [14] can be embedded into our algebraic model in a similar way.

6 On the Composition of Hybrid Systems

The product of two finite state machines as well as the parallel composition are well known. Since [11] defines a product and a parallel composition for hybrid automata, we will discuss the algebraic counterpart in this section.

Product Following Example 3.1 and the definition of product of [11] we ~~get~~ define for two hybrid automata (with disjoint sets of states Q_1 and Q_2) ~~we define~~ the following edge labelling functions (for jumps and events)

$$(q_1, q_2) \xrightarrow{a} (q_1', q_2') \Leftrightarrow_{df} \exists\, a_1 \in A_1, a_2 \in A_2 \,:\, q_1 \xrightarrow{a_1} q_1',\ q_2 \xrightarrow{a_2} q_2' \ .$$

To translate this behaviour to our algebraic model we just look at the product semiring.

For two (left) semirings $(A, +_A, 0_A, \cdot_A, 1_A)$, $(B, +_B, 0_B, \cdot_B, 1_B)$ the *product (semiring)* is defined as

$$(A \times B, +_\times, 0_A \times 0_B, \cdot_\times, 1_A \times 1_B) \ ,$$

where $+_\times$ and $\cdot_\times$ are componentwise operators. By standard results from universal algebra the product structure indeed forms a (left) semiring again.

Parallel Composition Parallel composition of hybrid systems can be used for specifying larger systems. An algebraic expression or a hybrid automaton is given for each part of the system communication between the components may occur via shared variables and synchronisation labels. The parallel composition seems to be more complicated. But as we will see, it is easily handled in the algebraic model. It again uses a Cartesian product, like the product semiring (only at a different place). We consider again the definition of Henzinger. Two hybrid automata H_1 and H_2 are assumed to interact via joint events. If event a is an event of both H_1 and H_2, then H_1 and H_2 must synchronise on a-transitions; if

a is an event of H_1 but not of H_2 then during the transition of H_1 the state of H_2 has to be kept constant and vice versa. I.e., if we look at trajectories (d,f) and (e,g), where d and e are elements of the same set of durations, we have

$$(d,f)||(e,g) = \begin{cases} (d, f \bigtriangledown g) & \text{if } d = e \\ (e, \texttt{const}(f) \bigtriangledown g) & \text{if } d = 0 \\ (d, f \bigtriangledown \texttt{const}(g)) & \text{if } e = 0\ , \end{cases}$$

where $(f \bigtriangledown g)(x) = (f(x), g(x))$ and $\texttt{const}(f)(x) = f(0)$ is the constant function. Now we see that in the case of two semirings of processes with the same set of durations the parallel-composed trajectories form again trajectories. I.e., if the first process contains only trajectories (d,f) with functions $f : D \to V$ and for all trajectories (e,g) of the second process we have $g : D \to V'$, then the parallelised process semiring contains a trajectory with functions of type $D \to V \times V'$. Hence the parallel composition of [11] can be modelled with the same algebraic theory just changing the range of trajectories.

7 Zeno Effects

Zeno of Elea's famous paradox of the Achilles and the tortoise is well known. Summarised we get:

> *"In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead."*
>
> *(Aristotle Physics VI:9, 239b15)*

However, when describing hybrid systems as well as any natural behaviour, Zeno effects are excluded in general. In the theory of hybrid systems usually the authors do not consider them, even if they appear in their theoretical model. In papers concerning such systems it is also not mentioned how to avoid those effects. In this section we present a possible way of handling Zeno effects in PRO and characterise the zeno and zenofree parts of hybrid systems.

Roughly spoken, a Zeno effect occurs if an infinite iteration does not take infinite time (duration). In Section 3 we already have defined finite and infinite parts of trajectories and processes. In general left semirings an element $a \in S$ is called *infinite* if it is a left zero, i.e., $a \cdot b = a$ for all $b \in S$. An equivalent characterisation is $a \cdot 0 = a$. By this property, $a \cdot 0$ may be considered as the infinite part of a. We assume that there exists a largest infinite element N, i.e.,

$$a \leq \mathsf{N} \Leftrightarrow_{df} a \cdot 0 = a\ .$$

Dually, we call an element a *finite* if its infinite part is trivial, i.e., if $a \cdot 0 = 0$. We also assume that there is a largest finite element F, i.e.,

$$a \leq \mathsf{F} \Leftrightarrow_{df} a \cdot 0 = 0\ .$$

In Boolean quantales N and F always exist and satisfy

$$\mathsf{N} = \top \cdot 0 \,, \qquad \mathsf{F} = \overline{\mathsf{N}} \,.$$

Moreover, every element can be split into its finite and infinite parts: $a = \mathsf{fin}\, a + \mathsf{inf}\, a$, where $\mathsf{fin}\, a = a \sqcap \mathsf{F}$ and $\mathsf{inf}\, a = a \sqcap \mathsf{N}$. For proofs and further properties see [16]. The above definitions of fin and inf fit in well with the ones for processes given earlier. As a general decomposition law for finite and infinite elements we have

Lemma 7.1 *If $a, b \in \mathsf{F}$ and $c, d \in N$ we have*

$$a + c \leq b + d \Leftrightarrow a \leq b \wedge c \leq d \ .$$

Proof. By lattice algebra and $\mathsf{N} \sqcap \mathsf{F} = 0$, we get

$$a + c \leq b + d \Leftrightarrow a \leq (b + d) \wedge c \leq (b + d) \Leftrightarrow a \leq b \wedge c \leq d$$

□

Now we are able to characterise parts where Zeno effects occur and parts where they do not. In a left ω algebra S

1. $a \in S$ is *convergent* if $a^\omega \leq \mathsf{F}$. If a is convergent, we call a^ω *zeno*.
2. $a \in S$ is *divergent* if $a^\omega \leq \mathsf{N}$. If a is divergent, a^ω is called *zenofree*.

In PRO, *all* trajectories in a zeno process show zeno behaviour. We denote the set of all convergent elements by $C(S)$ and the set of all divergent elements by $D(S)$. However, every element a^ω can be split into its zeno and zenofree parts.

$$a^\omega = (a^\omega \sqcap \mathsf{F}) + (a^\omega \sqcap \mathsf{N}) \qquad (1)$$

Lemma 7.2 *Let S be a left ω algebra. Then*

1. $a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^\omega$,
2. $a^\omega \sqcap \mathsf{N} = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + \mathsf{inf}\, ((\mathsf{fin}\, a)^\omega)$,
3. $a^\omega \sqcap \mathsf{F} = (\mathsf{fin}\, a)^\omega \sqcap \mathsf{F} \leq (\mathsf{fin}\, a)^\omega$.

Proof.

1. Same as Lemma 6.8.(d) of [16].
2. By (1), distributivity of inf and $\mathsf{inf}\,\mathsf{inf}\, x = \mathsf{inf}\, x$,

umformatiert

$$\begin{aligned} & a^\omega \sqcap \mathsf{N} \\ = \ & \mathsf{inf}\, ((\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^\omega) \\ = \ & \mathsf{inf}\, ((\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a) + \mathsf{inf}\, ((\mathsf{fin}\, a)^\omega) \\ = \ & (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + \mathsf{inf}\, ((\mathsf{fin}\, a)^\omega). \end{aligned}$$

umformatiert

3. By (1), distributivity, $\mathsf{N} \sqcap \mathsf{F} = 0$ and isotony,

$$
\begin{aligned}
& a^{\omega} \sqcap \mathsf{F} \\
= \; & ((\mathsf{fin}\, a)^{*} \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^{\omega}) \sqcap \mathsf{F} \\
= \; & ((\mathsf{fin}\, a)^{*} \cdot \mathsf{inf}\, a) \sqcap \mathsf{F} + (\mathsf{fin}\, a)^{\omega} \sqcap \mathsf{F} \\
= \; & (\mathsf{fin}\, a)^{\omega} \sqcap \mathsf{F} \\
\leq \; & (\mathsf{fin}\, a)^{\omega} .
\end{aligned}
$$

□

3. fits well with our intuition, since in PRO it means that Zeno effects can only occur when all trajectories in a process a are finite. 2. says that the zenofree part of a process contains either at least one infinite element or all finite parts have long enough durations that their iteration obtains infinite duration.

Example 7.3 Returning to our running example of the thermostat, we can now describe all nonzeno behaviours as $(\underset{\frown}{A}^{\mathrm{Off}} \cdot \underset{\frown}{A}^{\mathrm{On}})^{\omega} \sqcap \mathsf{N}$. □

To give another example for PRO in the special case where the set of durations $D = \mathbb{R}$, we first define a scaling function. We denote the set of all trajectories with $D = \mathbb{R}$ by $\mathrm{TRA}_{\mathbb{R}}$. Similarly, we denote by $\mathrm{PRO}_{\mathbb{R}}$ the set of processes where $D = \mathbb{R}$.

$$
\begin{aligned}
sc_n : \mathrm{TRA}_{\mathbb{R}} &\to \mathrm{TRA}_{\mathbb{R}} \\
(d, f) &\mapsto (\tfrac{d}{n}, g) \ ,
\end{aligned}
$$

where $n \in \mathbb{N}$ and $g(x) = f(x \cdot n)$. Then, given a trajectory $T_1 = (d_1, f_1)$ with $f_1(0) = f_1(d)$, we define a process

$$
P = \{ sc_n(T_1) \mid n \in \mathbb{N} \} \ .
$$

It is easy to see, that P^{ω} contains an infinite number of zenofree parts as well as an infinite number of zeno parts. $P^{\omega} \sqcap \mathsf{F}$ (the Zeno-parts of P^{ω} is closely related to the paradox of Achilles and the tortoise, because P contains trajectories with indefinitely short duration).

8 Case Study

To round off the paper, we give a longer case study. It concerns a railroad gate control and was introduced in [11]. The hybrid automaton of Figure 8 models a train on a circular track with a gate. The variable x represents the distance of the train from the gate. Initially, the speed of the train is between 40 and 50 metres per second. At the distance of 1000 metres from the gate, the train issues an *approach* event and may slow down to 30 metres per second. At the distance of 100 metres behind the gate, the train issues an *exit* event. The circular track is between 2000 and 5000 metres long. Contrary to our previous examples, this one treats a hybrid system with events.

For our algebraic expression we first define some special processes. To simplify matters and in contrast to Example 3.1 we skip the control modes in the

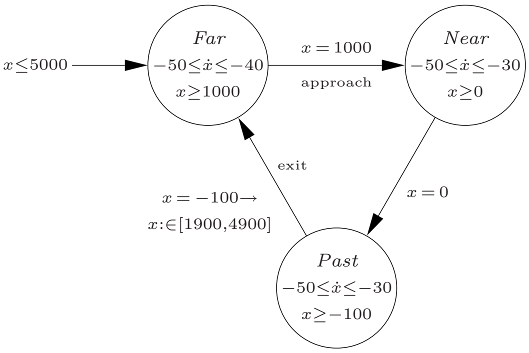

Fig. 2. Train automaton

algebraic expression. Since all control modes have the same structure we define the following abbreviations:

$$T^{[a,b]} =_{df} \{(d,g) \mid d \in \mathbb{R}^+,\ a \le \dot{x} \le b\} \ ,$$
$$P_a =_{df} \{\underline{a}\} \ .$$

Using the encoding from hybrid automata to PRO (Section 3) and $P_{\le 5000} =_{df} \{\underline{a} \mid a \le 5000\}$, we get for the train

$$\mathrm{TR} =_{df} P_{\le 5000} \cdot \Big((T^{[-50,-40]} \cdot P_{1000} \cdot T^{[-50,-30]} \cdot P_0 \cdot T^{[-50,-30]} \cdot P_{-100})_{\asymp} \Big)^{\omega} ,$$

where $\asymp =_{df} \{(-100, y) \mid y \in [1900, 4900]\}$ is a special compatibility relation employed at the right end of processes as described in Section 3. It is only needed at the point where we want to enforce a 'jump' in the function describing the distance between the train and the gate. The other multiplications require the identity relation as compatibility relation, since we want to avoid 'jumps'. That is why we do not need explicit compatibility relations for the other products.

As the second component of the railroad gate control we have a gate automaton (Figure 8).

The variable y of the gate automaton represents the position of the gate in degrees. Initially, the gate is open ($y = 90$). When a *lower* event is received, the gate starts closing at the rate of 9 degrees per second and when a *raise* event happens, the gate starts opening at the same rate. The algebraic expression for the gate automaton is

$$\mathrm{GA} =_{df} O \cdot \Big((M_l \cdot M_r)^* \cdot (C + O) \Big)^{\omega} \ ,$$

where

$O =_{df} \{(d, \mathtt{const}(90)) \mid d \in \mathbb{R}^+\}$ models control mode *Opened*,
$C =_{df} \{(d, \mathtt{const}(0)) \mid d \in \mathbb{R}^+\}$ models control mode *Closed*,
$M_l =_{df} \{(d,g) \mid \dot{g} = -9,\ d \in \mathbb{R}^+\}$ models control mode *Down*,
$M_r =_{df} \{(d,g) \mid \dot{g} = 9,\ d \in \mathbb{R}^+\}$ models control mode *Up*

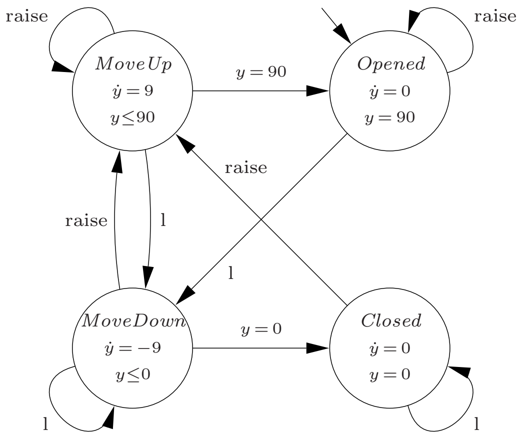

Fig. 3. Gate automaton

and `const` is again the constant function. $M_l \cdot M_r$ is iterated because the gate can start opening even if the gate is not totally closed ($y = 0$) and it can start closing even if the gate is not absolutely opened ($y = 90$).

The simplest way to combine both expressions is

$$\mathrm{TR} \,||\, \mathrm{GA}$$

where || is the pointwise lifted parallel composition of Section 6. But this algebraic expression contains all combinations of the train trajectories and the gate trajectories, e.g., the gate can be opened when the train passes.

To combine these two automata and to guarantee safety, one can use a third automaton (Controller automaton) as done in [11]. This controller has a reaction delay of some seconds. To simplify matters, we assume a reaction time of 0 seconds. (Another time of delay is also possible, but the algebraic expressions become more complicated, although the structure would be the same.) When an *approach* event is received, the controller issues a *lower* event and when an *exit* event is received, the controller starts an *raise* event. In sum we have:

$$\begin{aligned}\mathrm{TG} \;=\; &\Big(O||(P_{\leq 5000} \cdot T^{[-50,-40]} \cdot P_{1000})\Big) \cdot \Bigg(\Big((M_l \cdot C)||(T^{[-50,-30]} \cdot P_0)\Big)\\ &\cdot\; \Big(C||(T^{[-50,-30]} \cdot P_{-100})_{\asymp}\Big) \cdot \Big((M_r \cdot O)||(T^{[-50,-40]} \cdot P_{1000})\Big)\Bigg)^{\omega}\end{aligned} \tag{2}$$

Let us have a look at the single components. The first part $(O||(P_{\leq 5000} \cdot T^{[-50,-40]} \cdot P_{1000}))$ models the initial behaviour; the gate has to be open, the train starts somewhere before the gate (not farther than 5000 metres), and moves until it reaches the point $x = 1000$. Each of the components in the infinite iteration loop has as right operand of the parallel composition one control mode of the train automaton together with the attached event and as left operand the corresponding behaviour of the gate. Note that the nested iteration of GA has been removed, because that behaviour cannot occur.

Aspects of Safety The algebra of processes not only compacts the description by a parallelised hybrid automaton (which was not given by Henzinger), but also contains many aspects of safety. E.g., the expression $M_l \cdot C$ itself guarantees that the gate is closed at the time when the train passes the gate. This guarantee is not given in the original paper. Furthermore, it is easy to see that if the initial distance between the gate and the train is smaller than 1000, we have for the first factor of (2)

$$(P_{<1000} \cdot T^{[-50,-40]} \cdot P_{1000}) \;=\; 0 \;.$$

Thus we know that such an initial distance is not safe, since it is not possible that the gate gets closed in time. This problem is not discussed in [11]. In general, if an algebraic expression or a part of it at a strict position is equal to zero, the corresponding system is not safe. Another aspect of safety is the Zeno problem. In our example, Zeno effects can occur in the hybrid automata as well as in our algebraic expressions. But those effects can be excluded by taking

$$\mathrm{TG} \sqcap \mathsf{N} \;,$$

as discussed in Section 7. Sometimes it is desirable and necessary to introduce range assertions. For instance, we may, besides the normal conditions of operation, want to guarantee that no train is faster than 40 metres per seconds (e.g. if there is construction work on the track). Then we have to modify our expression (2). In [12] we have introduced range assertions and the algebraic expression becomes

$$\mathrm{TG} \sqcap \Box T^{[0,-40]} \;.$$

With this, we have a characterisation of the modified system and can now check safety, etc.

9 Outlook

It will be interesting to apply the approach in further case studies. On the more theoretical side, an algebraic treatment of time abstraction as well as further analysis of safety via range assertions and of liveness issues is necessary. The structures of Kleene and ω algebras should allow a convenient algebraic treatment of reachability questions [9]. Finally, the algebraic semantics for CTL* given in [17] prepares the connection to various logics for hybrid systems [8].

Acknowledgement

We are grateful to Kim Solin for helpful discussions and remarks.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine: The Algorithmic Analysis of Hybrid Systems. Theoretical Computer Science, 138:3–34, 1995.
2. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In Workshop on Theory of Hybrid Systems, Lyngby, Denmark, October 1992.
3. R. Alur, D.L. Dill. A Theory of Timed Automata. Theoretical Computer Science 126(2):183–236 (1994).
4. R. Alur, and D.L. Dill: Automata for Modeling Real-time Systems. In M.S. Paterson (ed.) ICALP 90: Automata, Languages, and Programming, Lecture Notes in Computer Science 630:340–354, Springer, 1992.
5. R. C. Backhouse *et al.* Fixed point calculus. *Information Processing Letters*, 53:131–136, 1995.
6. E. Cohen: Separation and Reduction. In R. Backhouse, J. N. Oliveira (eds.): Mathematics of Program Construction. LNCS 1837. Springer 2000, 45–59.
7. J. H. Conway: Regular Algebra and Finite Machines. Chapman & Hall, 1971.
8. J. Davoren, A. Nerode: Logics for Hybrid Systems. Proc. IEEE 88(7):985–1010 (2000).
9. J. Desharnais, B. Möller, G. Struth: Kleene Algebra with Domain. ACM Trans. Computational Logic (to appear 2006). Preliminary version: Universität Augsburg, Institut für Informatik, Report No. 2003-07, June 2003.
10. T.A. Henzinger: Hybrid Automata with finite Bisimulations. In Z. Fülöp and F. Gécseg (eds.) ICALP 95: Automata, Languages and Programming, 324–335. Springer, 1995.
11. T.A. Henzinger: The Theory of Hybrid Automata: Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1996, pp. 278-292. Extended version in M.K. Inan, R.P. Kurshan (eds.): Verification of Digital and Hybrid Systems. NATO ASI Series F: Computer and Systems Sciences, Vol. 170, Springer-Verlag, 2000, pp. 265–292.
12. P. Höfner, B. Möller: Towards an Algebra of Hybrid Systems. In W. MacCaull, M. Winter and I. Duentsch (eds.): Relational Methods in Computer Science. LNCS 3929 (in press).
13. D. Kozen: Kleene Algebra with Tests. ACM Trans. Programming Languages and Systems 19, 427–443 (1997).
14. N. Lynch, R. Segala, F. Vaandraager: Hybrid I/O Automata. Information and Computation, 185(1):105–157 (2003).
15. B. Möller: Towards pointer algebra. Science of Computer Programming 21, 57–90 (1993)
16. B. Möller: Kleene getting lazy. Science of Computer Programming, Special issue on MPC 2004 (to appear). Previous version: B. Möller: Lazy Kleene algebra. In D. Kozen (ed.): Mathematics of program construction. LNCS 3125. Springer 2004, 252–273.
17. B. Möller, P.Höfner, G. Struth: Quantales and Temporal Logics. (submitted to AMAST 2006).
18. M. Sintzoff: Iterative Synthesis of Control Guards Ensuring Invariance and Inevitability in Discrete-Decision Games. In O. Owe, S. Krogdahl, T. Lyche (eds.): From Object-Orientation to Formal Methods — Essays in Memory of Ole-Johan Dahl. LNCS 2635. Springer 2004, 272–301.