Towards Modeling of Hybrid Systems in Promela and Spin

Dragan Bošnački *
Computing Science Department
Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven
e-mail: dragan@win.tue.nl

Abstract

Promela is a specification language for modeling of concurrent systems. The models written in Promela can be simulated and validated by the software package Spin. We present an approach to the specification and validation of a class of discrete-time hybrid systems in Promela and Spin. The hybrid features are implemented using only Promela macro definitions, without any changes of Spin. Although this approach can be seen just as a prototype for a more efficient implementation of the validation on the level of Spin's C sources, some experimental results show that even at this stage it can be used for validation of some systems known in the literature. We also propose new extensions for Promela in order to comprise the hybrid characteristics of the systems. The extended Promela is intended to be an input language for the future versions of Spin capable to verify hybrid systems.

1 Introduction

Hybrid systems are dynamical systems that consist of a digital (discrete) component that interacts with a continuous (analog) environment [6, 7]. An often given example of a hybrid system is a program that controls some plant. The importance of studying such mixed systems grows with the application of embedded software and firmware for control of safety-critical systems, like air-plains, nuclear plants, etc.

*On leave from the Institute of Informatics, Faculty of Natural Sciences and Mathematics, University "Sts. Cyril and Methodius", Skopje, Macedonia. Supported by the project "Algebraic Structures and their Application in the Modeling of Communication and Other Processes" of the Ministry of Science of the Republic of Macedonia, and EC ESPIRIT LTR Project No. 23498 (VIRES).

Hybrid automata are mathematical models of hybrid systems, which are an extension of the finite state automata with continuous variables. In this paper we deal with a particular kind of hybrid systems which can be modeled with an important subclass of hybrid automata called discrete-time rectangular hybrid automata [7].

In this model, the analog variables evolve continuously with the time passage, but the discrete transitions between control locations can happen only in integer points of time. This is often a realistic modeling, because of the latency of the discrete part (controller) which is always present. The controller has some minimal time that needs to react, which can be best modeled with discrete-time hybrid automata.

Besides, unlike their dense time counterparts, it can be shown [7] that the discrete-time rectangular automata have finite bisimilarity quotient and can be executed symbolically, which guarantee finiteness of the correctness checks.

Promela is a high level modeling language for specification of concurrent systems. The models (programs) written in Promela are used as an input for *Spin* software package for their automated simulation and/or validation [10]. The standard distribution of Promela and Spin does not support timed and hybrid systems. Nevertheless, in the paper we describe how discrete-time rectangular automata can be modeled and verified in Promela and Spin. The modeling is done completely on a "high-level", using only the capabilities of Promela and without any change of Spin. This allows us to reuse all the simulation and verification possibilities that Spin offers.

In a pursuit of better efficiency, we plan to implement the ideas presented in this paper and for now realized only on high-level, by rewriting some parts of the Spin sources. For this purpose it will be convenient to have new syntactic extensions of Promela in order to be able to model the continuous system parts. We propose such hybrid extensions and give an informal semantics of the new features.

Paper outline: In the next section we define the basic notions and notation that will be used in this paper. Section 3 is similar in that regard and deals with the definitions and main results on discrete-time hybrid automata. The third section gives very brief overview of Promela and Spin. The next two sections are the core of the paper. In the implementation section, in a rather general way, we first give the underlying theoretical basis for the implementation and afterwards a brief discussion with examples of the concrete implementation in Promela. In Section 6 the new hybrid extensions of Promela are given, together with their informal semantics. In the last section we give a short overview of some experimental results and a usual conclusion with a discussion for the future work.

2 Preliminaries

We start by introducing some notions that will be used in this paper. The definitions are mainly taken from [7]. Some of them are adapted to the model-checking approach that we will use in our implementation - forward reachability instead of a backward one, which is preferred in [7].

We will give the semantics of the models in the terms of transition systems.

Definition 2.1 A *transition system* $S = (Q, \Sigma, \rightarrow, Q_I, \Pi, \models)$, consists of a set Q of *states*, a finite set Σ of *events*, a *transition relation* $\rightarrow \subseteq Q \times \Sigma \times Q$, a set $Q_I \subseteq Q$ of *initial states*, a set Π of *propositions*, and a *satisfaction relation* $\models \subseteq Q \times \Pi$. We will use $q \xrightarrow{\sigma} q'$ instead of $(q, a, q') \in \rightarrow$, and $q \models \pi$ instead of $(q, \pi) \in \models$. A *region* is a subset of Q. For a given proposition $\pi \in \Pi$ we will use $R_\pi = \{q \in Q \mid q \models \pi\}$ to denote the region that satisfies π.

The model-checking problem will be stated as a reachability problem, i.e. we answer the question whether, starting from the initial states, some undesired states are reached.

Definition 2.2 Let S be a transition system. For each event $\sigma \in \Sigma$, the *σ-successor operator* $Suc_\sigma : 2^Q \rightarrow 2^Q$ is defined as $Suc_\sigma(R) = \{q \in Q \mid \exists r \in R. r \xrightarrow{\sigma} q\}$, i.e. the set of states that can be reached from R through transitions labeled with σ. We define $Suc : 2^Q \rightarrow 2^Q$ as $Suc(R) = \bigcup_{\sigma \in \Sigma} Suc_\sigma(R)$, the set of all states that can be reached from R. A region $R \subseteq Q$ is *reachable* from Q_I if $R \cap Suc^k(Q_I) \neq \emptyset$ for some $k \in N$.

Definition 2.3 Let $\mathcal{C}$ be a class of transition systems. The safety verification problem for $\mathcal{C}$ is stated in the following way: given a transition system $S \in \mathcal{C}$ and a proposition $\pi \in \Pi$, determine whether the region R_π is *not* reachable in S.

Having dense variable domains means that we will have to deal with transition systems with infinite number of states. Therefore, in order to make the model-checking of such systems feasible, we will have to find a way to reduce the verification of the systems with infinite number of states, to a verification of finite transition systems. This can be solved by introducing the effectively presented transition systems.

Definition 2.4 A *symbolic execution theory* for the transition system S consists of a set $\mathcal{F}$ of *formulae*, a formula $\phi_I \in \mathcal{F}$, and a map $[\cdot] : \mathcal{F} \rightarrow 2^Q$ such that (1) for every proposition $\pi \in \Pi$ is a formula: $[\pi] = R_\pi$; (2) for all formulae: $[\phi_1 \wedge \phi_2] = [\phi_1] \cap [\phi_2]$, $[\phi_1 \vee \phi_2] = [\phi_1] \cup [\phi_2]$, and $[\neg\phi] = Q \backslash [\phi]$; (3) $[\phi_I] = Q_I$; (4) the set $\{\phi \in \mathcal{F} \mid [\phi] = \emptyset\}$ is recursive; and for each event $\sigma \in \Sigma$, there is a computable map $Suc_\sigma : \mathcal{F} \rightarrow \mathcal{F}$ such that $[Suc_\sigma(\phi)] = Suc_\sigma([\phi])$ for every formula $\phi \in \mathcal{F}$. An *effectively-presented transition system* consists of transition system S together with a symbolic execution theory for S.

Definition 2.5 A *bisimulation* on the transition system S is an equivalence relation $\cong$ on the state set Q such that (1) if $q \cong r$ then for all propositions $\pi \in \Pi$, it holds $q \models \pi$ iff $r \models \pi$, and (2) if $q \cong r$ and $q \xrightarrow{\sigma} q'$, then there is a state $r' \in Q$ such that $r \xrightarrow{\sigma} r'$ and $q' \cong r'$. We will denote the largest bisimulation with $\equiv$. The *bisimilarity quotient* $S/_\equiv$ is the transition system $(Q/_\equiv, \Sigma, \rightarrow_\exists, Q_\exists, \Pi, \models_\exists)$, where $R \in Q_\exists$ iff $R \cap Q_I \neq \emptyset$,

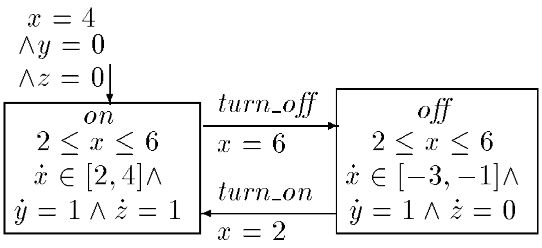

Figure 1: Rectangular automata for the thermostat example

$R \models \pi$ iff $R \cap R_I \neq \emptyset$, and $R \xrightarrow{\sigma}_{\exists} R'$ iff there exists two states $q \in R$ and $q' \in R'$ such that $q \xrightarrow{\sigma} q'$.

Theorem 2.6 *The safety verification problem is decidable for effectively-presented transition systems with finite bisimilarity quotients [6, 7].*

3 Hybrid Discrete-Time Rectangular Automata

The model of discrete-time rectangular automata which is used in this paper is mainly based on [7]. We recommend this reference for the complete formal definitions. Here we recall only some of them, after an informal description of the syntax and semantics of the rectangular automata using a simple example adapted from [9].

A rectangular automaton is a finite automaton extended with finite number of continuous variables in order to comprise the mixed discrete-continuous nature of the hybrid systems. The automaton is represented as a nondeterministic transition graph with nodes (also called *control modes*) labeled with special kind of predicates over the continuous variables called *rectangular predicates*. There are three kind of node labels: *invariants* that define the possible current values that the analog variables can have when system resides in the node, the *flow* predicates that constraint the behavior of the first derivatives of the continuous variables and *initial* condition predicates which define the initial values of the variables.

The edges, which are called *control switches* or *control transitions*, are also labeled by rectangular predicates called *jump conditions* that define for which values of the variables the automaton switches between the control modes and to which values the variables are possibly set after the jump. Additionally, the edges can be labeled with *events* which provide synchronization with the other automata in the system.

On Figure 1 there is an example of rectangular automaton that models a simple thermostat.

Initially, the automaton starts in node *on* with initial temperature x of 4 degrees and starts rising with a flow rate between 2 and 4 degrees per time unit. The variables y and z are clocks measuring the total time since the automaton started its operation, or the total amount of time spent in the mode *on*, respectively. In the *on* node they increase with a constant rate 1.

The automaton can stay in this mode only when x is in the interval $[2, 6]$. When the temperature reaches the value 6 the automaton switches to a new control mode where the temperature decreases with a rate between 1 and 3 degrees per time unit, z stays unchanged in this mode, while y increases constantly, like in the previous mode. When x drops back to 2 degrees than the automaton jumps back to the *on* mode and the previous invariants and flows become valid again.

Definition 3.1 Let $X = \{x_1, \ldots, x_n\}$ denote a set of real-valued variables. A formula of the form $x_i \sim c$, where c is an integer constant and $\sim \in \{<, \leq, =, \geq, >\}$ is called a *rectangular inequality.* A *rectangular predicate* over X is a conjunction of rectangular inequalities. With the rectangular predicate ϕ a set of vectors, denoted as $[\phi] = \{\mathbf{y} \in \mathbb{R}^n \mid \phi[X := y] \text{ is true}\}$, is defined, i.e. a set of variable valuations that satisfy ϕ. Such a set of vectors that satisfy a rectangular predicate is called a *rectangle.* Given a positive integer m, the rectangular predicate ϕ and the rectangle $[\phi]$ are *m-definable* if $\mid c \mid \leq m$ for every conjunct $x_i \sim c$ of ϕ. We will denote the set of all rectangular predicates over X with $Rect(X)$.

Definition 3.2 A *rectangular hybrid automaton* A is a tuple

$$(X, V, E, inv, init, jump, flow, \Sigma, event)$$

where:
(1) $X = \{x_1, \ldots, x_n\}$ is a set of real-valued *variables* corresponding to the continuous part of the system. We say that A has dimension n. We denote with $\dot{X}$ the set $\{\dot{x_i} \mid x_i \in X\}$ of dotted variables, and with X' the set $\{x'_i \mid x_i \in X\}$ of primed variables.
(2) The pair (V, E) is finite directed multigraph that corresponds to the discrete part of the system. We will call the vertices in V *control modes*, while the edges in E we will call *control switches.* For a control switch $(v, v') \in E$ the control modes v and v' are called *source* and *target*, respectively.
(3) $inv : V \rightarrow Rect(X)$ is a function mapping each control mode to its *invariant condition*, which is a rectangular predicate.
(4) $jump$ is a function mapping each control switch $e \in E$ into a predicate $jump(e)$ of the form $\phi \wedge \phi' \wedge \bigwedge_{i \notin update(e)} (x'_i = x_i)$, where $\phi \in Rect(X)$ and $\phi' \in Rect(X')$ are rectangular predicates, and $update(e) \subseteq \{1, \ldots, n\}$. We also call $jump(e)$ a *jump condition* and it specifies under which conditions the control transfer happens and how the variables change when the control mode changes. The unprimed variables x_i denote the values before the jump, while the primed variables x'_i refer to the corresponding value after the jump. We will call ϕ (the part of the $jump(e)$ which is a conjunction of inequalities only on unprimed variables) a *guard* or *enabling condition* for the control switch.
(5) $flow : V \rightarrow Rect(\dot{X})$ is a function that maps each control mode v to its *flow condition.* The flow condition is a rectangular predicate on dotted variables which represents

the constraints on the first derivatives of the variables with regard to time when the control of the system is in v.
(6) Σ is a finite set of *events*, and $event : E \rightarrow \Sigma$ is a function labeling each control switch with an event.

We say that the automaton A is *m-definable* if every rectangular predicate in the definition of A is *m-definable*. If for each control mode $v \in V$ the invariant rectangle $[inv]$ and flow rectangle $[flow]$ are subsets of the positive orthant $\mathbb{R}^n_{\geq 0}$ then the automaton A is *positive*. The automaton A is bounded if for every control mode $v \in V$, the invariant rectangle $[inv]$ is a bounded set. □

We will model our systems as parallel composition of hybrid automata which coordinate their operation via synchronization labels and/or shared variables. We deal with a usual notion of composition (c.f. [1]) where the product automaton is obtained as a Cartesian product of locations with conjunction of invariants, flow and jump predicates.

Definition 3.3 Let $A_1 = (X_1, V_1, E_1, inv_1, init_1, flow_1, \Sigma_1, event_1)$ and $A_2 = (X_2, V_2, E_2, inv_2, init_2, flow_2, \Sigma_2, event_2)$ be two hybrid automata of dimensions n_1 and n_2, respectively. The *product automaton* $A_1 \times A_2$ of A_1 and A_2 is the automaton $A = (X_1 \cup X_2, V_1 \times V_2, E, inv, init, flow, \Sigma_1 \cup \Sigma_2, event)$, where:

(1) With each control mode $(v_1, v_2) \in V_1 \times V_2)$ the initial condition $init(v_1, v_2) = init(v_1) \wedge init(v_2)$, the invariant $inv(v_1, v_2) = inv(v_1) \wedge inv(v_2)$, and the flow condition $flow(v_1, v_2) = flow(v_1) \wedge flow(v_2)$ are associated.

(2) The control switch $((v_1, v_2), (v_1', v_2'))$ is in E iff (i) $v_2 = v_2'$ and there is a control switch $e_1 = (v_1, v_1') \in E_1$ and $event(e_1) \notin \Sigma_2$; or (ii) $v_1 = v_1'$ and there is a control switch $e_2 = (v_2, v_2') \in E_2$ and $event(e_2) \notin \Sigma_1$; or (iii) there is a transition $e_1 = (v_1, v_1') \in E_1$ and a transition $e_2 = (v_2, v_2') \in E_2$ such that $event(e1) = event(e2)$.

In case (i) $jump(e) = jump_2(e_2)$ and $event(e) = event_2(e_2)$; in case (ii) $jump(e) = jump_1(e_1)$ and $event(e) = event_1(e_1)$; in case (iii) with $jump_j(e_j) = \phi_j \wedge \phi_j' \wedge \bigwedge_{i \notin update_j(e)} (x_i' = x_i)$, for $j = 1, 2$, and the additional requirement $update_1(e_1) \cap update_2(e_2) = \emptyset$ then $jump(e) = \phi_1 \wedge \phi_2 \wedge \phi_1' \wedge \phi_2' \wedge \bigwedge_{i \notin update(e)} (x_i' = x_i)$, where $update(e) = update_1(e_1) \cup update_2(e_2)$.

We describe the semantics of hybrid automata by means of timed transition systems. The states of the transition system consist of a *control mode (location)* that represents the discrete part, and vector (valuation) for the variables, which corresponds to the continuous part.

Definition 3.4 A *state* of a rectangular automaton A is a pair $(v, \mathbf{y})$, where $v \in V$ and $\mathbf{y} \in [inv(v)]$ is a vector that satisfies the invariant condition of v. Thus, the set of states can be defined as $Q = \{(v, \mathbf{y}) \in V \times \mathbb{R}^n \mid \mathbf{y} \in [inv(v)]\}$. A *region* of A is any subset of Q. A *rectangular state predicate* for A is a function $\psi : V \rightarrow Rect(X)$.

The region $[\psi] = \{(v, \mathbf{y}) \in Q \mid \mathbf{y} \in [\psi]\}$ defined by a rectangular state predicate ψ is called *rectangular region*. The function *init* defines the rectangular region $Q_I = [init]$ of *initial states*.

There are two types of transitions that a rectangular automaton can perform: discrete jump transitions which explicitly correspond to the control transitions (switches) and "implicit" time transitions that represent the time flow and the evolution of the variables. Being in discrete-time framework, we will consider only time transitions with fixed duration of one time unit, which implies a discrete-time transition system.

Definition 3.5 Let A be a *discrete-time* rectangular automaton. For each event $e \in \Sigma$ the *jump relation* $\stackrel{\sigma}{\rightarrow} \subseteq Q^2$ is defined by $(v, \mathbf{y}) \stackrel{\sigma}{\rightarrow} (v', \mathbf{y}')$ iff there exists a control switch $e = (v, v') \in E$ such that $event(e) = \sigma$ and $(\mathbf{y}, \mathbf{y}') \in [jump(e)]$.
The *flow relation* $\stackrel{\delta}{\rightarrow} \subseteq Q^2$ is defined as: $(v, \mathbf{y}) \stackrel{\delta}{\rightarrow} (v, \mathbf{y})$ iff (1) $v = v'$, and (2) $\frac{\mathbf{y}' - \mathbf{y}}{\delta} \in [flow(v)]$.

Given a control switch e and a state $(v, \mathbf{y})$, we say that e is enabled in $(v, \mathbf{y})$ if $\mathbf{y} \in \phi$, where ϕ is the enabling condition of e.

By taking $\delta = 1$ we obtain a discrete-time rectangular automaton, or more precisely its underlying discrete-time transition system. (The definition of a dense-time rectangular automata can be found in [7].)

Definition 3.6 Let A be a rectangular automaton. Then, the *discrete-time transition system* of A is $S_A^{disc} = (Q, \Sigma \cup \{1\}, \rightarrow, Q_I, \Pi, \models)$ where Π is the set of rectangular state predicates for A, and for all states $(v, \mathbf{y}) \in Q)$, $(v, \mathbf{y}) \models \pi$ iff $(v, \mathbf{y}) \in [\pi]$.

As we already mentioned in the Preliminaries section, we are trying to solve the problem of the infinite number of states in the discrete-time transition system S_A^{disc} that corresponds to the rectangular automaton A, by a "discretization" of S_A^{disc} via a bisimulation equivalence. We describe a bisimulation [7] on the states of S_A^{disc}, for which it can be shown that, provided A is positive or bounded rectangular automaton, induces a finite bisimilarity quotient transition system. Also, the bisimulation will provide a symbolic execution theory for S_A^{disc} that is necessary for effective presentation.

In discrete time we consider two states of the rectangular automaton to be bisimilar if (1) they have the same control mode, (2) the integer parts of the values of the corresponding variables are equal, and (3) the fractional part of the corresponding variables are both 0 or both different from 0.

The idea is that an *m-definable* rectangular automaton is positive it cannot distinguish variable values greater than m. In the case of positive rectangular automata the variables are only growing and never decreasing, and once a variable is over this threshold m its particular value (or more precisely, interval of values) is not important anymore, because the largest constant that can occur in the inequalities is greater or equal than m. Similarly if an *m-definable* automaton is bounded its variable vectors range over the cube $[-m, m]^n$. It is easy to see that in both cases the bisimilarity quotient is finite.

We formalize the above considerations in the following definition and corollaries from [7].

Definition 3.7 We define the equivalence relation $\approx_n$ on $\mathbb{R}^n$ by $\mathbf{y} \approx_n \mathbf{z}$ iff $\lfloor y_i \rfloor = \lfloor z_i \rfloor$ and $\lceil y_i \rceil = \lceil z_i \rceil$ for all $1 \leq i \leq n$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the usual floor and ceiling operators on real numbers. Given $m \in \mathbb{N}_{>0}$, define the equivalence relation $\approx_n^m$ on $\mathbb{R}^n$ by $\mathbf{y} \approx_n^m \mathbf{z}$ iff for each $1 \leq i \leq n$, either $y_i \approx_1 z_i$, or both y_i and z_i are greater than m, or both y_i and z_i are less then $-m$. For an n-dimensional rectangular automaton the equivalence relations $\cong_A$ and $\cong_A^m$ on the states of A are defined as: $(v, \mathbf{y}) \cong_A (w, \mathbf{z})$ iff $v = w$ and $\mathbf{y} \approx_n \mathbf{z}$, and $(v, \mathbf{y}) \cong_A^m (w, \mathbf{z})$ iff $v = w$ and $\mathbf{y} \approx_n^m \mathbf{z}$.

Theorem 3.8 *Let A be an n-dimensional rectangular automaton with k control modes. The equivalence relation $\cong_A$ is a bisimulation on the discrete-time transition system S_{disc}^A. If A is m-definable and either positive or bounded, then $\cong_A^m$ is also a bisimulation on S_{disc}^A.*

Corollary 3.9 *For every rectangular automaton A, the boolean combinations of rectangular state predicates for A form a symbolic execution theory for the discrete-time transition system S_{disc}^A.*

Corollary 3.10 *For positive or bounded rectangular automata, the discrete-time safety verification problem can be solved in PSPACE time.*

4 Spin, Xspin and Promela

Originally, Promela and Spin have been developed for analysis and validation of communication protocols. The language syntax is derived form C, but also uses the denotations for communications from Hoare's CSP and control flow statements based on Dijkstra's guarded commands.

The full presentation of Promela and Spin is beyond the scope of this paper and we suggest [10] as a reference to the interested reader. Following [11] we only give a short overview of the language and validation tools.

In Promela, the system components are modeled as *processes* that can communicate via *channels* either by buffered message exchanges or rendez-vous operations, and also trough shared memory represented as global *variables*. The execution of actions is considered asynchronous and interleaved, which means that in every step only one *enabled* action is performed and without any additional assumptions of the relative speed of the process execution.

Given as an input a Promela model Spin can do random or interactive simulations or to generate a C program that performs a validation of the system by scanning the state space.

The validation with the compiled C program is done in separate phases. The one phase is validation of the *safety* properties (absence of deadlock, unspecified message receptions, invalid end states and assertions) trough a search of the state space. Verification of most of the *liveness* properties can be handled by detection of progress and acceptance cycles. The never-claims constructs are the most general way to verify the system by providing the possibility to express properties as linear-time temporal formulae. All kinds of errors are reported by saving the trace of the actions leading to an invalid state or cycle. The erroneous sequence can be repeated as a guided simulation.

The simplest way to do scanning of the state space is trough an exhaustive search. For large state spaces some more sophisticated methods are used as state-vector compressing, partial-order reduction and bit-state hashing. The last technique only gives approximate results in a sense that there is only a guarantee that some part of the state space is covered.

Xspin is a graphical interface for Spin.It provides an integrating windowing environment for writing Promela models and carrying out virtually all Spin functions in a user friendly manner. The outputs are displayed in various ways, among which are Message Sequence Charts.

5 Implementation of Hybrid Automata in Promela

The validation in Spin can be viewed as an exploration of all possible execution runs of the system, done by a depth-first search (DFS) of the state space of the underlying transition system. In our case we will deal with S_A^{disc} for which we know from Corollary 3.9 that it is an effectively presented transition system. As it was already mentioned in the Preliminaries section, we will avoid the problems with the infinite state space by considering only the symbolic executions of the system. The states generated during the DFS can be regarded as representations of the formulae of the symbolic execution theory for S_A^{disc}. In our case they will always be rectangular state predicates. An alternative view is to consider the states generated during DFS as interpretations of the formulae — i.e. regions. Since, in this case, instead of dealing with single states, we work with sets of states corresponding to unions of bisimulation equivalence classes of S_A^{disc}. [1] In what follows we will freely mix these two approaches, depending on which interpretation is more appropriate for the context. In fact, we will introduce soon a new third kind of state interpretation and denotation.

In the sequel we will use τ to denote the events from $\Sigma \cup \{1\}$.

Recall that the adequacy of using symboic executions for solving the reachability problem for effectively-presented transition systems is established by Theorem 2.2. (For more details see [6].) Intuitively, a symbolic execution is as an approximation

[1] In fact, the view of the DFS states as union of bisimulation classes called *blocks* is a third approach (applied in [6]) and is slightly different from the ones used in this paper. It uses more directly the connection between the original transition system and its bisimulation quotient.

of an execution in the original transition system. For instance, during the symbolic execution, for a given control mode we do not know the exact valuation of the variable vector, but we know the intervals (the rectangle) of values that the variables can have. Consequently, the trajectory of the system in the state space is replaced by a "tube" of trajectories.

The usage of DFS for the reachability is formally justified by a simple check that the following proposition is correct with regard to Definition 2.2:

Proposition 5.1 *Given a transition system S, a region R is reachable in S, iff there is a sequence of labels $\sigma_1, \ldots, \sigma_k$ such that*
$R \cap Suc_{\sigma_k}(Suc_{\sigma_{k-1}}(\ldots Suc_{\sigma_1}(Q_I)\ldots)) \neq \emptyset$*, for some $k \in N$.*

The exploration of the state space starts from the initial state $Q_I = [init]$ and for each state R generates its successors (in DFS order) by computing $Suc_\tau(R)$ for each τ such that $\xrightarrow{\tau}$, or more precisely, the control switch labeled with τ, is enabled for some $q \in R$.

Definition 5.2 A formula ϕ of the symbolic execution theory of S_A^{disc} is *single-mode* iff for any two states $(v, \mathbf{y}), (v', \mathbf{y}') \in [\phi]$ it holds $v = v'$. Consistently with our dual approach of treating the transition system S_A^{disc} by means of the formulae of the symbolic execution theory as well as their interpretaion into regions, we will also call the region $[\phi]$ *single-mode* region.

In other words, the formula and its corresponding region are single-mode if only one control mode is active, and the states belonging to the region differ only in their variable vector parts.

In the sequel we will assume that A has *just one initial control mode*, i.e., the function $init(v) = false$ for all but one $v \in V$, or in our new terminology, the rectangular state predicate $init$ is a single-mode formula. This is not a serious constraint because an automaton with several initial control modes can be implemented by introducing a dummy initial state from which the control nondeterministically switches to the original control modes. On the other hand this assumption will significantly simplify the generation of the state space.

Also whithout loss of generality, we will assume that all *control switches are event-deterministic* in a sense that for all $\sigma \in \Sigma$ and for all pairs of control modes $v, v' \in V$ there is *at most one* control switch from v to v' labeled with the event σ.

From Definitions 2.2 and 2.4, under assumption of event-deterministic control switches, one can see that if the formula ϕ_1 is single-mode, then also the formula $\phi_2 = Suc_\tau(\phi_1)$ is single-mode. This preservation feature of Suc_τ causes the property of only one control mode being active to be propagated from the initial state to all the states visited during the DFS.

Morover, note that $init(v)$ for the unique active node v is a rectangular predicate. This property of a rectangular predicate being assigned to the unique active node will also be invariant during the DFS in our implementation. This is again the case because, under determinacy of the control switches, Suc_τ is closed for rectangular state predicates.

Proposition 5.3 *If $init(v) \neq false$ for only one $v \in V$, then all states ϕ generated during the DFS are single-mode formulae. Moreover, ϕ is single-mode rectangular state predicate.*

Proof By a straightforward induction on the order the states are visited by DFS, using the preservation property of Suc_τ. □

A direct corollary from the last proposition is that for each state ψ, if $\psi(v) \neq false$, then $\psi(v) \in Rect(X)$.

Since, every symbolic state ψ generated during the DFS is a rectangular state predicate that can be seen as a pair (v, R), consisting of the control mode v that is the unique one for which $\psi(v) \neq false$ and a rectangular predicate $R = \psi(v)$. However, for our description and analysis of the implementation it is convenient to extend the symbolic state to a quadruple (v, R, F, I), where v and R are the same, and $I = inv(v)$ and $F = flow(v)$ are the invariant and flow predicate for the control mode v, respectively. The reason for this is that we are going to build the invariant and flow predicates on-the-fly during the DFS. This is not the case for the most of the verification tools, i.e., for their input languages, for which the invariants are usually defined for each location in advance in the program text. Nevertheless, we adopted our solution not only because of the implementational reasons, but also because it is more in the spirit of Promela.

In the sequel we will work with *closures* of rectangular predicates:

Definition 5.4 Given a rectangular predicate R over variables from X, a *closure* of R is a rectangular predicate $\bar{R}$ over variables from X which is equivalent to R, i.e. $[R] = [\bar{R}]$, and has a form

$$\bigwedge_{\forall x_i \in X} (x_i \sqsupseteq c_{i_1} \wedge x_i \sqsubseteq c_{i_2})$$

where the constants c_i range over the extended domain $\mathbb{Z} \cup \{-\infty, \infty\}$, and $\sqsubseteq \in \{<, \leq\}$ and $\sqsupseteq \in \{>, \geq\}$.

The closure can be regarded as a kind of canonical form for the rectangular predicates, containing for each variable exactly two inequalities that define its lower and upper bound. The closure of R is obtained by adding to R the missing conjuncts of the form $x_i > -\infty$ or $x_i < \infty$ such that the value of the predicate stays the same for all vectors $\mathbf{y} \in \mathbb{R}^n$. Also, in what follows $x_i = c_i$ will be regarded as an abbreviation of $x_i \leq c_i \wedge x_i \geq c_i$. In an analogous way we can define the closures of F and I. In the sequel we will work with the closures of R, F and I omitting the bar.

We proceed with the implementations of Suc_τ for the two kinds of transitions that occur in the system: control switch (or jump) transitions and time transitions with one time unit duration.

Definition 5.5 Given a rectangular predicate P over variables from X and a rectangular predicate ΔP over variables from X', let $P\{\Delta P\}$ denote the rectangular predicate obtained from P such that for each conjunct $x'_i \sqsupseteq' c'_i$ ($x'_i \sqsubseteq' c'_i$, respectively) that occurs in ΔP the corresponding inequality on the unprimed variable $x_i \sqsupseteq c_i$ ($x_i \sqsubseteq c_i$) in P is replaced by the the conjunct $x_i \sqsupseteq' c'_i$ ($x_i \sqsubseteq' c'_i$, resp.), where $\sqsupseteq \in \{\geq, >\}$ and $\sqsubseteq \in \{\leq, <\}$, and $\sqsupseteq$ and $\sqsupseteq'$ ($\sqsubseteq$ and $\sqsubseteq'$) do not have to match. Analogously the predicate $P\{\Delta P\}$ is defined when the inequalities of P and ΔP are over dotted variables.

Thus, the lower (upper) bounds from P are replaced with the corresponding lower (upper) bounds from ΔP.

The Suc_σ operator is a transition between two symbolic states during DFS. The following theorem give a formal basis for the construction of this transition.

Theorem 5.6 *For given symbolic state (v, R) and event σ, let e be the (unique) control switch transition enabled in (v, R) and labeled with σ, and with a target control mode v'. Further, let G and ΔR be the closures of the guard (enabling condition) and the conjunction of inequalities involving only primed variables, respectively, of the jump predicate $jump(e)$. Then, $Suc_\sigma(v, R) = (v', R')$, where $R' = (R \wedge G)\{\Delta R\}$.*

Proof That the new active mode is v' follows from the preservation of the single-mode property by Suc_τ. It remains to see that the resulting predicate R' is in accord with Definition 3.5. With a conjunction of R and G we register the fact that the transition can happen only for those variable vectors that satisfy the enabling condition. This conjunction represents an intersection of rectangular regions. The effect of the control switch is represented by the substitution of the lower and upper bounds of ΔR into the restricted region. $\Box$

Note that R' is again a rectangular predicate, which, having in mind that v' is the unique active mode, means that the state (v', R') corresponds again to a single-mode rectangular state predicate. The preservation of this property by Suc_σ was a requirement for the correctness of Property 5.3.

The transition corresponding to the Suc_σ operator is implemented by a compound Spin statement which general structure is $< G, \Delta R, \Delta F, \Delta I >$, where, as before, G corresponds to the guard of the predicate $jump(e)$ and ΔR, ΔF and ΔI are rectangular predicates such that

$$R' = (R \wedge G)\{\Delta R\},\ I' = I\{\Delta I\},\ \text{and}\ F' = F\{\Delta F\}$$

Abusing the notation we can write $(v, R, F, I) \stackrel{<G,\Delta R,\Delta F,\Delta I>}{\longrightarrow} (v', R', F', I')$. The three equations above give the procedure for finding the predicates of the successor state. The equation for R' is justified by Theorem 5.6. The new invariant and flow predicates are also obtained in a straightforward way by replacement of inequalities in order to obtain the invariant and flow conditions for the control mode of the successor state.

In what follows we will denote the lower and upper bounds of the variables from R, F and I with c_i^l, c_i^u, c_i^{flowl}, c_i^{flowu} and c_i^{invl}, c_i^{invu}, respectively. Further, for the sake of simpler presentation we will assume that the inequalities in the flow predicates contain only the operators $\leq$ and $\geq$. The case with strict inequalities is similar, only the obtained expressions are more complex.

Definition 5.7 Let A be an *m-definable* rectangular automaton. For a given rectangular closure predicate P over variables from X and a rectangular closure predicate Q over variables from $\dot{X}$, the rectangular predicate $P + Q$ is obtained in the following way: assuming that $\infty + c = \infty$ and $-\infty + c = -\infty$, for each conjunct $\dot{x}_i \sqsubseteq c_i^{flowu}$ from Q, the corresponding conjunct $x_i \sqsubseteq c_i^u$ in P is replaced either with
(1) $x_i \sqsubseteq c_i^u + c_i^{flowu}$, if $c_i^u + c_i^{flowu} \leq c_i^{invu}$, or
(2) $x_i \sqsubseteq c_i^{uinv}$, otherwise;
and for each conjunct $\dot{x}_i \sqsupseteq c_i^{flowl}$ from Q the corresponding conjunct $x_i \sqsupseteq c_i^l$ in P is replaced either with
(1) $x_i \sqsupseteq c_i^l + c_i^{flowl}$, if $c_i^l + c_i^{flowl} \geq c_i^{invl}$; or
(2) $x_i \sqsupseteq c_i^{invl}$, if $c_i^{invl} > c_i^l + c_i^{flowl}$; or
(3) $x_i > m$, if $c_i^l + c_i^{flowl} > m$.

The implementation of Suc_1 operator is based on the following

Theorem 5.8 *Given a symbolic state (v, R), for its time transition successor $(v', R') = Suc_1(v, R)$ it holds $v = v'$ and $R' = R + F$.*

Proof From Definition 3.5 it follows directly that the control mode stays unchanged during the time transition.

We have to show how the rectangle $[R]$ is indeed transformed into $[R']$ by Suc_1. We are going to consider the continuous part of the state one variable at a time.

Note first that the time transition is enabled if, for all i, $c_i^l + c_i^{flowl} \leq c_i^{invu}$ and $c_i^u + c_i^{flowu} \geq c_i^{invl}$. This is because it is obvious that for each variable x_i the lower bound must be always increased at least with the value of the lower bound of $\dot{x}_i$, but never above the limits given by the upper bound of the invariant. Similarly, the upper bound of x_i cannot go below the lower limit of the invariant.

Once having established the enabling conditions for Suc_1, we can take a look first how the upper bound of x_i is transformed. The case (1) concerning the upper bounds from the theorem above describes the "normal" situation when the lower limit of $\dot{x}_i$ is such that the lower bound of x_i stays within the invariant condition. The case (2) is included to prevent the impossible situation when the value of x_i goes outside the invariants boundaries.

The cases for the lower bounds are analogous, except for the case (3). It can occur only for positive rectangular automata, when, because of the positive derivative, after some time the variable can go beyond m. As we already saw, this means that the concrete interval of values for x_i is not important anymore, because for the symbolic execution it suffices to know that $m < x_i < \infty$. □

The time transition that corresponds to Suc_1 $(v, R) \xrightarrow{1} (v', R')$ where $v' = v$ is implemented in Spin by the statement-transition

$$(v, R, F, I) \xrightarrow{1} (v', R', F', I')$$

where

$$R' = R + F, v' = v, F' = F \text{ and } I' = I$$

.

In the second part of this section we give a brief overview of some details of the implementation.

The jump transitions between symbolic states that correspond to the edges in the rectangular automaton are implemented as sequences of Promela macro definitions glued together by Promela `atomic` statement. To each part of the compound statement $< G, \Delta R, \Delta F, \Delta I >$ corresponds a group of macros.

For instance, the control switch transition labeled as *turn_off*, from *on* state to *off* state of the thermostat automaton (Fig. 1) is implemented as

```
atomic{eqtst(x,6) -> eqint(x,6);
       gedset(x,-3); ledset(x,-1); eqdset(z,0)};
```

The guard of the control switch generates two macros: `eqtst(x,6)` and `eqint(x,6)`. The `eqtst(x,6)` tests whether x=6 lies in the the interval of possible current values of the variable `x`. If so, then the guard is satisfied and the transition is enabled. The next step is to record the fact that the transition is happening exactly when x=6. This is done by macro `eqint(x,6)` that, in accord with the procedure of computing the next state (Suc_σ), intersects the singleton interval [6,6] with the interval of the possible current values of x. The control switch does not change the variable values, thus there is no macro corresponding to the ΔR part. The ΔF part is realized by the next three macros. The macros `gedset(x,-3)` and `ledset(x,-1)` set the lower and upper bound, respectively, of the derivative of x, i.e., implement the conjuncts $\dot{x} \geq -3$ and $\dot{x} \leq -1$. The new derivative for z is set by `eqdset(z,0)` (corresponding to the conjunct $z = 0$). Like for ΔR, there is no part for ΔI because the invariant in the new control mode *off* is the old one from *on* control mode.

The whole translation is wrapped within an `atomic` clause to implement the fact that the transition is instantaneous and all rectangle resets are an indivisible operation.

```
#define p !eqpre(y,120) || gepre(z,0) && ltpre(z,80)

never {
do
:: skip
:: !(p) -> break
do;
}
```

Figure 2: Definition and never claim for the LTL formula $\Box(y = 120 \Rightarrow z < 80)$

The time transition is implemented by a special clocking procedure which competes with the ordinary control switch transitions providing that no invariant will be violated with the passage of one time unit.

The fact that the implementation is on high level using only the existing features in Promela means that we reduce the hybrid automaton to an ordinary untimed automaton (transition system). This ensures that for the rectangular automata we can check all kind of properties as for the untimed systems using standard Promela: the absence of deadlocks, non-progress and acceptance cycles, and, as the most general, linear-time temporal logic (LTL) formulae. The rectangular inequalities can be used in assertions also in the atomic propositions in LTL formulae (i.e. in the statements of never claims).

For instance, assuming that the time unit is 1 minute, we can check that during the first two hours of its operation the thermostat is on for less than 80 minutes, by adding to the model the definition and the never claim from Figure 2.

The macros `eqpre(z,120)`, `gepre(z,0)` and `ltpre(z,80)` implement the conditions $z = 120, z \geq 0$, and $z < 80$, respectively. The `never` process implements in a standard way the LTL formula $\Box p$. The presented example is a typical one showing how a reachability can be checked. Note that the `Thermostat` automaton is not bounded nor positive, but the never claim adds the constraints which makes the whole system, together with the never claim, to be bounded and therefore feasible for validation.

6 Hybrid Extensions of Promela

The modeling of the rectangular automata described in the previous section was done by hand-coding of the automata description in Promela, using macro definitions, as well as ordinary statements. The natural next step is to automate this encoding. For this purpose we will need an input language capable to include hybrid characteristics. Besides being a language for description of the rectangular automata, the new language can be considered as a new independent formalism for modeling a class of hybrid systems that can be represented as (a subclass of) rectangular automata.

$$
\begin{array}{rcl}
\mathit{stmnt} & ::= & \mathit{ordinary\_stmnt} \mid \mathit{hybrid\_stmnt} \\
\mathit{hybrid\_stmnt} & ::= & \texttt{when\{}\ C\ \texttt{\}}\ \mathit{edc\_ordinary\_stmnt}\ \texttt{set\{}\ R\ \texttt{\}}\ \texttt{dif\{}\ B\ \texttt{\}}\ \texttt{inv}\ \texttt{\{}\ R\ \texttt{\}} \\
C & ::= & \mathit{ineq}\ ,\ C \mid \mathit{ord\_boolexp}\ ,\ C \mid \texttt{true} \mid \texttt{asap} \\
R & ::= & \mathit{ineq}\ ,\ R \\
B & ::= & \mathit{ineqb}\ ,\ B \\
\mathit{ineq} & ::= & \mathit{analogvar}\ \mathit{op}\ \mathit{short} \\
\mathit{ineqb} & ::= & \mathit{analogvar}\ \mathit{opb}\ \mathit{short} \\
\mathit{analogvar} & ::= & \mathit{analog} \mid \mathit{analog}\texttt{[}\ \mathit{expr}\ \texttt{]} \\
\mathit{op} & ::= & \texttt{<} \mid \texttt{>} \mid \mathit{opb} \\
\mathit{opb} & ::= & \texttt{<=} \mid \texttt{>=} \mid \texttt{=}
\end{array}
$$

Figure 3: Hybrid syntax extensions of Promela

6.1 Syntax

We propose some syntax extensions of Promela to represent the new hybrid features, which we call *HyPromela* in the sequel. In the proposed extensions we follow the similar work of [13] for real (dense) time extensions of Promela and Spin, using also some features from HyTech (c.f.[9]).

In order to represent continuous variables we introduce in Promela a new type `analog`. The `analog` variables can be scalar or arrays and are declared globally like in the following examples:

```
analog x, level, steam[3];
```

The next step is to extend the statements for the hybrid aspects keeping as much as possible of the spirit and compatibility with the standard version of the language. We do this by adding to the ordinary statements new hybrid parts as a header and/or trailer roughly given with the grammar rules shown in Fig. 3:

The nonterminal symbol *edc_ordinary_stmnt* is either: *a*) a sequence of ordinary statements which is deterministically executable (like for Spin's `d_step`) and the first statement is enabled (executable), or *b*) a simple message passing statement. Similarly, *ord_boolexp* is an ordinary boolean expression in which only non-analog (discrete) variables occur; *analog* means any variable of type `analog` and they can occur in boolean expressions only in assertions and never claims, and only in rectangular inequalities. *short* is an integer constant of type `short`, or special predefined values -INF and INF, meaning negative and positive infinity [2] , respectively, and *analog* is a variable of type `analog`. List of rectangular inequalities in the hybrid parts are interpreted like conjunctions which define the corresponding rectangles for values, flows or invariants. The `set`, `dif` and `inv` part of the statement can be omitted.

For example, the following are legal hybrid statements

[2] The infinities are meant to be used only in `inv` part and it is likely that they will be omitted in the next versions

```
when {asap} skip set {p1 = 0} dif {t1 >= 2, t1 < 3}
when {w >= N_1, w <= N_2}  p_1!on inv {w>-INF, w <= N_2}
when {k >= 1 && k<3, steam <=5} skip set {z<2}
when {true} activate?on;
```

In order to preserve the conjunctions of rectangular predicates in hybrid automata compositions we impose some additional restrictions on the variable occurrences in the setting parts of hybrid statements. Let I_i (resp. D_i) is the set of variables that occur in the `inv` (resp. `dif`) parts of hybrid statements in the i-th process (automaton). Then we require $\bigcap_{\forall i} D_i = \emptyset$ and $\bigcap_{\forall i} I_i = \emptyset$ In other words, no two different processes are allowed to change the same variable in their `inv` and `dif` parts. This is the essential difference between the models that can be specified by HyPromela and the ones in discrete-time rectangular hybrid automata, which do not have this constraints.

On the `set` parts we have a requirement only in case of synchronized action of two processes, which in HyPromela can be achieved only via rendez-vous communication. In such a case the sets of the variables of the `set` parts of the two communicating statements should be disjoint, i.e. the statements are not allowed to act on the same variable. [3]

The analog variables can occur only in the hybrid parts of HyPromela statements, assertions and never claims. Thus, the ordinary statements cannot change variables of type `analog`.

As an illustration for the syntax of hybrid statements we present on Figure 4 a HyPromela model of the thermostat example, which we are also going to revisit in the semantics section.

6.2 Semantics

We give just an informal semantics of the HyPromela programs. Although the full formal treatment will have to wait for a forthcoming paper, our conjecture is that

[3]Needless to say that we are not very satisfied with this solution. Nevertheless, one can argue that the requirements on flow rates and synchronized actions are not very severe and they are even useful imposing some encapsulation discipline in writing HyPromela programs. It is still to be seen what would be the impact of the restrictions on invariants on the language expressiveness.

Perhaps there is another more attractive solution for the invariants - not to use them at all. Instead, we can use the concept of deadlines of [12], replacing the invariant conditions of the states with bounds on transition arcs. In our opinion this is a natural approach for a language like Promela which is transitionally oriented rather than state oriented. In favor of this possibility speaks also our implementation of the language which does not need the invariants to model the urgency of actions. It seems that the only problem is how to cope with the invariants denoting physical limits, (like water level becoming zero in leaking water tanks, for instance). Without invariants the analog values go beyond some limits for a given state, producing unrealistic results. One not very effective solution is to include those physical limits in every jump condition in order to prevent the unwanted "imaginary" edge transitions in the situation when a variable has obtained "impossible" values (normally prevented by invariants).

```
analog x, y, z;

proctype Thermostat()
{
   when{asap} skip
              set{x=2,y=0,z=0}
              dif{x>=2,x<=4,y=1}
              inv{x>=1,y<=3,y<=60};
   on:
   when{x=3} skip /*turn_off*/
             set{} dif{x>=1,x<=3,z=0} inv{};
   off:
   atomic{when{x=1} skip /*turn_on*/
             set{} dif{x>=2,x<=4,z=1} inv{};
             goto on;}
}

init()
{
   run Thermostat();
}
```

Figure 4: Thermostat model in HyPromela

the approach of [13] for real time extensions can straightforwardly be adjusted to apply for hybrid case: the formal semantics of the HyPromela programs can be given via discrete-time transition systems that can later be reduced to ordinary (untimed) transition systems (defining Spin) and eventually to Büchi automata. Of course, as an intermediate step between HyPromela and timed transition system one can always have hybrid rectangular automata. In fact, in this subsection we use the HyPromela models and their corresponding hybrid automata interchangeably, considering them semantically equivalent (although we have not formalized this equivalence).

There are two kind of transitions in the underlying discrete-time transition system for a HyPromela program. The jump transitions are given explicitly by the hybrid statements [4].

A hybrid HyPromela statement is executable when the condition given by the list of items in the `when` clause is satisfied. The list of items (inequalities or Boolean expressions) in the curly brackets is interpreted as a conjunction of rectangular predicates over analog variables and boolean expressions on non-analog variables. Like in RT Promela and HyTech, the disjunction of rectangular inequalities can be realized by a nondeterministic branching statement.

The execution of the statements is considered instantaneous and this is the main reason for the requirement that the (sequence of) ordinary statement(s) which can be listed after `when` must not be delayable. The analog variables are affected only by `set`, `dif` and `inv` parts. The `set` part sets the boundaries for the analog values, in `dif` the flow rectangle is set, while `inv` defines the invariant rectangle. All these settings are "differential" in a sense that they only affect the boundaries of the analog variables mentioned in the lists - the limits for the other variables remain unchanged.

The series of settings defined in hybrid parts, as well as the ordinary statements that belong to a same hybrid statement, are considered as atomic and cannot be interrupted by any other statement. Also, in the case of rendez-vous communication the hybrid parts of both statements are executed atomically and they cannot interfere because of the conditions we imposed on the occurrences of the analog variables in the hybrid parts.

The executions of the hybrid statements are interleaved and asynchronous, except for the rendez-vous communications which are synchronous. [5]

The ordinary statements without any hybrid additives as well as the hybrid statements starting with `when {asap}`, are interpreted as executable as soon as possible (analogously to `asap` condition in HyTech), while `when {true}` construct represents a unboundedly delayable action which can be executed in any time slice.

Unlike the edge transitions that do not take time and correspond to hybrid statements in the program text, the other kind of transitions -the "invisible" time transitions

[4] We do not consider separately the ordinary statements because they can be treated as asap with empty inequality lists in `set`, `dif` and `inv`.

[5] Another exception is the execution of the monitoring `never` process, which is synchronized with all the other processes of the system

```
#define p !(y=60) || (z<40)

never {
do
:: skip
:: !(p) -> break
do;
}
```

Figure 5: Definition and never claim for the LTL formula $\Box(y = 60 \Rightarrow z < 40)$

- are not explicitly associated to any statement. They happen by letting the time elapse for one unit which causes analog variables to change their values continuously according to the flow rates and limited by the invariants.

For example, consider the HyPromela model of the thermostat from Figure 4. After the declaration of the analog variables, the proctype Thermostat is given. It starts by a hybrid action which initializes the values of the analog variables as well as the flow and invariant bounds. Speaking in terms of the hybrid automaton from Figure 1, with this initialization statement the system is put in the *on* state.

The rest of the process is an endless iteration [6] consisting of two hybrid statements that happen interchangeably, causing the system to bounce between *on* and *off* state. The first statement becomes active when `x` rises to the value 6. This will cause a transition to a new control mode with the same invariant (the `set` list is empty), but with different flow rates for `x` and `z`. Similarly, the transition back to the *on* control mode, that happens when `x` drops to 2, does not change the invariant, but has to restore the flow and invariant rectangles.

The `Thermostat` process is run in a standard way from `init`.

6.3 Verification

On the HyPromela models we can perform all the types of validations as for standard Promela: the absence of deadlocks, non-progress and acceptance cycles, LTL formulae. The rectangular inequalities can be used in assertions also in the atomic propositions in LTL formulae (i.e. in the statements of never claims).

On Figure 5 we revisit the verification example from the previous section. Except for the macro definition of p, i.e. the inequalities, the code is almost the same.

[6] For easier comparison with the automaton model we use `if` and `goto` to realize the loop, but, of course we could have equally well used Promela's `do` statement

7 Some Results, Conclusions and Future Work

The presented implementation of the discrete-time rectangular automata is intended to be just a first step towards a "low level" implementation of the discrete-time rectangular automata by changing the source of the Spin validator, so that the new hybrid features can be added.

Nevertheless it is encouraging that even at this stage the prototype could be used for verification of smaller systems. For instance, the simpler version of the steam-boiler control-system from [8], often suggested as a benchmark test for hybrid system formalisms, can be verified in 5 seconds with less then 3.5 MB memory. This result is comparable to the one reported for the same system (and very similar model) 3.0 MB of memory, using HyTech. Using a new option of the recent releases of Spin (3.0 and above) called minimized aotomaton the memory usage can be reduced up to 1.8 MB on the expense of increased validation time (6 min). The implementation was applied to some other classical examples like two water tanks, pursuit games, the thermostat example of this paper, etc. with similar results.

As an encouraging fact can also be considered the verification of Bounded Retransmission Protocol [4, 5], an industrial protocol used by Philips, done with less then 6 MB memory consumption for realistic parameters, using an implementation of discrete timing in Spin that is essentially a version of the implementation described in this paper, only optimized for clocks with constant flow rates. [3], [2].

Nevertheless, the real challenge remains the implementation of the ideas presented above on the level of Spin, which can result in a significant percentage of state space reduction. Recent implementations made by the author with discrete time in Spin prove that this reduction can be up to 40%. For instance, with low level implementation we can avoid the generation of new states even when nothing happens after a time transition, i.e., in the case when we have several "idle" time transitions between the edge actions. These kind of transitions can effectively be reduced to just one transition with a duration of several time units, with no extra states in between.

Maybe the most significant gain from the low-level implementation should be partial order reduction which is one of the most powerful techniques used against state space explosion. On the high level there is almost no reduction because all analog variables in our implementation must be global in order to be visible for the timing procedure. Again, the recent low-level prototypes of discrete-time extensions for Spin show that the number of states and transitions can be reduced in some cases to 20% of the original one.

Acknowledgments: The author would like to thank for their answers and suggestions: Dennis Dams, Thomas A. Henzinger and Stavros Tripakis.

References

[1] Alur, R., Henzinger, T. A., Ho, P.-H., *Automatic Symbolic Verification of Embedded Systems*, IEEE Transactions of Software Engineering, vol. 22, 1996, pp. 181-201

[2] Bošnački, D., *Implementing Discrete-Time in Promela and Spin*, Proceedings of LIRA'97, Novi Sad University, Yugoslavia, 1997

[3] Dams, D., Gerth, R., *Bounded Retransmission Protocol Revisited*, Second International Workshop on Verification of Infinite Systems, Infinity, 1997

[4] D'Argenio, P. R., Katoen, J.-P., Ruys, T., Tretmans, J., *The Bounded Retransmission Protocol Must Be on Time!*, TACAS'97, 1997

[5] Groote, J. F., van de Pol, J., *A Bounded Retransmission Protocol for Large Data Packets*, in Wirsing, M., Nivat, M., ed., Algebraic Methodology and Software Technology, LCNS 1101, pp. 536-550, Springer-Verlag, 1996

[6] Henzinger, T. A., *Hybrid Automata with Finite Bisimulations*, Proceedings of the 22th International Colloquium on Automata, Languages and Programming (ICALP 1995), LNCS 944, pp. 324-335, Springer-Verlag, 1995

[7] Henzinger, T. A., Kopke, P. W., *Discrete-Time Control for Rectangular Automata*, Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP 1997), LNCS 1256, pp. 582-593, Springer-Verlag, 1997

[8] Henzinger, T. A., Wang-Toi, H., *Using HyTech to Synthesize Control Parameters for a Steam Boiler*, In Formal Methods for Industrial Application: Specifying and Programming the Steam Boiler Control, LNCS 1165, pp. 265-282, Springer-Verlag, 1996

[9] Henzinger, T. A., Ho, P.-H., Wang-Toi, H., *HyTech: A Model Checker for Hybrid Systems*, Proceedings of the 9th International Conference on Computer-aided Verification (CAV 1997), LNCS 1254, pp. 460-463, Springer-Verlag, 1997

[10] Holzmann, G. J., *Design and Validation of Communication Protocols*, Prentice Hall, 1991. Also: `http://netlib.bell-labs.com/netlib/spin/whatispin.html`

[11] Kars, P., *Formal Methods in the Design of Storm Surge Barrier Control System*, Hand-outs of School on Embedded Systems 1996, Veldhoven, The Netherlands, 1997

[12] Sifakis, J., Yovine, S., *Compositional Specification of Timed Systems (Extended Abstract)*, STACS 1996

[13] Tripakis, S., Courcoubetis, C., *Extending Promela and Spin for Real Time*, TACAS '96, LCNS 1055, Springer Verlaag, 1996