UNIVERSITÄT
KOBLENZ · LANDAU
Institut für Informatik

FB 4
Informatik

Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata

Ammar Mohammed
Frieder Stolzenburg

Nr. 6/2009

Arbeitsberichte aus dem Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Zöbel

Kontaktdaten der Verfasser

Ammar Mohammed, Frieder Stolzenburg
Institut für Informatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: ammar@uni-koblenz.de, fstolzenburg@hs-harz.de

Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata

Ammar Mohammed
Universität Koblenz-Landau
Computer Science Department
Universitätsstr. 1
56070 Koblenz, Germany
ammar@uni-koblenz.de

Frieder Stolzenburg
Hochschule Harz
Automation and Computer Sciences Department
38855 Wernigerode, Germany
fstolzenburg@hs-harz.de

Hybrid systems are the result of merging the two most commonly used models for dynamical systems, namely continuous dynamical systems defined by differential equations and discrete-event systems defined by automata. One can view hybrid systems as constrained systems, where the constraints describe the possible process flows, invariants within states, and transitions on the one hand, and to characterize certain parts of the state space (e.g. the set of initial states, or the set of unsafe states) on the other hand. Therefore, it is advantageous to use constraint logic programming (CLP) as an approach to model hybrid systems. In this paper, we provide CLP implementations, that model hybrid systems comprising several concurrent hybrid automata, whose size is only straight proportional to the size of the given system description. Furthermore, we allow different levels of abstraction by making use of hierarchies as in UML statecharts. In consequence, the CLP model can be used for analyzing and testing the absence or existence of (un)wanted behaviors in hybrid systems. Thus in summary, we get a procedure for the formal verification of hybrid systems by model checking, employing logic programming with constraints.

1 Introduction

1.1 Motivation

Hybrid automata [20] are a standard means for the specification and analysis of dynamical systems, where computational processes interact with physical processes. Essentially, hybrid automata are state machines for describing discrete-event systems, augmented with differential equations for the treatment of continuous processes. They are widely used for the specification of embedded systems. There are numerous applications, e.g. in the fields of robotics, logistics,

multiagent systems, and for technical systems in general, especially in safety-critical contexts, where formal verification of system properties is desirable.

There are several model checking tools for hybrid automata available, e.g. HyTech [23] or PHAVer [14]. In the context of hybrid automata, the terms formal verification and model checking usually refer to reachability analysis, i.e. to the question whether some (un)wanted state is reachable from the initial configuration of the specified system. For this, systems of (linear) (in)equations have to be solved, which is usually implemented by algorithms manipulating sets of convex polyhedra. In this paper, it is demonstrated that model checking of hybrid systems can be understood as constraint solving. Hence, it appears to be a good idea to employ constraint logic programming (CLP) [31] for this task, which extends logic programming with Prolog [11].

In this paper, we therefore propose a methodology that exploits CLP for the specification and analysis of hybrid systems. CLP has already been applied to model hybrid systems including solving differential equations (see e.g. [26]). However, efficiency can only be expected, if a full CLP language is employed as e.g. Eclipse Prolog [4], where a multitude of constraint solvers is available. By introducing hierarchies (as in UML statecharts, cf. [33]) different levels of abstraction besides representation of concurrency in the specification are expressible, which is certainly advantageous.

Usually hierarchical specifications of hybrid systems are transformed into flat standard finite hybrid automata (see e.g. [5, 15, 35]). Concurrent automata have to be composed, which leads to the state explosion problem, because the number of states in the resulting flat automaton is the product of the number of states of all concurrent automata. As demonstrated in this paper, computing the composition of automata can be avoided by employing CLP, where in addition many efficient constraint solvers e.g. for interval constraints and finite domains are available.

1.2 Overview on the Rest of the Paper

In summary, the main contributions of this paper are as follows: First, we present a lean but effective implementation of hybrid automata, that hosts an explicit formulation of hierarchies and concurrency. Second, compositions of automata do not have to be computed explicitly, which avoids the state explosion problem. Instead, we generate configurations of the whole system only if required, and thus the size of the corresponding CLP program is only straight proportional to the size of the given hierarchical hybrid automaton description. Last but not least, by employing CLP, constraints can be derived automatically, under which certain states of the system can be reached. This enhances standard formal verification and model checking methodologies.

In the following, we therefore introduce our formalism of *hierarchical hybrid automata* (HHA) with a running example, namely a railroad gate controller, that is well-known in the literature [21] (Sect. 2). Then, we describe its implementation with CLP (Sect. 3), by introducing an abstract state machine for HHA eventually. We briefly compare different implementations for model checking hybrid automata and discuss related works (Sect. 4), before we end up with conclusions (Sect. 5).

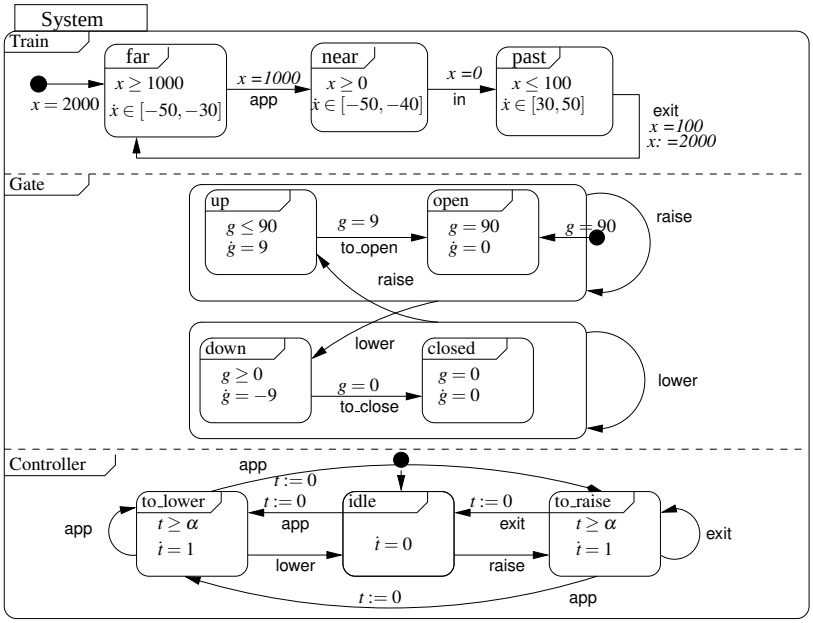

Figure 1: Specification of the train gate controller automata.

2 Example and Formalism

Before we are going to present the definitions and formalism for HHA, we first introduce an illustrating running example that we use throughout the paper, followed by the basic formalism which we use to demonstrate the CLP implementation.

2.1 The Railroad Gate Controller Example

A train gate controller [21] is a system consisting of three automata components: the train, the gate, and the controller. In this system, a road is crossing a train track, that is guarded by a gate, which must be lowered to stop the traffic when the train approaches, and raised after a train passed the road. The gate is supervised by a controller that has the task to receive signals from the train and to issue lower or raise signals to the gate. Initially, a train is at a distance of 2000 m far from the gate and moves at a speed between 40 and 50 m/s. When the train reaches the gate at a distance of 1000 m, it issues an *app* signal to the controller (with the meaning that the train approaches the gate) and may slow down to 30 m/s.

When the controller is idle upon receipt of the approach event *app*, it requires up to α seconds to send the command *lower* to the gate. At the distance of 100 m past the gate, the train issues an *exit* signal to the controller, which after another delay of up to α seconds, it sends the command *raise* to the gate. Initially, the gate is completely opened in a position of 90 radius degrees. Upon receiving the *lower* signal at the open position, the gate is lowered from 90 radius degrees to 0 degrees at a constant rate of 9 degrees per second, and the same holds when it is closed upon receipt of the command *raise*.

The three hybrid automata that model the train, the gate, and the controller are shown in Fig. 1. In the graphical representation, the variable x represents the distance of the train from the gate. The variable t represents the delay time of the controller, while the position of the gate in radius degrees is represented by the variable g.

First, the train gate controller system must satisfy a *safety property*. The purpose of the safety property is to ensure that the system cannot reach an unsafe state, i.e. a state where the train is in the crossing but the gate is not closed. Second, in addition to safety property, a *utility property* has to hold. The purpose of the utility property is to avoid degenerate solutions, e.g. lowering the gate and keeping it lowered. Basically, safety critical system must not only operate safely. To be useful, they must perform certain functions within specified time intervals, i.e., they must exhibit response times within given bounds. For example, the train leaves the crossing within 36 s after its approach. Third, another interesting property that appears generally in systems with synchronization is a simple *logical property*. That is a property that depends on logical and/or temporal dependences between events.

2.2 Formalism

Let us now define formally the notation used in this paper, before we introduce a CLP implementation of hybrid automata (in Sect. 3). We first define basic components of hybrid automata (Def. 1). Then, we introduce state hierarchies and concurrency (Def. 2 and 3), before we define the semantics of HHA (Def. 4). For further details and examples, especially on the latter definitions, the reader is referred to Sect. 3.3.

Definition 1 (basic components) *A hybrid automaton is a tuple* $H = (X, Q, Inv_q, Flow_q, E, Jump, Event, Init)$ *where:*

- $X \subseteq \mathbb{R}^n$ *is a finite set of n real-valued variables.*
- Q *is a finite set of control locations. For example, the train automaton (Fig. 1) has the locations far, near, and past.*
- Inv_q *is the invariant predicate, which assigns a constraint on variables X for each control location* $q \in Q$*. For example, the location far in the train automaton has the invariant* $x \geq 1000$*. When the hybrid automaton H is in a control location* $q \in Q$*, the variables in X must satisfy the invariant* Inv_q*.*
- $Flow_q$ *is the flow predicate on variables X for each control location* $q \in Q$*, which defines how the variables in X evolve over the time at location q. In the graphical representation, a flow of a variable x is represented as* $\dot{x}$*. For example,* $\dot{g} = 9$ *describes the continuous activity of the gate.*
- $E \subseteq Q \times Q$ *is the discrete transition relation over the control locations. Each edge* $e \in E$ *is augmented by the following annotations:*

  Jump: *jump condition (guard), which is a constraint over X; if the jump condition holds, the transition e fires and may change the values of the variables X by executing a specific action;*

Event: *synchronization label, used to synchronize concurrent automata; the train automaton contains the synchronization labels app, exit, and in, that must be synchronized with all automata sharing the same synchronization labels.*

- *Init is the initial condition that assigns an initial condition to the variables X to each control location $q \in Q$. For example, $x = 2000$ is the initial condition of the train automaton, while it is $g = 90$ at the location open in the gate automaton.*

Hybrid systems typically consist of several components which operate concurrently and communicate with each other. Each component usually is described as a separate hybrid automaton. The automata are coordinated through shared variables and synchronization labels on the transitions [20]. For example, in Fig. 1, the train automaton communicates with the controller automaton via the synchronization labels *app* and *exit*. Describing the behavior of the hybrid automata demands for a mechanism to coordinate the execution among automata. This can be accomplished by means of automata composition. The result of the composition process is an automaton that describes the entire behavior of the hybrid system. Basically, the composition of automata is done by means of the Cartesian product of the automata, but automata with mutual synchronization labels have to be considered simultaneously, which helps to reduce the complexity of automata composition.

In hierarchical hybrid automata (HHA), locations are generalized to states, stemming from the set of states S. It is partitioned into three disjoint sets: S_{simple}, S_{comp}, and S_{conc} — called simple, composite and concurrent states, containing one designated *start state* $s_0 \in S_{comp} \cup S_{conc}$. In essence, the locations of plain hybrid automata correspond to simple states in HHA. Composite and concurrent states belong to the definition of statecharts [19] and have become part of UML [33]. They are useful for expressing the overall system on several levels of abstraction. Events are treated as global variables in this context. Based on this, we will now introduce the concepts of HHA. For the sake of completeness, we adopt and restate some of the definitions of [15], which describes some case studies with standard model checking tools, not employing CLP. For more details on this and the synchronization concept of HHA, the reader is referred to this paper.

Definition 2 (state hierarchy) *Each state s is associated with zero, one or more* initial states $\alpha(s)$*: a simple state has zero, a composite state exactly one, and a concurrent state more than one initial state. In the latter case, the initial states are called* regions*. Moreover, each state $s \in S \setminus \{s_0\}$ is associated to exactly one superior state $\beta(s)$. Therefore, it must hold $\beta(s) \in S_{conc} \cup S_{comp}$. A concurrent state must not directly contain other concurrent ones. Furthermore, it is assumed that all transitions (s_1, s_2) keep to the hierarchy, i. e. $\beta(s_1) = \beta(s_2)$. Variables $x \in X$ may be declared locally in a certain state $\gamma(x) \in S$. A variable $x \in X$ is* valid *in all states $s \in S$ with $\beta^n(s) = \gamma(x)$ for some $n \geq 0$ (i.e. in all states below $\gamma(x)$ in the state hierarchy), unless another variable with the same name overwrites it locally.*

For the example in Fig. 1, we consider the states *train*, *gate*, and *controller* as composite states, that are regions of one concurrent state, that represents the whole *system*. Thus, according to the previous Def. 2, it holds e.g.: $\alpha(train) = far$, $\alpha(gate) = open$, $\alpha(controller) = idle$, $\alpha(system) = \{train, gate, controller\}$; $\beta(near) = train$, $\beta(train) = system$; $\gamma(x) = train$, $\gamma(g) = gate$, $\gamma(t) = controller$.

Definition 3 (configuration and completion) *A* configuration *c is a rooted tree of states where the root node is the topmost initial state s_0 of the overall state machine. Whenever a state s is an immediate predecessor of s' in c, it must hold $\beta(s') = s$. A configuration is* completed *by applying the following procedure recursively as long as possible to leaf nodes: if there is a leaf node in c labeled with a state s, then introduce all $\alpha(s)$ as immediate successors of s.*

A hybrid automaton may change in two ways: *discretely*, from location q_1 to another location q_2, when the transition $e \in E$ between the two locations is enabled (i.e., the jump condition holds) and *continuously* within a control location $q \in Q$, by means of a finite (positive) time delay t. The semantics of our automata can now be defined by alternating sequences of discrete and continuous steps. Following the synchrony hypothesis, we assume that discrete state changes happen in zero time, while continuous steps (within one state) may last some time.

Definition 4 (semantics) *The state machine starts with the* initial configuration*, i.e. the completed topmost initial state s_0 of the overall state machine. In addition, an initial condition must be given as a predicate with free variables from $X \cup \{t\}$. The current* situation *of the whole system can be characterized by a triple (c,v,t) where c is a configuration, v a valuation (i. e. a mapping $v : X \to \mathbb{R}^n$), and t the current time. The* initial situation *is a situation (c,v,t) where c is the initial configuration, v satisfies the initial condition, and $t = 0$. The following steps are possible in the situation (c,v,t):*

discrete step: *a discrete/micro-step from one configuration c of a state machine to a configuration (c',v',t) by means of a transition $(s,s') \in E$ with some jump condition in the current situation (written $c \to c'$) is possible iff:*

1. *c contains a node labeled with s;*
2. *the jump condition of the given transition holds in the current situation (c,v,t);*
3. *c' is identical with c except that s together with its subtree in c is replaced by the completion of s';*
4. *the variables in X' are set according to the jump condition.*

continuous step: *a continuous step/flow within the actual configuration to the situation (c,v',t') requires the computation of all $x \in X$ that are valid in c at the time t' according to the conjunction of all state conditions (i.e. flow conditions plus invariants) of the active states $s \in c$, where it must hold $t' > t$.*

The cautious reader may have noticed that invariants (see Def. 1) are merged here with the flow conditions in continuous steps (see Def. 4). In particular, while jump conditions are checked during a discrete transition, flow and invariant conditions are only tested at the beginning and at the end of a continuous flow within one configuration, i.e. only at the boundaries. Hence, it would not be detected that a variable $x(t)$, that is checked only at the times t_1 and t_3 with e.g. $x(t_1) = x(t_3) = 0$, does not satisfy an an invariant, say $x \leq c$ for some constant $c > 0$, if there exists a time point t_2 with $t_1 < t_2 < t_3$ and $x(t_2) > c$. Since it appears to be time-consuming to detect such cases, we do it without such tests. In practice, these case should not occur very often. For monotonous flow functions $x(t)$ and linear inequalities of the form $x \leq c$ such an expensive test is not necessary at all.

3 Implementation with CLP

In this section, we provide CLP models of hybrid automata in two steps. The first one is a straightforward implementation of plain hybrid automata without explicitly computing the composition of different parts of automata. There, we enforce the time constraints over the generated events during transitions to coordinate the execution of hybrid automata. After that, we present an implementation of HHA that allows modeling hybrid automata at different levels of abstraction by expressing hierarchies and concurrency directly, based on an explicit abstract state machine programmed in Prolog (Sect. 3.3). In all cases, we employed the *ic* library for interval constraints, available with ECLiPSe Prolog [4], which also includes finite domain constraint solving. SWI Prolog [36] also offers interesting functionality and could be used alternatively.

3.1 Straightforward Implementation

In the following, we give a CLP model for analyzing hybrid systems consisting of different interacting concurrent automata. Our model follows the formal definition of hybrid automata (Def. 1) and the semantics of the labeled transition semantics of hybrid automata, too (cf. [20]).

We start by modeling locations. They are implemented in the `automaton` predicate, ranging over the respective locations of the automaton, real-valued variables, and the time:

```
automaton(+Location,?Vars,+Vars0,+T0,?Time):-
                  Vars#c2(Vars0,T0,Time),
                  c1(Inv),Time $>=T0.
```

Here, `automaton` is the name of automaton itself, and `Location` represents the current locations of the automaton. `Vars` is a list of real variables participating in the automata, whereas `Vars0` is a list of the corresponding initial values. `c1(Invs)` is the invariant constraint inside the location, and the constraint predicate `Vars` $\bowtie$ `c2(Vars0,T0,Time)` , where $\bowtie \in \{<, \leq, >, \geq, =\}$ are constraints, which represent the continuous flows of the variables in `Vars` wrt. time `T0` and `Time` , given initial values `Vars0` of the variables `Vars` at the start of the flow. `T0` represents the initial time at the start of continuous flow, while `(Time-T0)` is the delay inside the location. For example, the location *far* is modeled as:

```
train(far,Y,Y0,T0,Time):-
      Y $>= Y0-50*(Time-T0),
      Y $=<Y0-40*(Time-T0),
      Y $>=1000, Time $>=T0.
```

At any instance of time, a state of a hybrid automaton is a pair (loc, v), where loc is a location and v is the assignment of values for the variables. Intuitively, the semantics of (standard) hybrid automata can formally be described as runs of labeled transition systems [20], where the execution of a hybrid automaton corresponds to a sequence of transitions from a state to another. Thus, hybrid automata have two kinds of transitions: *continuous* transitions, capturing the continuous evolution of variables, and *discrete* transitions, capturing the changes of location. For this purpose, one can encode the transition system into CLP clauses. The predicate `evolve` achieves this mission:

```
evolve(Automaton,(L1,Var1),(L2,Var2),T0,Time,Event) :-
      continuous(Automaton,(L1,Var1),(L1,Var2),T0,Time,Event);
      discrete(Automaton,(L1,Var1),(L2,Var2),T0,Time,Event).
```

Each transition is accompanied with a guard that must fire, when a discrete transition takes place. In our model, the guard is represented as constraint relation of $c(T0, Time)$, where `Time` is the minimum elapsed time needed to generate an event (a discrete transition) and is computed during the evolvement of automata. When a discrete transition occurs, it gives raise to update the initial variables from `Var1` into `Var2` , where `Var1` and `Var2` are the initial variables of locations `L1` and `L2` respectively. Otherwise, a delay transition is taken using the predicate `continuous` . In addition, an $event \in Event_{Automaton}$ is associated with each transition, that defines the parallel composition from the automata individual sharing the same event. To this end, we augment the predicate `evolve` with a constraint variable `Event` that ranges over symbolic domains. It guarantees that whenever an automaton generates an event, the corresponding synchronized automata have to be taken into consideration simultaneously. When an automaton generates an event, the symbolic domain solver will exclude all the domain values that are not coincident with the generated event from the automata having the common event. This means that only one event is generated at a time. Consequently, it shows that the automata composition can be implicitly constructed efficiently on the fly, during the computation. The following is the general implementation of the `discrete` predicate, which defines transitions between locations:

```
discrete(automaton,(Loc1,Var1),(Loc2,Var2),T0,Time,Event):-
          automaton,(Location,Var1,Var,T0,Time),
    jump(Var), reset(Var2)
    Event &::events,Event &=event.
```

Here, $Jump(Var)$ represents the constraints of the jump condition on the variables Var, whereas $reset(Var2)$ is a constraint predicate used to reset the variable $Var2$ before the control of the automaton goes to the location $Loc2$. Here, the `Event` must be a member in $Event_{Automaton}$. The & symbol is the constraint relation for symbolic domains (library *sd* in ECLiPSe Prolog), while the $ symbol (see below) marks interval constraints (library *ic*). It follows an instance showing the implementation of the `discrete` predicate between locations *far* and *near* in automaton *train*.

```
discrete(train,(far,[X0]),(near,[XX0]),T0,Time,Event):-
            train(far,[X0],[X],T0,Time),
      X $=1000, XX0 $=X,
      Event &::events,Event &=app.
```

The description of the above *discrete* predicate means that a transition between the locations *far* and *near* in the *train* automata takes place if the continuous variable `X` , based on the initial value `X0`, satisfies the jump condition given as `X=1000` . If such a case occurs, then the new variable, denoted `XX0` , is updated, and the event *app* is fired. The executed events afterwards synchronize the *train* automaton with the automata sharing the same event.

Once the transition rules have been modeled, a driver program needs to be supplied:

```
drive(_,_,..,_,_,0,[]):- !.
driver((L1,Var01),(L2,Var02),...,(Ln,Var0n),T0,Steps,
          [(L1,L2,..,Ln,Var1,Var2,..,Varn,Time,Event)|NextReached]) :-
      automata1(L1,Var1,Var01,T0,Time1),
```

```
      automata2(L2,Var2,Var02,T0,Time2),
      ... ,
      automatan(Ln,Varn,Var0n,T0,Timen),
      Time1 $=Time2, Time1 $=Time3, ..., Time1 $=Timen
      evolve(automata1,(L1,Var01),(NextL1,Nvar01),T0,Time1,Event),
      evolve(automata2,(L2,Var02),(NextL2,Nvar02),T0,Time2,Event),
      ... ,
      evolve(automatan,(Ln,Var0n),(NextLn,Nvar0n),T0,Timen,Event),
      get_bounds(Time1,_,Newstarttime),
      Steps > 0,Steps1 is Steps -1,
driver((NextL1,Nvar01),(NextL2,Nvar02),...,(NextLn,Nvar0n),
        Newstarttime,Steps1,SNextReached).
```

The `driver` is a simulator predicate that is responsible to generate and control the execution behavior of the concurrent hybrid automata, as well as to provide the reachable states symbolically. Recall again, *Event* is a symbolic domain variable shared among all automata, where the solver uses it to ensure that only one event is generated at a time. All automata sharing the same events have to be synchronized. During the computational procedure, from the times of all automata the minimum time among the automata is determined. This minimum time is the time needed to fire an event. Consequently, the predicate *evolve*, based on this time, alternates each automaton between continuous and discrete transitions. To prevent the driver from infinite runs, the number of discrete steps should be provided in advance. The last argument of the predicate *driver* is the list of finitely reachable regions. At each step of the driver, a region of the form $(location, Variables)$ represents symbolically by arithmetical constraints the set of states reachable to each control location. Additionally, each region contains the time delay of the continuous variables. Finally, each region contains the event generated immediately before the control goes to another region using a discrete step. The driver of the train gate controller example (Fig. 1) takes the form

```
driver(_,_,_,0,[]) :- !.
driver((S1,X0),(S2,G0),(S3,T0),Steps,[(S1,S2,S3,Time,Event,X)|Rest]).
```

where `S1`, `S2` , and `S3` represent the locations of the train, the gate, and the controller respectively, while `X0`, `G0`, and `T0` represent their corresponding initial values consecutively. The last argument is a list of possible reached states along with the variable `X` represents symbolically the possible reached values of the train distance. `Time` is the global time of the reached states, and `Event` is the event generated immediately before the control changes to another configuration. It seems to be a good idea that the reachable states contain only the variables that are important for the verification of a given property. Therefore, the last argument list of the predicate `driver` can be expanded or shrunk as needed to contain the really important variables involved, e.g. the variables G, T, and α of the automata named *gate* and *controller*.

3.2 Verification as Reachability Analysis

After setting up the driver, we have an executable CLP model for hybrid automata, and several properties can now be investigated. In particular, one can check properties on states using reachability analysis of hybrid automata. The reachable set consists of all states that can be reached by dynamical evolution, starting from an initial state. However, it is well-known, that checking

reachability for (linear) hybrid automata is undecidable in general (while it is decidable for timed automata) [22].

Reachability analysis consists of two basic steps: computing the state space of the automaton under consideration and searching for states that satisfy or contradict given properties. In terms of CLP, a state is reached iff the constraint solver succeeds in finding a satisfiable solution for the constraints representing the intended state. In other words, assuming that *Reached* represents the set of all reachable states computed by the CLP model from an initial state, then the reachability analysis can be generally specified, using CLP and checking whether $Reached \models \Psi$, where Ψ is the constraint predicate that describes a property of interest.

In practice, many problems to be analyzed can be formulated as a reachability problem. For example, a safety requirement can be checked as a reachability problem, where Ψ is the constraint predicate that describes forbidden states, then checking Ψ is not satisfiable wrt. *Reached*. For example, one can check that the state, where the train is near at distance $X = 0$ and the gate is closed, is a disallowed state. Even a stronger condition can be investigated, namely that the state where the train is near at distance $X = 0$ and the gate is down, is a forbidden state. The CLP computational model, with the help of the standard Prolog predicate *member*/2, gives us the answer *no* as expected, after executing the following query:

```
?- driver((far,2000),(open,90),(idle,0),Steps,Reached),
   member((near,down,_,Time,_,X,),Reached), X $= 0.
```

Other properties concerning the reachability of certain states can be verified similarly.

As demonstrated in Sect. 3.1, the set of reachable states *Reached* contains the set of finite, reachable regions. Within each region, the set of all states is represented symbolically as a mathematical constraint, together with the time delay. Therefore, ideally constraint solvers can be used to reason about the reachability of interesting properties within some region. For example, an interesting property is to find the shortest distance of the train to the gate before the gate is entirely closed. This can be checked by posing the following query:

```
?- driver((far,2000),(open,90),(idle,0),Steps,Reached),
   member((near,_,_,Time,to_close,_),Reached), get_max(Time,Tm),
   member((near,_,_,Tm,_,X),Reached), get_min(X,Min).
```

Setting $\alpha = 9.8$, the previous query returns $Min = 9.99$, which is the minimum distance of the train that the model guarantees before the gate is completely closed. This query gets the interval value of the variable X, when the event *to_close* is raised. Consequently, the distance, that is reached when the event *to_close* is generated, is constrained to $406.0 \geq X \geq 9.99$. Hence $Min = 9.99$ is the minimum distance of the train that the model guarantees before the gate is completely closed.

As just said, the previous verification experiments run with setting $\alpha = 9.8$. The ability to compute the value of a parameter, however, is a great advantage of the CLP approach. Ideally, CLP can be used to find a condition on some parameters that violates a given safety property. For this purpose, we can use our model to provide us e.g. with the value of the cutoff point α for the controller to issue commands that causes a bad state to be reached. In particular, we can find the minimum value required for α to reach to a forbidden state, where the train is at distance $x = 0$ to the gate and the gate is opened.

```
?- driver((far,2000),(open,0),(idle,0),Steps,Reached),
   member((near,open,to_lower,Time,_,X,T,Alpha),Reached), X $= 0.
```

```
complete(T,Rest,State,[State:Var|Complete]) :-
         init(T,State,[Var|Rest],Init,_),
         maplist(complete(T,[Var|Rest]),Init,Complete).

discrete(T,Rest1,Rest2,[State1:Var1|_],[State2:Var2|Conf]) :-
         trans(T,State1,[Var1|Rest1],State2,[Var2|Rest2]),
         complete(T,Rest2,State2,[State2:Var2|Conf]).
discrete(T,Rest1,Rest2,[Top:Var1|Sub],[Top:Var2|Tree]) :-
         Sub \= [],
         maplist(discrete(T,[Var1|Rest1],[Var2|Rest2]),Sub,Tree).

continuous(T1,T2,Rest1,Rest2,[State:Var1|Sub],[State:Var2|Tree]) :-
         flow(T1,T2,State,[Var1|Rest1],[Var2|Rest2]),
         maplist(continuous(T1,T2,[Var1|Rest1],[Var2|Rest2]),Sub,Tree).
```

Figure 2: Code for the abstract state machine for HHA in CLP. The `Rest` variables host nested lists of the variables declared in the states superior to the current state. The built-in predicate `maplist` is a macro for applying a predicate call (first argument of `maplist`) to a list of arguments (second and third argument) one by one.

We augment the reached states `Reached` in the previous query with the local timer `T` of the controller automaton together with the cutoff point α. Then the CLP system returns $\alpha \geq 20$. Therefore setting α to any value in this open interval, the forbidden state can be reached.

Since the events are recorded in the reached states, in particular, at the end of the continuous evolution of each reached regions, verifying timing properties or computing the delay between events are further tasks that can be done within our approach, too. For instance, we can find the maximal time delay between *in* and *exit* events, by stating the following query:

```
?- driver((far,2000),(open,0),(idle,0),Steps,Reached),
       append(A,[(_,_,_,Time1,in,_)|_],Reached),
       append(B,[(_,_,_,Time2,exit,_)|_],A),
       get_max(Time1,Tmax1),get_max(Time2,Tmax2),
       Delay $= Tmax1-Tmax2.
```

The constraint solver answers *yes* and yields $Delay = 3.33$. This value means that the train needs maximally 3.33 s to be in the critical crossing section before leaving it. Similarly, other timing properties can be verified.

3.3 Treating Hierarchies and Concurrency More Explicitly

The previous sections described a direct CLP implementation of hybrid automata. Now we will show, how to implement an abstract state machine for HHA, treating hierarchies and concurrency more explicitly. This leads to a lean implementation of hybrid automata, where efficient CLP solvers are employed for performing complex analyses.

Fig. 2 shows parts of the abstract state machine in Prolog, namely the code for completion and for performing discrete and continuous steps according to Def. 3 and 4. Discrete steps take zero

Figure 3: Configuration trees of the running example.

time because of the synchrony hypothesis; continuous steps remain within the same configuration, but the variable values may differ. The flow conditions of active states (in the configuration) must be applied, as time passes by. In this context, configurations are encoded in Prolog lists, where the head of a list corresponds to the root of the respective configuration tree. In addition, each state is conjoined by a colon : with its list of local variables. Thus, according to Def. 3, the completed start configuration will be represented as shown below. Here, the event and the delay α (here represented by the variable `Alpha`) are treated as global variables of the whole system.

```
[system:[none,Alpha],
   [train:[2000],[far:[]]],
   [gate:[90],[open:[]]],
   [controller:[0],[idle:[]]]]
```

The corresponding configuration is shown also as a tree in Fig. 3 (left). Certainly, trees could be represented more efficiently, i.e. consuming less space, than by Prolog lists as shown above. But the use of lists is straightforward and allows us to implement the abstract state machine for HHA (Fig. 2) within only a dozen lines of CLP/Prolog code. By this technique, explicit composition of automata is avoided. For each state, its initial states have to be declared plus their continuous flow conditions. For all discrete transitions, the jump conditions have to be stated. Local variables are expressed by nested list of variables valid in the respective state. Since the abstract state machine is of constant size and the abstract machine computes complex configurations only on demand, there is a one-to-one correspondence between the elements of the HHA and its CLP/Prolog implementation. Thus, the program size is linear in the size of the HHA.

In the concrete implementation of the example, the overall start state s_0 is indicated by the predicate `start` , while `init` defines the initial states for each state (α values according to Def. 2). The flow and the jump conditions have to be expressed by means of the predicates `flow` and `trans` . The reader can easily see from Fig. 4, that the size of the CLP program is only straight proportional to the size of the given HHA, because there is a one-to-one correspondence between the graphical specification and its encoding in Prolog, whereas computing the composition of concurrent automata explicitly leads to an exponential increase. Furthermore, since the overall system behavior is given by the abstract state machine (Fig. 2), this approach is completely declarative and concise.

For reachability analysis, iterative deepening seems to be best suited as search strategy, because an explicit cycle test or breadth-first search is difficult, because otherwise constraints would have to be buffered somehow. After one continuous and one discrete step according to Def. 4, the configuration shown below (see Fig. 3, right) will be reached after 0.0–25.0 s. The event *app* occurs, when the train has traveled 1000 m. Then, the simple states *near* and *to_lower* in the composite states *train* and *controller*, respectively, are entered.

```
%%% system
start(system).
init(T,system,[[Event,Alpha]],[train,gate,controller],_) :-
        Event = none.
flow(T1,T2,system,[[Event,Alpha]],[[Event,Alpha]]).


%%% train
init(T,train,[[X]|_],[far],system) :-
        X $= 2000.
flow(T1,T2,train,_,_).

init(T,far,[[]|_],[],train).
flow(T1,T2,far,[[],[X1]|_],[[],[X2]|_]) :-
        X2 $>= 1000,
        X2 $>= X1-50*(T2-T1),
        X2 $=< X1-40*(T2-T1).
trans(T,far,[[],[X],[Event1,Alpha]],far,[[],[X],[Event2,Alpha]]) :-
        Event2 = lower ; Event2 = raise.
trans(T,far,[[],[X],[Event1,Alpha]],near,[[],[X],[Event2,Alpha]]) :-
        Event2 = app,
        X $= 1000.
```

Figure 4: First part of the HHA implementation of the running example.

```
[system:[app,Alpha],
   [train:[1000],[near:[]]],
   [gate:[90],[open:[]]],
   [controller:[0],[to_lower:[]]]
```

Our experiments with the implementation in ECLiPSe Prolog are encouraging. Employing the *ic* library for interval constraints, the query whether a situation can be reached where the train is at the gate, i.e. $x = 0$, but the gate is open, yields the answer *yes* and the constraints $40.0 \leq T \leq 58.3$ for the time and $\alpha \geq 15$ for the delay of the controller with the smallest possible solution $\alpha = 20$ (after applying the so-called squash procedure [4]). This means, this forbidden state is reached if the delay α of the controller is too long, namely greater or equal than 20 s after 40.0–58.3 s overall time. The answer is as expected, as can be easily checked. Employing the *eplex* library, also available in ECLiPSe Prolog, the lower bound $\alpha = 20$ can also be computed. This library is related to the CPLEX system [27] which provides a more powerful optimization engine that also can be applied for optimization analyses of hybrid systems (see also [34]).

4 Comparison with Other Approaches

This section demonstrates the feasibility of our approach described above. Generally, real-time verification tools vary from simple formalisms for restricted problem classes like timed automata to more expressive formalisms like hybrid automata. It should be remarked that the latter formalisms are more expressive than the former ones. Therefore, tools following the former formalisms as e.g. Uppaal [6, 7] are not discussed here. We did several experiments comparing our approach with HyTech [23]. In contrast to our approach, HyTech treats the continuous dynamics by using a polyhedral manipulation library [18]. We chose HyTech as reference tool, because it is the most well-known tool for verification of hybrid automata, and it tackles verification based on reachability analysis similar to the approach in this paper. It is noteworthy that the major strength of HyTech compared to the other hybrid automata verification tools is its ability to perform parametric analysis. In HyTech, the automata working in parallel are composed, before they are involved in the verification phase. Obviously, this may lead to state explosion as stated earlier.

4.1 Benchmark Examples

In the following, we will refer to standard benchmarks of verification of real-time systems, well-known from the literature, querying these benchmarks in order to check safety properties (cf. Fig. 5). First, in the *scheduler* example [18], it is checked whether a certain task (with number 2) never waits. Second, in the *temperature control* example [1], it has to be guaranteed, that the temperature always lies in a given range. Third, in the *train gate controller* example [21], it has to be ensured that the gate is closed whenever the train is within a distance less than 10 m toward the gate. The second version of the train gate controller example, as formulated in this paper, is used to calculate a parameter analysis, i.e. finding the condition on the parameter α. Last but not least, in the *water level* example [1, 18] the safety property is to ensure that the water level is always between given thresholds (1 and 12). For more details on the examples, the reader is referred to the cited literature.

Example	HyTech	CLP/HA		CLP/HHA	
	seconds	seconds	iterations	seconds	steps
Scheduler	0.12	0.07	6	0.34	12
Temperature Controller	0.04	0.02	6	0.02	12
Train Gate Controller	0.05	0.02	7	0.03	12
Train Gate Controller 2	0.10	0.05	10	0.02	9
Water Level	0.03	0.01	4	0.02	8

Figure 5: Experimental results.

The benchmarks can be solved by all considered implementations, namely HyTech, the straightforward implementation of hybrid automata (column *CLP/HA*), and the HHA implementation with CLP, within milliseconds. Fig. 5 shows the concrete run-time results. It reveals that the CLP/HA implementation of hybrid automata (middle column) performs quite well. The CLP/HHA implementation (last column) allows the briefest problem formulations because of the use of the abstract state machine, but with slightly longer run-times. Since in this approach the time points of performing discrete steps are not computed explicitly, it is susceptible for rounding errors. In order to guarantee termination of the CLP implementations, the search depth is fixed in advance: For the CLP/HA implementation, the number of iterations, i.e. the number of time points of discrete state changes, is bounded; for the CLP/HHA implementation, the number of continuous plus discrete steps is given. These limits are also listed in the table.

When comparing HyTech to the approach depicted in this paper, several issues have to be taken into consideration. The first issue concerns the expressiveness of the dynamical model. HyTech restricts the dynamical model to linear hybrid automata in which the continuous dynamics is governed by differential equations. The nonlinear dynamics e.g. of the form $\dot{x} \bowtie c1 * x + c2$, where $c1, c2 \in \mathbb{R}, c1 \neq 0, \bowtie \in \{<, \leq, >, \geq, =\}$ are firstly approximated either by a linear phase portrait or clock translation [24]. Then, the verification phase is done on the approximated model. CLP, on the other hand, is more expressive, because it allows more general dynamics. In particular, CLP can directly handle dynamics expressible as a combination of polynomials, exponentials, and logarithmic functions explicitly without approximating the model. For instance the last equation can be represented in CLP form as $X \$\bowtie X0 - c2/c1 + c2/c1 * exp(c1 * (T - T0))$, where $(T - T0)$ is the computational delay. Although clearly completeness cannot be guaranteed, from a practical point of view, this procedure allows to express problems in a natural manner. The CLP technology can be fully exploited; it suspends such complex goals until they become solvable. Recall that decidability of model checking is given only for limited classes of hybrid systems.

Another issue that should be taken into account is the type of verifiable properties. HyTech cannot verify simple properties that depend on the occurrence of events, despite of the fact that synchronization events are used in the model. On the other hand, simple real-time duration properties between events can be verified using HyTech. However, to do so, the model must be specified by introducing auxiliary variables to measure delays between events or the delay needed for a particular conditions to be hold. Bounded response time and minimal event sep-

aration are further properties that can be verified using HyTech. This properties, however, can only be checked after augmenting the model under consideration with what is called a *monitor* or *observer* automaton (cf. [21]), whose functionality is to observe the model without changing its behavior. It records the time as soon as some event occurs. Before the model is verified, the monitor automaton has to composed with the original model. As demonstrated in this paper (Sect. 3), however, there is no need to augment the model with an extra automata for the reason that during the run, not only the state of variables are recorded, but also the events and the time, where the constraint solver can be used to reason about the respective properties.

4.2 Related Works

Using hybrid automata [20] is a well accepted method to model and analyze (mobile) multiagent systems [2, 3]. Hierarchical hybrid automata (HHA) can be used for building up and describing multi-layer control architectures based on physical motion dynamics of moving agents [9, 15]. In many applications they form a link between multi-robot systems and theories of hybrid systems as in [37]. CLP as a programming paradigm has already been applied to modeling hybrid systems including solving differential equations [26]. Several authors propose the explicit composition of different concurrent automata by hand leading to one single automaton, before a CLP implementation is applied. This is a tedious work, especially when the number of automata increases. The latter case is exemplified in [35, 29], where approach to model and analyze hybrid systems using CLP(R) [28] is introduced.

In [5], it is shown how reachability analysis for linear hybrid automata can be done by means of CLP, again by computing compositions of (simple) hybrid automata. Events are handled as constraints, which avoids some of the effort for computing composition, which leads to an exponential increase in the number of clauses in general. In our approach, however, we compute configurations of the overall system only if required.

In contrast to our approach, some authors approached modeling the behavior of hybrid systems as an automaton using CLP, but they do not handle a hybrid system consisting of different interacting hybrid automata. For example, [25] presents a hybrid system modeled as an automaton using CLP(F) [26], but neither handling concurrency nor hierarchies. Other authors employ CLP for implementing hybrid automata [10, 12, 17], but restrict attention to a simple class of hybrid systems (e.g. timed systems). They do not construct the overall behavior prior to modeling, but model each automaton separately. However, the run of the model takes all possible paths into consideration, resulting from the product of each component, which leads to unnecessary computation.

Another interesting approach on model checking hybrid systems is presented in [16]. There, an analysis technique is proposed that is able to derive verification conditions, i.e. constraints that hold in reachable states. These conditions are universally quantified and transformed into purely existentially quantified conditions, which is more suitable for constraint solving. For this, an implementation in Lisp is available employing a satisfiability modulo theories (SMT) solver, whereas the Prolog implementation proposed in this paper, allows to express discrete transitions explicitly and allows the use of several constraint solvers.

Another approach for verification of a hybrid systems is presented in [13]. In particular, the authors apply so-called bounded model checking (BMC) [8] to linear hybrid automata, by en-

coding them into predicative formulae suitable for BMC. For this reason, they developed a tool called HySAT that combines a SAT solver with linear programming, where the Boolean variables are used for encoding the discrete components, while real variables represent the continuous component. The linear programming routine is used to solve large conjunctive system of linear inequalities over reals, whereas the SAT solver is used to handle disjunctions. Similar to this approach, this paper has the essence of BMC. However, instead of checking the satisfiability of a formulae to some given finite depth k, we find the the set of reachable states and verify various properties on this set. In [8], neither concurrency nor hierarchy of hybrid automata is taken into consideration.

Differently to this paper, [30] introduces symbolic reachability analysis of lazy linear hybrid automata. They provided a verification technique based on bounded model checking and k-induction for reachability analysis. In their technique, SAT-based decision procedures are used to perform a symbolic analysis instead of an enumerative analysis. However, they did not show how the interacting concurrent components can be handled in their approach.

5 Conclusion

In this paper, we used CLP to model and to analyze hybrid systems composed of several interacting concurrent hybrid automata. We have proposed novel CLP implementations that model concurrent interacting hybrid automata in two steps. In the first step we presented, how to control the behavior of hybrid automata without explicitly composing the interacting automata by using constraints, while in the second step, we modeled hybrid automata at different levels of abstraction by making use of hierarchies among the participating automata. Both CLP implementation models are able not only to analyze hybrid systems, but also to handle the complexity that may raise due to the interacting parts of hybrid system. We illustrated the implementations by modeling a train gate controller system, and showed how several properties can be analyzed and proved.

As a future work, we plan to extend our CLP approaches to be used for symbolic model checking in conjunction e.g. with computational temporal logic. Additionally, solving more optimization problems will be taken into consideration in our approaches, too.

Acknowledgements

This research has been supported partly by the grant *Sto 421/2* from the German research foundation *DFG* within the special priority program 1125 on *Cooperating Teams of Mobile Robots in Dynamic Environments*. A shorter and preliminary version of this paper appeared as [32].

References

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In *ICAOS: International Conference on Analysis and Optimization of Systems – Discrete-Event Systems*, Lecture Notes

in Control and Information Sciences 1994, pages 331–351. Springer, Berlin, Heidelberg, New York, 1994.

[2] R. Alur, J. M. Esposito, M. Kim, V. Kumar, and I. Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In *World Congress on Formal Methods*, pages 212–232, 1999.

[3] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[4] K. R. Apt and M. Wallace. *Constraint Logic Programming Using Eclipse*. Cambridge University Press, Cambridge, UK, 2007.

[5] G. Banda and J. P. Gallagher. Analysis of linear hybrid systems in CLP. In M. Hanus, editor, *Pre-Proceedings of LOPSTR 2008 – 18th International Symposium on Logic-Based Program Synthesis and Transformation*, pages 58–72. Technical University of Valencia, Spain, 2008.

[6] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *Proceedings of 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems – Formal Methods for the Design of Real-Time Systems (SFM-RT)*, LNCS 3185, pages 200–236. Springer, Berlin, Heidelberg, New York, 2004.

[7] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, LNCS 3098, pages 87–124. Springer, Berlin, Heidelberg, New York, 2004.

[8] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, LNCS 1579, pages 193–207. Springer, Berlin, Heidelberg, New York, 1999.

[9] J. Borges de Sousa, K. H. Johansson, J. Silva, and A. Speranzon. A verified hierarchical control architecture for coordinated multi-vehicle operations. *International Journal of Adaptive Control and Signal Processing*, 21(2-3):159–188, 2007. Special issue on autonomous adaptive control of vehicles.

[10] A. Ciarlini and T. Frühwirth. Automatic derivation of meaningful experiments for hybrid systems. *Proceeding of ACM SIGSIM Conf. on Artificial Intelligence, Simulation, and Planning (AIS'00)*, 2000.

[11] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer, Berlin, Heidelberg, New York, 4th edition, 1994.

[12] G. Delzanno and A. Podelski. Model checking in CLP. In *Proceedings of 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, LNCS 1579, pages 223–239. Springer, Berlin, Heidelberg, New York, 1999.

[13] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[14] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, Proceedings*, LNCS 3414, pages 258–273. Springer, Berlin, Heidelberg, New York, 2005.

[15] U. Furbach, J. Murray, F. Schmidsberger, and F. Stolzenburg. Hybrid multiagent systems with timed synchronization – specification and model checking. In M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, editors, *Post-Proceedings of 5th International Workshop on Programming Multi-Agent Systems at 6th International Joint Conference on Autonomous Agents & Multi-Agent Systems*, LNAI 4908, pages 205–220. Springer, 2008.

[16] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In J.-F. Raskin and P. S. Thiagarajan, editors, *Proceedings of 20th International Conference on Computer Aided Verification (CAV 2008)*, LNCS 5123, pages 190–203, Princeton, NJ, 2008. Springer, Berlin, Heidelberg, New York.

[17] G. Gupta and E. Pontelli. A constraint-based approach for specification and verification of real-time systems. *Proceedings of IEEE Real-time Symposium*, pages 230–239, 1997.

[18] N. Halbwachs, Y. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Static Analysis – Proceedings of 1st International Static Analysis Symposium (SAS'94)*, LNCS 864, pages 223–223, Namur, Belgium, 1994. Springer, Berlin, Heidelberg, New York.

[19] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[20] T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, NJ, 1996. IEEE Computer Society Press.

[21] T. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 1019, pages 41–71. Springer, Berlin, Heidelberg, New York, 1995.

[22] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's Decidable about Hybrid Automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[23] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The Next Generation. In *IEEE Real-Time Systems Symposium*, pages 56–65, 1995.

[24] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[25] T. J. Hickey and D. K. Wittenberg. Rigorous modeling of hybrid systems using interval arithmetic constraints. In R. Alur and G. J. Pappas, editors, *Proceedings of 7th International Workshop on Hybrid Systems: Computation and Control (HSCC 2004)*, LNCS 2993, pages 402–416, Philadelphia, PA, USA, 2004. Springer, Berlin Heidelberg, New York.

[26] T. J. Hickey and D. K. Wittenberg. Using analytic CLP to model and analyze hybrid systems. In *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 2004.

[27] ILOG. *CPLEX 10.0, User's Manual*, 2006.

[28] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[29] J. Jaffar, A. Santosa, and R. Voicu. A clp proof method for timed automata. *Real-Time Systems Symposium, IEEE International*, 0:175–186, 2004.

[30] S. Jha, B. A. Brady, and S. A. Seshia. Symbolic reachability analysis of lazy linear hybrid automata. In J.-F. Raskin and P. S. Thiagarajan, editors, *Proceedings of 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2007)*, LNCS 4763, pages 241–256, Salzburg, Austria, 2007. Springer, Berlin, Heidelberg, New York.

[31] K. Marriott and P. J. Stuckey. *Programming with Constraints*. MIT Press, Cambridge, MA, London, 1998.

[32] A. Mohammed and F. Stolzenburg. Implementing hierarchical hybrid automata using constraint logic programming. In S. Schwarz, editor, *Proceedings of 22nd Workshop on (Constraint) Logic Programming*, pages 60–71, Dresden, 2008. University Halle Wittenberg, Institute of Computer Science. Technical Report 2008/08.

[33] Object Management Group, Inc. *UML Version 2.1.2 (Infrastructure and Superstructure)*, November 2007.

[34] C. Reinl, F. Ruh, F. Stolzenburg, and O. von Stryk. Multi-robot systems optimization and analysis using MILP and CLP. In P. U. Lima, N. Vlassis, M. Spaan, and F. S. Melo, editors, *Workshop 1: Formal Models and Methods for Multi-Robot Systems at 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 11–16, Estoril, Portugal, 2008. International Foundation for Autonomous Agents and Multi-Agent Systems (IFAAMAS).

[35] L. Urbina. Analysis of hybrid systems in CLP(R). In *Proceedings of 2nd International Conference on Principles and Practice of Constraint Programming (CP'96)*, LNAI 1118, pages 451–467, 1996.

[36] J. Wielemaker. *SWI-Prolog 5.6 – Reference Manual*. University of Amsterdam, Amsterdam, The Netherlands, August 2008. Updated for version 5.6.59.

[37] S. Zelinski, T. J. Koo, and S. Sastry. Hybrid system design for formations of autonomous vehicles. In *Proceedings of 42nd IEEE Conference on Decision and Control*, volume 1, pages 1–6, 2003.

Bisher erschienen

Arbeitsberichte aus dem Fachbereich Informatik

(http://www.uni-koblenz.de/fb4/publikationen/arbeitsberichte)

Ammar Mohammed, Frieder Stolzenburg, Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 6/2009

Tobias Kippert, Anastasia Meletiadou, Rüdiger Grimm, Entwurf eines Common Criteria-Schutzprofils für Router zur Abwehr von Online-Überwachung, Arbeitsberichte aus dem Fachbereich Informatik 5/2009

Hannes Schwarz, Jürgen Ebert, Andreas Winter, Graph-based Traceability – A Comprehensive Approach. Arbeitsberichte aus dem Fachbereich Informatik 4/2009

Anastasia Meletiadou, Simone Müller, Rüdiger Grimm, Anforderungsanalyse für Risk-Management-Informationssysteme (RMIS), Arbeitsberichte aus dem Fachbereich Informatik 3/2009

Ansgar Scherp, Thomas Franz, Carsten Saathoff, Steffen Staab, A Model of Events based on a Foundational Ontology, Arbeitsberichte aus dem Fachbereich Informatik 2/2009

Frank Bohdanovicz, Harald Dickel, Christoph Steigner, Avoidance of Routing Loops, Arbeitsberichte aus dem Fachbereich Informatik 1/2009

Stefan Ameling, Stephan Wirth, Dietrich Paulus, Methods for Polyp Detection in Colonoscopy Videos: A Review, Arbeitsberichte aus dem Fachbereich Informatik 14/2008

Tassilo Horn, Jürgen Ebert, Ein Referenzschema für die Sprachen der IEC 61131-3, Arbeitsberichte aus dem Fachbereich Informatik 13/2008

Thomas Franz, Ansgar Scherp, Steffen Staab, Does a Semantic Web Facilitate Your Daily Tasks?, Arbeitsberichte aus dem Fachbereich Informatik 12/2008

Norbert Frick, Künftige Anfordeungen an ERP-Systeme: Deutsche Anbieter im Fokus, Arbeitsberichte aus dem Fachbereicht Informatik 11/2008

Jürgen Ebert, Rüdiger Grimm, Alexander Hug, Lehramtsbezogene Bachelor- und Masterstudiengänge im Fach Informatik an der Universität Koblenz-Landau, Campus Koblenz, Arbeitsberichte aus dem Fachbereich Informatik 10/2008

Mario Schaarschmidt, Harald von Kortzfleisch, Social Networking Platforms as Creativity Fostering Systems: Research Model and Exploratory Study, Arbeitsberichte aus dem Fachbereich Informatik 9/2008

Bernhard Schueler, Sergej Sizov, Steffen Staab, Querying for Meta Knowledge, Arbeitsberichte aus dem Fachbereich Informatik 8/2008

Stefan Stein, Entwicklung einer Architektur für komplexe kontextbezogene Dienste im mobilen Umfeld, Arbeitsberichte aus dem Fachbereich Informatik 7/2008

Matthias Bohnen, Lina Brühl, Sebastian Bzdak, RoboCup 2008 Mixed Reality League Team Description, Arbeitsberichte aus dem Fachbereich Informatik 6/2008

Bernhard Beckert, Reiner Hähnle, Tests and Proofs: Papers Presented at the Second International Conference, TAP 2008, Prato, Italy, April 2008, Arbeitsberichte aus dem Fachbereich Informatik 5/2008

Klaas Dellschaft, Steffen Staab, Unterstützung und Dokumentation kollaborativer Entwurfs- und Entscheidungsprozesse, Arbeitsberichte aus dem Fachbereich Informatik 4/2008

Rüdiger Grimm: IT-Sicherheitsmodelle, Arbeitsberichte aus dem Fachbereich Informatik 3/2008

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik 2/2008

Markus Maron, Kevin Read, Michael Schulze: CAMPUS NEWS – Artificial Intelligence Methods Combined for an Intelligent Information Network, Arbeitsberichte aus dem Fachbereich Informatik 1/2008

Lutz Priese,Frank Schmitt, Patrick Sturm, Haojun Wang: BMBF-Verbundprojekt 3D-RETISEG Abschlussbericht des Labors Bilderkennen der Universität Koblenz-Landau, Arbeitsberichte aus dem Fachbereich Informatik 26/2007

Stephan Philippi, Alexander Pinl: Proceedings 14. Workshop 20.-21. September 2007 Algorithmen und Werkzeuge für Petrinetze, Arbeitsberichte aus dem Fachbereich Informatik 25/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS – an Intelligent Bluetooth-based Mobile Information Network, Arbeitsberichte aus dem Fachbereich Informatik 24/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS - an Information Network for Pervasive Universities, Arbeitsberichte aus dem Fachbereich Informatik 23/2007

Lutz Priese: Finite Automata on Unranked and Unordered DAGs Extented Version, Arbeitsberichte aus dem Fachbereich Informatik 22/2007

Mario Schaarschmidt, Harald F.O. von Kortzfleisch: Modularität als alternative Technologie- und Innovationsstrategie, Arbeitsberichte aus dem Fachbereich Informatik 21/2007

Kurt Lautenbach, Alexander Pinl: Probability Propagation Nets, Arbeitsberichte aus dem Fachbereich Informatik 20/2007

Rüdiger Grimm, Farid Mehr, Anastasia Meletiadou, Daniel Pähler, Ilka Uerz: SOA-Security, Arbeitsberichte aus dem Fachbereich Informatik 19/2007

Christoph Wernhard: Tableaux Between Proving, Projection and Compilation, Arbeitsberichte aus dem Fachbereich Informatik 18/2007

Ulrich Furbach, Claudia Obermaier: Knowledge Compilation for Description Logics, Arbeitsberichte aus dem Fachbereich Informatik 17/2007

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 16/2007

Rüdiger Grimm, Anastasia Meletiadou: Rollenbasierte Zugriffskontrolle (RBAC) im Gesundheitswesen, Arbeitsberichte aud dem Fachbereich Informatik 15/2007

Ulrich Furbach, Jan Murray, Falk Schmidsberger, Frieder Stolzenburg: Hybrid Multiagent Systems with Timed Synchronization-Specification and Model Checking, Arbeitsberichte aus dem Fachbereich Informatik 14/2007

Björn Pelzer, Christoph Wernhard: System Description:“E-KRHyper“, Arbeitsberichte aus dem Fachbereich Informatik, 13/2007

Ulrich Furbach, Peter Baumgartner, Björn Pelzer: Hyper Tableaux with Equality, Arbeitsberichte aus dem Fachbereich Informatik, 12/2007

Ulrich Furbach, Markus Maron, Kevin Read: Location based Informationsystems, Arbeitsberichte aus dem Fachbereich Informatik, 11/2007

Philipp Schaer, Marco Thum: State-of-the-Art: Interaktion in erweiterten Realitäten, Arbeitsberichte aus dem Fachbereich Informatik, 10/2007

Ulrich Furbach, Claudia Obermaier: Applications of Automated Reasoning, Arbeitsberichte aus dem Fachbereich Informatik, 9/2007

Jürgen Ebert, Kerstin Falkowski: A First Proposal for an Overall Structure of an Enhanced Reality Framework, Arbeitsberichte aus dem Fachbereich Informatik, 8/2007

Lutz Priese, Frank Schmitt, Paul Lemke: Automatische See-Through Kalibrierung, Arbeitsberichte aus dem Fachbereich Informatik, 7/2007

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, Marcel Weinand, Jörg Helbach: Security Requirements for Non-political Internet Voting, Arbeitsberichte aus dem Fachbereich Informatik, 6/2007

Daniel Bildhauer, Volker Riediger, Hannes Schwarz, Sascha Strauß, „grUML – Eine UML-basierte Modellierungssprache für T-Graphen“, Arbeitsberichte aus dem Fachbereich Informatik, 5/2007

Richard Arndt, Steffen Staab, Raphaël Troncy, Lynda Hardman: Adding Formal Semantics to MPEG-7: Designing a Well Founded Multimedia Ontology for the Web, Arbeitsberichte aus dem Fachbereich Informatik, 4/2007

Simon Schenk, Steffen Staab: Networked RDF Graphs, Arbeitsberichte aus dem Fachbereich Informatik, 3/2007

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik, 2/2007

Anastasia Meletiadou, J. Felix Hampe: Begriffsbestimmung und erwartete Trends im IT-Risk-Management, Arbeitsberichte aus dem Fachbereich Informatik, 1/2007

„Gelbe Reihe“
(http://www.uni-koblenz.de/fb4/publikationen/gelbereihe)

Lutz Priese: Some Examples of Semi-rational and Non-semi-rational DAG Languages. Extended Version, Fachberichte Informatik 3-2006

Kurt Lautenbach, Stephan Philippi, and Alexander Pinl: Bayesian Networks and Petri Nets, Fachberichte Informatik 2-2006

Rainer Gimnich and Andreas Winter: Workshop Software-Reengineering und Services, Fachberichte Informatik 1-2006

Kurt Lautenbach and Alexander Pinl: Probability Propagation in Petri Nets, Fachberichte Informatik 16-2005

Rainer Gimnich, Uwe Kaiser, and Andreas Winter: 2. Workshop "Reengineering Prozesse" – Software Migration, Fachberichte Informatik 15-2005

Jan Murray, Frieder Stolzenburg, and Toshiaki Arai: Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification, Fachberichte Informatik 14-2005

Reinhold Letz: FTP 2005 – Fifth International Workshop on First-Order Theorem Proving, Fachberichte Informatik 13-2005

Bernhard Beckert: TABLEAUX 2005 – Position Papers and Tutorial Descriptions, Fachberichte Informatik 12-2005

Dietrich Paulus and Detlev Droege: Mixed-reality as a challenge to image understanding and artificial intelligence, Fachberichte Informatik 11-2005

Jürgen Sauer: 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen, Fachberichte Informatik 10-2005

Pascal Hitzler, Carsten Lutz, and Gerd Stumme: Foundational Aspects of Ontologies, Fachberichte Informatik 9-2005

Joachim Baumeister and Dietmar Seipel: Knowledge Engineering and Software Engineering, Fachberichte Informatik 8-2005

Benno Stein and Sven Meier zu Eißen: Proceedings of the Second International Workshop on Text-Based Information Retrieval, Fachberichte Informatik 7-2005

Andreas Winter and Jürgen Ebert: Metamodel-driven Service Interoperability, Fachberichte Informatik 6-2005

Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaaki Kikuchi, and Minoru Asada: Getting closer: How Simulation and Humanoid League can benefit from each other, Fachberichte Informatik 5-2005

Torsten Gipp and Jürgen Ebert: Web Engineering does profit from a Functional Approach, Fachberichte Informatik 4-2005

Oliver Obst, Anita Maas, and Joschka Boedecker: HTN Planning for Flexible Coordination Of Multiagent Team Behavior, Fachberichte Informatik 3-2005

Andreas von Hessling, Thomas Kleemann, and Alex Sinner: Semantic User Profiles and their Applications in a Mobile Environment, Fachberichte Informatik 2-2005

Heni Ben Amor and Achim Rettinger: Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation, Fachberichte Informatik 1-2005