Automated Reasoning for Hybrid Systems — Two Case Studies —

Peter Höfner

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

Abstract. At an abstract level hybrid systems are related to variants of Kleene algebra. Since it has recently been showed that Kleene algebras and their variants, like omega algebras, provide a reasonable base for automated reasoning, the aim of the present paper is to show that automated algebraic reasoning for hybrid system is feasible. We mainly focus on applications. In particular, we present case studies and proof experiments to show how concrete properties of hybrid systems, like safety and liveness, can be algebraically characterised and how off-the-shelf automated theorem provers can be used to verify them.

1 Introduction

Hybrid systems are heterogeneous systems characterised by the interaction of discrete and continuous dynamics. Because of their widespread applications there was a rapid growth of interest in such systems during the last decade. They are an effective tool for modelling, design and analysis of a large number of technical systems such as traffic control [9, 13] and automated manufacturing [8].

The most elementary and classical hybrid system usually consists of a controller and a controlled subsystem. Usually the controller represents discrete behaviour and the environment is described by the continuous behaviour. In general, the behaviour of the controller depends on the state and the behaviour of the controlled system and cannot be considered in isolation. More complicated hybrid systems usually arise by composing smaller systems.

Nearly from the beginning of their formal introduction in computer science it was proposed to model hybrid systems as hybrid automata [11, 14]. Hybrid automata are based on timed automata [4] and have, next to nodes and edges, differential equations and variables. These additional features reflect the behaviour of the environment in each node. The study of hybrid systems in computer science is still largely focused on hybrid automata. There are only few other approaches to hybrid systems, e.g., [5]. In [17] an approach that combines variants of Kleene algebra with the concept of hybrid systems is given.

Over the last decades Kleene algebras have proved to be fundamental first-order structures in computer science with widespread applications ranging from program analysis and semantics to combinatorial optimisation and concurrency control. They offer operators for modelling actions, programs or state transitions

under non-deterministic choice, sequential composition and finite iteration. They allow the formalisation and specification of safety and liveness properties for hybrid systems at an abstract level.

Recently, it has been showed that Kleene algebra and their variants provide a reasonable base for automated deduction [20, 21]. Therefore the techniques developed there should be reuseable for automated reasoning about hybrid systems in an algebraic setting. The aim of the paper is to show that the algebraic approach indeed yields proofs for safety and liveness, and to discover if automated algebraic reasoning for hybrid system is feasible.

This paper mainly focuses on applications. In particular, we present case studies to show how properties can be algebraically specified and how off-the-shelf automated theorem provers can be used to verify them. The first case study is a technical system where a selected route is automatically compared with the specification. If the specification is not satisfied another route has to be chosen. This case study is developed step by step to briefly define and illustrate the underlying theory. The second case study is more complex and describes an assembly line scheduler.

2 Case Study I — Checking a Specification

To illustrate the basic definitions and concepts used in the remainder, we take the following example.

Example 2.1. We assume a security service that has to control three locations (bank, disco and university). The corresponding hybrid automaton (Figure 1) models all possible routes the security service can use when starting at university.

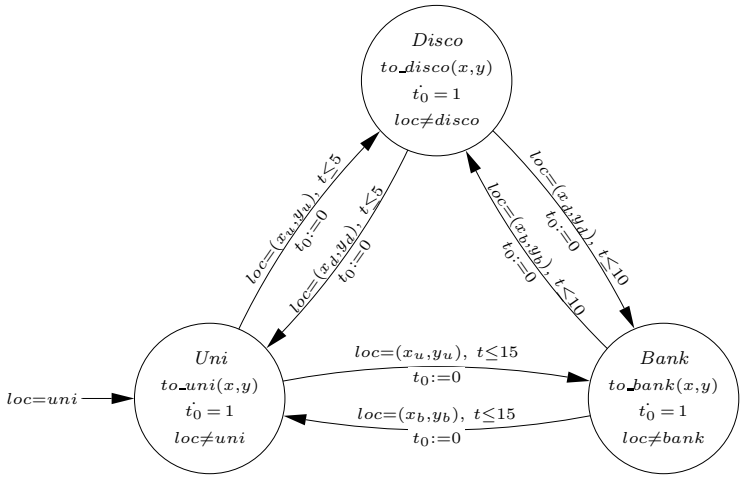

Fig. 1. A simple system for route planning

We briefly explain the meaning of the automaton. Details about hybrid automata in general can be found in [3, 14]. Employees of the security service can be in three different states: either they travel to university (described by state *Uni*) or they are on their way to the *Bank* or they are going to control the *Disco*. The functions *to_uni* and t_0 describe the continuous behaviour of the hybrid system when moving to university (continuous behaviour in node *Uni*): $to\_uni(t)$ determines the path to university starting from the actual time and the current position given by the two coordinates x_c and y_c. Usually this function is specified by an initial value problem combined with (ordinary) differential equations. To measure time between two locations a clock (the function t_0) is introduced. Special locations for university, bank and disco are denoted by (x_u, y_u), (x_b, y_b) and (x_d, y_d), respectively. As long as the university is not reached (denoted by the invariant condition $loc \neq (x_u, y_u)$), the security service continues to move towards the university. If the university is reached ($loc = (x_u, y_u)$), the employees have the (non-deterministic) choice to go either to the bank or to the disco. This state-changing situation represents the discrete part of the hybrid system. Typically, this decision is made by a controller. The other states and functions are built in a similar way. The time conditions like $t_0 \leq 5$, given at the edges, guarantee that the way between uni and disco takes at most 5 minutes; the way between disco and bank needs less than 10 minutes and the one between bank and uni less than 15 minutes. After changing the state the clock is reset to 0. Now we assume that the security service has to check every place at least every half an hour. Due to the small size it is easy to see that e.g. the circle starting at university and then via bank to disco and back to university satisfies the required safety condition, if it is repeated again and again.

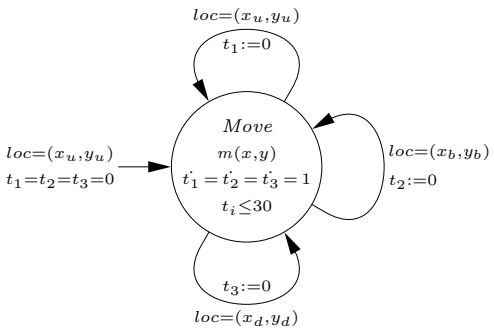

Fig. 2. An alternative route planning automaton

To encode the time constraint that every location has to be visited every 30 minutes, one can use the hybrid automaton of Figure 2. The main idea is to have one state in which the service is moving. The action of moving is denoted by $m(t)$, e.g., $\dot{m}(t) = v$ if the movement is done with a constant velocity v, and

the current position as initial condition $m(0) = (x_c, y_c)$.[1] Unfortunately, in this automaton the time constraints between the 3 locations cannot be encoded. To model the specification within hybrid automata one has to combine both automata presented. This yields an automaton with 4 clocks. To check the given safety property using one of these hybrid automata is not an easy and straightforward exercise. □

But how can it be (automatically) checked that a given run of a hybrid automaton satisfies a given specification, in general? The example above shows that it is not easy to determine an answer. In the remainder we show that in an algebraic setting the above safety property yields a surprisingly simple inequality that can easily be proved.

3 An Algebra for Hybrid Systems

We aim at the use of first-order automated reasoning for hybrid systems. For that an algebraic (first-oder) view of hybrid systems is needed. We follow the lines of [17]. The algebra for hybrid systems uses trajectories that reflect the variation of the values of the variables over time.

Let V be a set of *values* and D a set of *durations* (e.g. $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, ...). We assume 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. If $+$ is cancellative, 0 is the least element and $+$ is isotone w.r.t. $\leq$. Moreover, 0 is indivisible. 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.

The idea of composing two trajectories $T_1 = (d_1, g_1)$ and $T_2 = (d_2, g_2)$ is to extend T_1 at the right end, i.e., at time d_1, with T_2 to a trajectory $(d_1 + d_2, g)$, if reasonable. Figure 3 illustrates the concept. Since g needs to be a function, one needs to decide how to handle the time-point d_1. The definition of sequential composition is given by

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

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

[1] This example is not realistic, but will illustrate the crucial ideas.

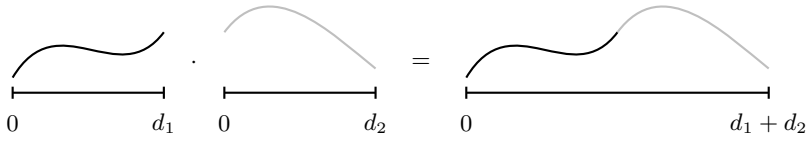

Fig. 3. Composition of two finite trajectories

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

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

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 a *compatibility relation*, which describes the behaviour at the point of composition is introduced in [18]. That relation allows 'jumps' at the connection point between T_1 and T_2. In the remainder we do not need this concept; we mention it only for completeness.

Example 3.1. We want to give an algebraic expression for the automaton of Figure 1. For that we define $V = \mathbb{R}^2$, where an element determines the current position (x, y). A possible way is to define a process for each node for a hybrid automaton. For example

$$u =_{df} \{(d, g) \,|\, g(t) = to\_uni(t)\} \ .$$

The clock t_0 can be dropped since we have the duration d available and therefore the clock is redundant. Similar to u one can define processes for the nodes *Disco* and *Bank*. But, since the functions *to_uni*, *to_bank* and *to_disco* are not specified we abstract to a general "move action". In particular, we define

$$a_n =_{df} \{(d, g) \,|\, d \leq n, g = m(t)\} \ .$$

It describes all routes that the security service can use and take at most n minutes. To check if the security service is at a certain point, we use zero-length

trajectories:

$$\begin{aligned} at_u &=_{df} \underline{(x_u, y_u)} = \{(0,g) \mid g(0) = (x_u, y_u)\} \ , \\ at_b &=_{df} \underline{(x_b, y_b)} = \{(0,g) \mid g(0) = (x_b, y_b)\} \ , \\ at_d &=_{df} \underline{(x_d, y_d)} = \{(0,g) \mid g(0) = (x_d, y_d)\} \ . \end{aligned}$$

These sets describe the situation when the security service is exactly at the locations university (at_u), bank (at_b) and disco (at_d). In the remainder we use such elements to model tests and assertions. Now, we are able to describe the hybrid automaton of Figure 1 in an algebraic setting. The main construct is of the form $at_u \cdot a_5 \cdot at_d$ which describes all possible ways from university to the disco taking at most 5 minutes. The whole automaton can be described by

$$\begin{aligned} at_u \cdot \big(& at_u \cdot a_5 \cdot at_d \cup at_d \cdot a_5 \cdot at_u \cup \\ & at_d \cdot a_{10} \cdot at_b \cup at_b \cdot a_{10} \cdot at_d \cup \\ & at_b \cdot a_{15} \cdot at_u \cup at_u \cdot a_{15} \cdot at_b \big)^{\omega} \ , \end{aligned} \tag{1}$$

where $^{\omega}$ models infinite iteration and therefore an infinite loop. The exact definition of this iteration operator is given in the next section. □

4 Algebraic Background

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. The latter axiom (right-strictness) is dropped to model infinite behaviour. Differences between left semirings and standard semirings are listed e.g. in [25].

An idempotent left 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 Boolean algebra. In these cases the meet operator $\sqcap$ is available, too.

By simple calculations we get the two splitting laws

$$a + b \leq c \ \Leftrightarrow \ a \leq c \wedge b \leq c \quad \text{and} \quad a \leq b \sqcap c \ \Leftrightarrow \ a \leq b \wedge a \leq c \ . \tag{2}$$

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* PRO $=_{df}$ $(\mathcal{P}(\mathrm{TRA}), \cup, \emptyset, \cdot, I)$ *with* $I =_{df} \{(0,g) \mid (0,g) \in \mathrm{TRA}\}$.
2. *Additionally,* $\cdot$ *is positively disjunctive in its right argument.*

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

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

Informally, the *-operator characterises finite iteration. To express infinite iteration we axiomatise an ω-operator over a left Kleene algebra. A *left omega algebra* [25] is a pair $(S, {}^\omega)$ such that S is a left Kleene algebra and ${}^\omega$ satisfies the *unfold* and *coinduction* axioms

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

As a consequence of fixpoint fusion (e.g. [10]) we have the following lemma.

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 omega 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 following lemma lists a couple of properties for left omega algebras which are needed afterwards. Some of them can be found in [25].

Lemma 4.3. *Assume a left omega algebra* S *and* $a, b \in S$.

1. $a \cdot (b \cdot a)^\omega \leq (a \cdot b)^\omega$.
2. $a^\omega \cdot b \leq a^\omega$.
3. $(a \cdot b)^\omega \leq (a + b)^\omega$.
4. $\forall i \in \mathbb{N}, i > 0 : (a^i)^\omega \leq (a^+)^\omega = a^\omega$, *where* $a^+ =_{df} a^* \cdot a$.

All proofs (except the first inequality of Lemma 4.3.4) have been done by the automated theorem prover Prover9 (cf. Section 5) and can be found at a website [19]. The property $(a^i)^\omega \leq (a^+)^\omega$ cannot be encoded with Prover 9 because it is universally quantified. But it is a simple consequence of $a^i \leq a^+$ and isotony.

In Example 3.1, we have already used sets of zero-length trajectories to model assertions. The algebraic counterparts of such elements are tests in (left) semirings (e.g. [12, 23]). One defines a *test* in an idempotent left semiring (quantale) to be an element $p \leq 1$ that has a complement q relative to 1, i.e., $p + q = 1$ and $p \cdot q = 0 = q \cdot p$. The set of all tests of S is denoted by $\mathsf{test}(S)$. It is not hard to show that $\mathsf{test}(S)$ is closed under $+$ and $\cdot$ and has 0 and 1 as its least and greatest elements. Moreover, the complement $\neg p$ of a test p is uniquely determined by the definition and $\mathsf{test}(S)$ forms a Boolean algebra. In particular,

tests are idempotent w.r.t. multiplication and we have the *shunting rule* for a test p:

$$p \cdot (p \cdot a)^\omega = (p \cdot a)^\omega \qquad \text{and} \qquad (p \cdot a)^\omega = (p \cdot a \cdot p)^\omega \ . \tag{3}$$

Again, the proofs can be done fully automatically using Prover9 (see Section 5). Due to Lemma 4.1 and 4.2, we also have finite iteration * and infinite iteration $^\omega$ with all their laws available in PRO. Moreover we can now formulate the specification of Example 2.1.

Example 4.4. Remember that we want to check that, for a given trajectory of the hybrid automaton, the security service checks every location at least every 30 minutes. Let us consider the following (infinite) route for the security service.

$$\tau =_{df} (at_u \cdot a_5 \cdot at_d \cdot a_{10} \cdot at_b \cdot a_{15})^\omega \ .$$

It is straightforward to show that τ is a trace of the hybrid automaton's encoding of Figure 1 (cf. Equation (1)). To formulate the safety criterion for visiting each place at least once in 30 minutes, we have to check $\tau \leq (a_{30} \cdot at_u)^\omega \sqcap (a_{30} \cdot at_d)^\omega \sqcap (a_{30} \cdot at_b)^\omega$. By (2) it is equivalent that

$$\tau \leq (a_{30} \cdot at_u)^\omega \ , \qquad \tau \leq (a_{30} \cdot at_d)^\omega \qquad \text{and} \qquad \tau \leq (a_{30} \cdot at_b)^\omega \ . \tag{4}$$

We only show that the second equation can easily checked by hand; the other inequalities can be showed similarly. In the next section we present a possibility to automate such calculations. By isotony and definition of a_n we get

$$at_u \cdot a_5 \cdot at_d \cdot a_{10} \cdot at_b \cdot a_{15} \ \leq \ a_5 \cdot at_d \cdot a_{10} \cdot a_{15} \ \leq \ a_5 \cdot at_d \cdot a_{25} \ .$$

Therefore it is sufficient to show that $(a_5 \cdot at_d \cdot a_{25})^\omega \leq (a_{30} \cdot at_d)^\omega$. By unfold, Lemma 4.3.1, isotony, and unfold:

$$\begin{array}{ll} & (a_5 \cdot at_d \cdot a_{25})^\omega \\ = & (a_5 \cdot at_d \cdot a_{25}) \cdot (a_5 \cdot at_d \cdot a_{25})^\omega \\ \leq & a_5 \cdot at_d \cdot (a_{25} \cdot a_5 \cdot at_d)^\omega \\ \leq & a_{30} \cdot at_d \cdot (a_{30} \cdot at_d)^\omega \\ = & (a_{30} \cdot at_d)^\omega \ . \end{array}$$

This calculation shows that the chosen trace satisfies the safety criterion. In the algebraic setting it is a simple and short calculation, whereas in the setting of hybrid automata it was not possible in a straightforward way. □

5 Automated Deduction

Having the algebraic characterisation of hybrid systems we can now use off-the-shelf theorem provers to verify or falsify properties. We use McCune's Prover9 tool [24] for proving theorems, but any first-order theorem prover should lead to similar results. Kleene algebras have already been integrated into higher-order

theorem provers [1, 22, 29] and their applicability as a formal method has successfully been demonstrated in that setting. Nevertheless higher-order theorem provers need a huge amount of user interaction, whereas first-order provers need no interaction at all.

Prover9 is a saturation-based theorem prover for first-order equational logic. It implements an ordered resolution and paramodulation calculus and, by its treatment of equality by rewriting rules and Knuth-Bendix completion, it is particularly suitable for reasoning within variants of semirings. Prover9 is complemented by the counterexample generator Mace4, which is very useful in practice.

Prover9 and Mace4 accept input in a syntax for first-order equational logic. The input file consists essentially of a list of hypotheses (the set of support), e.g., the axioms of left omega algebra, and a goal to be proved. Prover9 negates the goal, transforms the hypotheses and the goal into clausal normal form and tries to produce a refutation. Mace4, in contrast, enumerates finite models of the hypothesis and checks whether they are consistent with the goal.

The inference process of saturation-based theorem proving is discussed in detail in the Handbook on Automated Reasoning [28]. Roughly, it consists of two interleaved modes.

- The deduction mode closes a given clause set under the inference rules of resolution, factoring and paramodulation. The paramodulation rule implements equational reasoning by replacing equals by equals.
- The simplification mode discards clauses from the working set if they are redundant with respect other clauses.

In this process, simplification rules are applied eagerly and deduction rules lazily to keep the working set small. The process stops when the closure has been computed or when the empty clause `$F` — which denotes inconsistency — has been produced. Obviously the termination cannot be guaranteed. In the second case, Prover9 reconstructs and displays a proof.

Saturation-based theorem proving implements a semi-decision procedure for first-order equational logic. Whenever the goal is entailed by the hypotheses, the empty clause can be produced in finitely many steps. Otherwise, if the goal is not entailed, a counterexample exists, though not necessarily a finite one.

Since we are interested in robust results that can quickly be obtained by non-experts, we use the prover more or less as a black box and rely on the default strategies provided by Prover9. This makes our experiments more relevant to formal software development contexts.

First we have to encode left omega algebra for Prover9. This is done in a straightforward way; the code can be found in Appendix B. The goal to be proved is also encoded in the same way, i.e., to prove Lemma 4.3.1 one has to add the lines

```
formulas(goals).
  x;(y;x)^ + (x;y)^ = (x;y)^.
end_of_list.
```

whereas ; denotes multiplication, + denotes addition and ^ denotes the omega operator. The proof takes around $100s$ and is fully automatically.[2] To speed up the proofs one can use hypotheses learning techniques [21, 30]. This reduces the set of axioms and yields a proof in less than a second for the above equation. Such techniques seem very promising since the simple first-order equational calculus of idempotent left semirings (left Kleene algebras/left omega algebra) yields particularly short proofs. Let us now return to our running example.

Example 5.1. We will now check the Equations (4) fully automatically. Since standard theorem provers are not able to handle simple arithmetics, we have to encode the relationship between different elements like $a_5 \cdot a_{15} \leq a_{30}$ by hand. But, obviously it is not difficult to produce such formulas with an automated preprocessor. The three equations are encoded by

```
formulas(goals).
  all all u all d all b(
      u;u=u & u+1=1 & d;d=d & d+1=1 & b;b=b & b+1=1     %preconditions
    ->
      (u;a5;d;a10;b;a15)^ + (a30;u)^ = (a30;u)^ &
      (u;a5;d;a10;b;a15)^ + (a30;d)^ = (a30;d)^ &       %the 3 equations
      (u;a5;d;a10;b;a15)^ + (a30;b)^ = (a30;b)^).
end_of_list.
```

In the code u corresponds to at_u, d to at_d, a5 to a_5, etc. Since at_u, at_d and at_b are zero-length processes and therefore tests, we have to specify tests for Prover9. This can be done in a general setting (see [19]) or by specifying properties of tests. The preconditions reflect the two main properties for tests, namely that tests are idempotent and subidentities. Prover9 shows each of the equations in about $5\,s$. Their conjunction takes several minutes. The full input and output files as well as further information including the number of proofsteps and exact running times, can be found at [19]. The files also show how the needed arithmetic is encoded. □

So far we have showed that algebraic reasoning for hybrid systems is feasible. In particular, we have presented a safety property for a concrete hybrid system. Furthermore we have encoded the property with the off-the-shelf theorem prover Prover9 and have proved it fully automatically. Therefore our algebra provides an interesting new way of verifying hybrid systems. Other approaches are discussed in Section 7. It is straightforward to extend the above example. For instance, one can add more locations or one can refine the safety property (e.g., "The security service has to drive to a petrol station every 10 hours and refuel there for 5 minutes".) All these extensions do not change the algebra and/or the way of verifying the specification.

Verifying larger systems might need more time to prove properties fully automatically. But, checking properties are usually done in advance and not in real time. Moreover Prover9 can prove even complex properties in reasonable time; see e.g. Back's atomicity refinement law in [21]. Therefore we expect that one can use our approach for larger systems, too.

[2] We use a Pentium 4, 3 GHz with Hyper-Threading, 2 GB RAM.

6 Case Study II — An Assembly Line Scheduler

To further underpin our approach we sketch a more complex example: an assembly line scheduler that must assign elements from an incoming stream to one of two assembly lines [15].

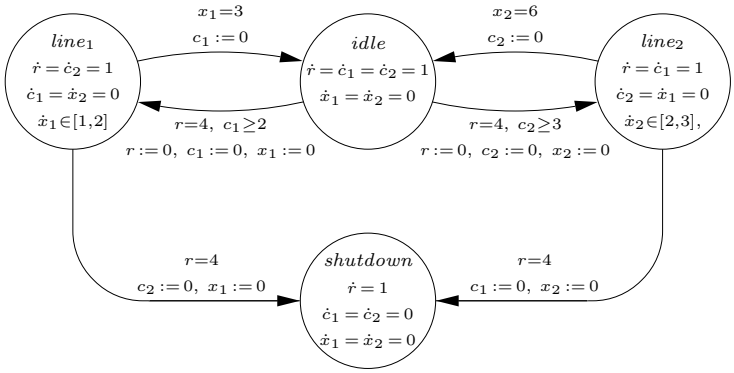

Fig. 4. Two assembly lines

New parts occur every four minutes in the stream. The lines themselves process the parts at different speeds: jobs travel between one and two meters per minute on the first line, while on the second the speed is between two and three metres per minute. The first line is three metres, the second six metres long. Once the lines finish a job, they insert cleaning phases of two and three minutes, respectively, during which no job can be taken up. The whole system accepts a job if both lines are free, and at most one is cleaning up. If the system cannot accept a job it shuts down.

The system is modeled by a hybrid automaton (Figure 4). There are four states: in *idle* no jobs are being processed; in $line_1$ and $line_2$ the lines for processing jobs are modelled; in *shutdown* the system shuts down. The variables x_1 and x_2 measure the distance a job has travelled along the first and second line, respectively. The variable c_1 and c_2 indicate the amount of time for cleaning up. Finally the variable r measures the elapsed time since the last arrival of a job.

As a liveness property one wants to avoid the system to go down. In [16] it is mentioned that any feasible schedule must choose the first line infinitely often. We will characterise this liveness property in our algebraic setting. Similar to Section 3 we define sets of trajectories l_1, l_2, i and s for the nodes $line_1$, $line_2$, *idle* and *shutdown* respectively (see Appendix A for the definitions). Since s is an error state we further assume that the corresponding process only consists of trajectories of infinite length. (If it is reached once, it will never be left.)

$$s =_{df} \{(d, g) \mid d = \infty,\ \dot{r} = 1,\ \dot{c}_1 = \dot{c}_2 = \dot{x}_1 = \dot{x}_2 = 0\} ,$$

with $g =_{df} r \times c_1 \times c_2 \times x_1 \times x_2$. We want to use the following statement:

"If the system is not in state *shutdown*, it must be in one of the other states."

Using the set of *all* trajectories TRA we cannot characterise such an behaviour. Therefore we have to pick a subalgebra of TRA.

Lemma 6.1. *Let $A \subseteq$ TRA a set of trajectories. Then the structure*

$$\mathrm{PRO}(A) =_{df} (\mathcal{P}(A^* \cup A^\omega), \cup, \emptyset, \cdot, I)$$

forms a Boolean left omega algebra.

To model liveness properties concerning the assembly line scheduler, we calculate in $\mathrm{PRO}(l_1 \cup l_2 \cup i \cup s)$. The property that the system never reaches the state *shutdown* is now equivalent to the statement of never leaving the other states. The liveness property can be encoded as

$$(\mathsf{F} \cdot l_1)^\omega \leq (l_1 + l_2 + i)^\omega \ ,$$

where F denotes the set of all trajectories with finite duration. (F exists and can be defined in a general setting (e.g. [18, 25]); here we only focus on applications and omit the theory.)

By coinduction and the hypothesis that $\mathsf{F} \leq (l_1 + l_2 + i)^*$ the claim follows immediately and can also be proved automatically. The hypothesis is by the additional assumption on s and can also be proved with Prover9 within 1 second. Details, like a proof by hand, can be found in Appendix A.

Therefore we have proved a liveness criterion for the assembly line scheduler.

7 Related Work

Although there is some related work concerning the verification of hybrid systems, we are not aware of any verification techniques based on first-order equational reasoning. But this is the key to using paramodulation-based first-order theorem provers.

Many verification techniques are based on hybrid automata [2]. But all these do not yield an algebraic approach; therefore no equation-based reasoning is possible. Furthermore, higher order theorem provers exist and are used to verify properties of hybrid systems. One of them is KeYmaera that extends the theorem prover KeY with Mathematica. It is a special purpose prover designed just for the verification of hybrid systems. Its advantage compared to our approach is that it also integrates arithmetic operators (see Section 8); but it needs a lot of interaction, since KeY is a higher-oder prover. HyTech is a modell-checker for hybrid systems. In [16] a preprocessor for HyTech is implemented which handles a limited version of LTL. A detailed comparison between that approach and our algebraic characterisation is still missing. A discussion on further related work is omitted for lack of space.

8 Conclusion and Outlook

In the paper we have showed that a trajectory-based algebra can be used to specify and verify safety and liveness properties. Algebraisation yields simple

and short calculations. Moreover, these proofs can be automated with first-oder theorem provers.

The presented work is only a first step of still on-going work. On the one hand the examples are still small. For that reason we want to do more case studies with larger systems. As a base we plan to use the examples of [6, 26].

Although we have showed that the algebraic approach combined with first-order theorem proving is feasible, one still has to integrate arithmetics in our approach. So far we have derived preconditions by hand; namely the arithmetic constraints in the first example and the condition $\mathsf{F} \leq (l_1 + l_2 + i)^*$ in the second. It would be interesting to see how this can be generalised and automated. At the moment we have two alternatives in mind: (1) There is some theory how to combine first-order theorem proving with arithmetics. In particular, for arithmetics based on integers there exists SPASS+T [27]. (2) In [16] HyTech is used to locally analyse hybrid systems. The outcome could be used to characterise and generate preconditions for our approach.

Acknowledgements. I am grateful to Georg Struth and Bernhard Möller for valuable remarks and discussions. Further I thank Martin Magnusson for discussions concerning the security service example.

References

1. K. Aboul-Hosn and D. Kozen. KAT-ML: An interactive theorem prover for Kleene algebra with tests. *Journal of Applied Non-Classical Logics*, 16(1–2):9–33, 2006.
2. 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 Comp. Sc.*, 138(1):3–34, 1995.
3. 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 *Hybrid Systems*, pages 209–229. Springer, 1993.
4. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Comp. Sc.*, 126(2):183–235, 1994.
5. J. A. Bergstra and C. A. Middleburg. Process algebra for hybrid systems. *Theoretical Comp. Sc.*, 335(2-3):215–280, 2005.
6. K.-H. Cho, K. H. Johansson, and O. Wolkenhauer. A hybrid systems framework for cellular processes. *Biosystems*, 80(3):273–282, 2005.
7. J. H. Conway. *Regular Algebra and Finite Machines.* Chapman & Hall, 1971.
8. J. M. Corbett. Designing hybrid automated manufacturing systems: A european perspective. In *Conference on Ergonomics of Hybrid Automated Systems I*, pages 167–172. Elsevier, 1988.
9. W. Damm, H. Hungar, and E.-R. Olderog. On the verification of cooperating traffic agents. In F. S. de Boer, M. M. Bonsangue, G. Susanne, and W. P. de Roever, editors, *Formal Methods for Components and Objects*, volume 3188 of *LNCS*, pages 77–110. Springer, 2004.
10. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order, 2nd ed.* Cambridge University Press, 2002.
11. J. M. Davoren and A. Nerode. Logics for hybrid systems. *Proc. of the IEEE*, 88(7):985–1010, 2000.

12. J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Trans. Comp. Logic*, 7(4):798–833, 2006.
13. J. Faber and R. Meyer. Model checking data-dependent real-time properties of the european train control system. In *FMCAD '06*, pages 76–77. IEEE Press, 2006.
14. T. A. Henzinger. The theory of hybrid automata. In *IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292. IEEE Press, 1996. Extended Version: In Kemal M. Inan and Robert P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series F: Computer and Systems Sciences*, pages 265–292. Springer, 2000.
15. T. A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *CONCUR '99*, LNCS, pages 320–335. Springer, 1999.
16. T. A. Henzinger and R. Majumdar. Symbolic model checking for rectangular hybrid systems. In *TACAS '00*, volume 1770 of *LNCS*, pages 142–156. Springer, 2000.
17. P. Höfner and B. Möller. Towards an algebra of hybrid systems. In W. MacCaull, M. Winter, and I. Duentsch, editors, *Relational Methods in Computer Science*, volume 3929 of *LNCS*, pages 121–133. Springer, 2006.
18. P. Höfner and B. Möller. An algebra of hybrid systems. Technical Report 2007-08, Institut für Informatik, Universität Augsburg, 2007.
19. P. Höfner and G. Struth. <`http://www.dcs.shef.ac.uk/~georg/ka`>. January 14, 2008.
20. P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfennig, editor, *CADE 2007*, volume 4603 of *LNAI*, pages 279–294. Springer, 2007.
21. P. Höfner and G. Struth. Can refinement be automated? In E. Boiten, J. Derrick, and G. Smith, editors, *Refine 2007*, ENTCS. Elsevier, 2007. (To appear).
22. W. Kahl. Calculational relation-algebraic proofs in Isabelle/Isar. In R. Berghammer, B. Möller, and G. Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science*, volume 3051 of *LNCS*, pages 179–190. Springer, 2004.
23. D. Kozen. Kleene algebra with tests. *Trans. Prog. Languages and Systems*, 19(3):427–443, 1997.
24. W. McCune. Prover9 and Mace4. <`http://www.cs.unm.edu/~mccune/prover9`>.
25. B. Möller. Kleene getting lazy. *Sc. Comp. Prog.*, 65:195–214, 2007.
26. O. Müller and T. Stauner. Modelling and verification using linear hybrid automata – A case study. *Math. and Comp. Modelling of Dynamical Systems*, 6(1):71–89, 2000.
27. V. Prevosto and U. Waldmann. SPASS+T. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *ESCoR: FLoC'06*, volume 192 of *CEUR Workshop Proceedings*, pages 18–33, 2006.
28. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
29. G. Struth. Calculating Church-Rosser proofs in Kleene algebra. In H. de Swart, editor, *Relational Methods in Computer Science, 6th International Conference*, volume 2561 of *LNCS*, pages 276–290. Springer, 2002.
30. G. Sutcliffe and P. Yury. SRASS-A semantic relevance axiom selection system. In F. Pfennig, editor, *CADE 2007*, volume 4603 of *LNAI*, pages 295–310. Springer, 2007.

A Omitted Details for the Assembly Line Scheduler

In the example of the assembly line scheduler all functions are real-valued, i.e., $r, c_1, c_2, x_1, x_2 : \mathbb{R} \to \mathbb{R}$; the set of durations is $\mathbb{R}$, too. The processes l_1, l_2, i and s are defined as follows:

$$\begin{aligned}
l_1 &=_{df} \{(d,g) \,|\, \dot{r} = \dot{c}_2 = 1,\ \dot{c}_1 = \dot{x}_2 = 0,\ \dot{x}_1 = [1,2]\}\ , \\
l_2 &=_{df} \{(d,g) \,|\, \dot{r} = \dot{c}_1 = 1,\ \dot{c}_2 = \dot{x}_1 = 0,\ \dot{x}_2 = [1,2]\}\ , \\
i &=_{df} \{(d,g) \,|\, \dot{r} = \dot{c}_1 = \dot{c}_2 = 1,\ \dot{x}_1 = \dot{x}_2 = 0\}\ , \\
s &=_{df} \{(d,g) \,|\, d = \infty,\ \dot{r} = 1,\ \dot{c}_1 = \dot{c}_2 = \dot{x}_1 = \dot{x}_2 = 0\}\ ,
\end{aligned}$$

where g is defined as $g = r \times c_1 \times c_2 \times x_1 \times x_2$ and just collects all information of the behaviour. By coinduction,it is sufficient to show that $(\mathsf{F} \cdot l_1)^\omega \leq (l_1 + l_2 + i)^* \cdot (\mathsf{F} \cdot l_1)^\omega$. This follows from unfold, neutrality of 1, finiteness of 1 ($1 \leq \mathsf{F}$), unfold again and the assumption:

$$(\mathsf{F}\cdot l_1)^\omega = \mathsf{F}\cdot l_1 \cdot (\mathsf{F}\cdot l_1)^\omega \leq \mathsf{F}\cdot\mathsf{F}\cdot l_1 \cdot (\mathsf{F}\cdot l_1)^\omega = \mathsf{F}\cdot(\mathsf{F}\cdot l_1)^\omega \leq (l_1 + l_2 + i)^* \cdot (\mathsf{F}\cdot l_1)^\omega.$$

B Prover9 Source Code

Left omega algebras can be encoded in Prober9 as follows:

```
op(500, infix_left, "+").  %choice
op(490, infix_left, ";").  %composition
op(480, postfix,    "*").  %finite iteration
op(450, postfix,    "^").  %infinite iteration (omega)

formulas(sos).
% standard axioms of idempotent left semirings %%%%%%%%%%%%%
    x+y = y+x.                %commutative additive monoid
    x+0 = x.
    x+(y+z) = (x+y)+z.        %multiplicative monoid
    x;1 = x & 1;x = x.
    x;(y;z) = (x;y);z.
    0;x = 0.                  %annihilation laws
    x+x = x.                  %idempotence
    (x+y);z = x;z+y;z.        %distributivity

% standard axioms for finite iteration (star) %%%%%%%%%%%%%%
    1+x;x* = x*.
    (x;y+z)+y=y -> x*;z+y=y.

% standard axioms for infinite iteration (omega) %%%%%%%%%%%
    x;x^= x^.
    y+(x;y+z)=x;y+z -> y+(x^+x*;z)=x^+x*;z.

end_of_list.

formulas(goals).
  %lemma to be proved
end_of_list.
```

There exist also other implementations, e.g. an inequational encoding. They can be found at our website, too.