Safety Verification of Rule-Based Controllers

Michal Paulovič [*], **Michal Kvasnica** [*], **Alexander Szücs** [*], and **Miroslav Fikar** [*]

[*] *Institute of Information Engineering, Automation, and Mathematics*
Faculty of Chemical and Food Technology
Slovak University of Technology in Bratislava
Radlinského 9, 812 37 Bratislava, Slovakia
(Tel: +421 259 325 352; e-mail: michal.kvasnica@stuba.sk)

Abstract: This paper proposes how to transform a control algorithm, written in MATLAB, into a hybrid system in order to verify its stability properties. The procedure first converts the code into a corresponding HYSDEL equivalent, which is then used to generate a suitable mathematical model. Safety verification is then formulated as a mixed integer linear program with feasibility objective.

Keywords: hybrid systems, safety verification, reachability analysis, MATLAB, HYSDEL

1. INTRODUCTION

Processes that evolve according to dynamic equations and logic rules can be described by hybrid models (Bemporad and Morari, 1999). Typical examples are real-time systems, where physical plants are governed by embedded rule-based controllers. When such systems are designed, it is important to provide a certificate that they will always operate in a safe manner, e.g. that the control rules never drive the plant into an "unsafe" area. Such a certificate can be provided by performing *reachability analysis* (Lygeros et al., 1999; Torrisi, 2003), which answers the following question: given a set of initial conditions $\mathcal{X}_0$, find the initial condition $x(0) \in \mathcal{X}_0$ for which the plant enters a set of unsafe states $\mathcal{X}_f$ in a finite number of steps T. If the reachability problem is infeasible, there is a guarantee that no such "unsafe" initial condition exists, hence providing the required safety certificate.

In this paper we propose how to solve the reachability problem when the control rules are implemented as a standard MATLAB function, composed of several IF-THEN-ELSE logic rules. First, the code of the function is converted into the HYSDEL (Torrisi, 2002) language, which is a high-level language tailored for describing behavior of hybrid systems. The translation process creates a one-to-one equivalent of the MATLAB control loop from which a suitable mathematical description is derived. The model then captures all interconnections between continuous plant dynamics and logic-based control rules. Once the model is available, the reachability problem is formulated as a mixed-integer linear program (MILP) with a pure feasibility objective.

The paper is structured as follows. First we introduce basic notion of hybrid systems and review most popular mathematical abstractions of such systems. Then, in Section 3 we describe the translation process in details. Reachability problems are then formulated in Section 4 and illustrated on a concrete example in Section 5. The paper is wrapped up by concluding remarks.

2. HYBRID SYSTEMS

Hybrid systems represent a compact framework which captures behavior of systems where continuous dynamics is coupled with discrete logic. Examples include, but are not limited to, systems with discrete-valued actuators (such as on/off switches), piecewise linear nonlinearities, and finite state machines. Mathematically, hybrid systems can be described by Piecewise Affine (PWA) models (Sontag, 1981), Mixed Logical Dynamical (MLD) systems (Bemporad and Morari, 1999), Linear Complementarity systems (Heemels et al., 2000) and max-min-plus-scaling models (De Schutter and Van den Boom, 2001). Under mild assumptions, all these frameworks are equivalent to each other (Heemels et al., 2001). In the sequel we review PWA and MLD approaches to modeling of hybrid systems. Since the aim of the paper is on verifying safety properties of closed-loop systems where the plant is governed by a set of internal IF-THEN-ELSE rules, only autonomous systems are considered.

2.1 Piecewise Affine (PWA) Systems

Autonomous PWA systems are defined by partitioning the space into polyhedral regions,and associating each region with a different linear (or affine) state-update equation:

$$x(k+1) = \begin{cases} A_1 x(k) + f_1 \text{ if } x(k) \in \mathcal{R}_1 \\ \vdots \\ A_n x(k) + f_n \text{ if } x(k) \in \mathcal{R}_n. \end{cases} \tag{1}$$

Here, $x(k) \in \mathbb{R}^{n_x}$ is the state vector at time instance k, $x(k+1)$ is the successor state at the next sampling instance, $\mathcal{R}_i \subseteq \mathbb{R}^{n_x}$, $i = 1, \ldots, n$ are polyhedral regions of the joint state-input space, and n is the number of individual affine dynamics. PWA systems arise naturally when nonlinear plants are approximated by the technique of multiple linearizations.

2.2 Mixed Logical Dynamical (MLD) Systems

MLD systems represent systems governed by discrete logic by a system of linear inequalities involving binary variables, which can be derived using so-called *big-M* formulation (Williams, 1993). To illustrate the procedure, consider a logic statement of the following form

$$\delta = \begin{cases} 1 \text{ if } a^T x \le b \\ 0 \text{ if otherwise} \end{cases} \tag{2}$$

which connects the truth value of binary variable δ to satisfaction of the linear inequality $a^T x \le b$ (which involves a real-valued variable $x \in \mathbb{R}^{n_x}$) via a logic equivalence relation. Let M and m denote, respectively, the maximum and minimum values which the linear expression $a^T x - b$ attains over the domain $\mathcal{X} \subseteq \mathbb{R}^{n_x}$, i.e.

$$M = \max_{x \in \mathcal{X}} a^T x - b, \tag{3a}$$

$$m = \min_{x \in \mathcal{X}} a^T x - b. \tag{3b}$$

Then the IF-THEN-ELSE rule (2) is equivalent to satisfaction of the following system of linear inequalities:

$$a^T x - b \le M(1 - \delta), \tag{4a}$$

$$a^T x - b \ge \epsilon + m\delta. \tag{4b}$$

Here, ϵ is a small constant, typically the machine precision, used to convert a strict inequality into a non-strict form. More complex logic expressions involving e.g. one-way implications ($\Leftarrow$ or $\Rightarrow$) and logic operations (and, or, negation) can be translated in a similar fashion, see e.g. (Williams, 1993; Bemporad and Morari, 1999).

In the most general form, autonomous MLD systems are described by

$$x(k+1) = Ax(k) + B_\delta \delta(k) + B_z z(k) + B_0, \tag{5a}$$

$$E_x x(k) + E_\delta \delta(k) + E_z z(k) \le E_0, \tag{5b}$$

where $x \in \mathbb{R}^{n_x}$ is the vector of states, $\delta \in \{0, 1\}^{n_\delta}$ is the vector of binary variables, $z \in \mathbb{R}^{n_z}$ is the vector of auxiliary real variables, and A, B_δ, B_z, B_0, E_x, E_δ, E_z, E_0 are matrices (or vectors) of appropriate dimensions. Given a value of $x(k)$, the state update $x(k+1)$ can be computed by solving a feasibility problem, i.e. by finding a compatible combination of binary $\delta(k)$ and real $z(k)$ variables which satisfy constraints (5b).

2.3 HYSDEL

Modeling of hybrid systems involves finding parameters of the corresponding mathematical model. In the PWA case (1), this boils down to finding matrices A_i, B_i, and the regions $\mathcal{R}_i$. In the MLD case (5), one needs to apply the big-M procedure to find matrices A, B_δ, B_z, B_0, E_x, E_δ, E_z, and E_0. Clearly, as the system to be described becomes more complex, such a "manual" approach to modeling can become cumbersome and error prone.

To accelerate development of hybrid models, HYSDEL (Hybrid Systems Description Language) was developed (Torrisi, 2002). It features a high-level modeling language which allows to describe behavior of hybrid systems using a custom syntax, which is similar to the C language. Once the system is described, the HYSDEL compiler parses the model and converts it into the MLD description (5) using the big-M technique. The MLD model can subsequently be converted into the PWA form.

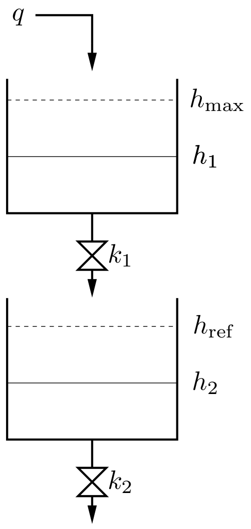

Fig. 1. The two-tanks arrangement.

Although HYSDEL significantly simplifies synthesis of mathematical representations of hybrid systems, it requires the user to learn its syntax. Therefore it is not directly applicable to verify control algorithms written in standard languages, such as in MATLAB or in C. To bridge this gap between standard control engineering tools and HYSDEL, we have developed a novel tool which automatically translates a MATLAB code to a corresponding HYSDEL model.

3. THE MATLAB-TO-HYSDEL TRANSLATOR

This section describes the process of translating a control algorithm written in MATLAB into the HYSDEL form. The translator consists of a *lexer*, which cleans the MATLAB code and identifies its key components. Operation of this phase is reported as Algorithm 1. Following components are being identified:

- operators: +, -, *, ^, /, &, |, ~, =, >, <
- keywords: `if`, `else`, `end`, `function`, `global`
- names of variables

The cleaned-up code is subsequently processed by a *parser*, which operates according to Algorithm 2. The parser first creates declarations of state, input and output variables, which serve as an interface between the control algorithm and the outside world. Subsequently, each line of the MATLAB source file is converted into its HYSDEL equivalent. Since HYSDEL only supports linear expressions, only a subset of valid MATLAB expressions can be converted.

To illustrate the translation, consider the following example. Given is a system composed of two liquid tanks, situated above each other, as shown in Figure 1. The linearized mathematical model of such a system is given by

$$h_1(k+\Delta_t) = h_1(k) + {}^{\Delta_t}\!/_{F_1}(q(k) - s_1 h_1(k)), \tag{6a}$$
$$h_2(k+\Delta_t) = h_2(k) + {}^{\Delta_t}\!/_{F_2}(s_1 h_1(k) - s_2 h_2(k)), \tag{6b}$$

where h_1 and h_2 are the liquid levels in the corresponding tanks, Δ_t is the sampling period, F_1, F_2 are the tanks' cross-section areas, $q(k)$ is the liquid inflow to the first tank (which is the control input), and s_1 and s_2 are linearization coefficients. The system is to be controlled by a rule-based controller:

$$q(k) = \begin{cases} q_{\max} \text{ if } h_2(k) \le h_{\text{ref}} \text{ and } h_1(k) \le h_{\max} \\ 0 \text{ otherwise.} \end{cases} \tag{7}$$

The rules are such that the liquid inflow is set to a non-zero value $q_{\max}$ whenever the liquid level in the bottom tank is below its reference h_{ref} and the upper tank is not overflowing. Otherwise the control input is set to zero. Important to notice is that due to accumulation of the liquid in the upper tank, liquid in the lower tank may continue to rise even after the control input is set to zero. In the next section we will show how to verify suitability of such a control scheme (i.e. that it guarantees a safe operation of the equipment where none of the tanks will overflow) by employing reachability analysis.

To illustrate the automatic code translator, suppose that the closed-loop system is described by the following MATLAB code:

```
function closed_loop

% declaration of internal states
global h1 h2

% declaration of parameters
F1 = 31.8319;
F2 = 31.8319;
s1 = 1;
s2 = 0.9;
dT = 5;
href = 76;  % reference level in centimeters
hmax = 100; % safety limit for the upper tank
qmax = 100; % default flow rate to upper tank

% control rules
if ( h2 <= href ) & ( h1 <= hmax )
 q = qmax;
else
 q = 0;
end

% dynamical system
h1 = h1 + dT/F1*( q - s1*h1 );
h2 = h2 + dT/F2*( s1*h1 - s2*h2 );
```

The code supported by the translator is structured as follows. First line always contains definition of the MATLAB function. State variables are represented as global variables, since they constitute an internal storage which needs to be updated between consecutive executions of the code. Concrete numerical values of parameters are provided next, followed by definition of switching control rules (7). The rules can contains logic operators such as `and` (&), `or` (|), and negation (∼). Multiple rules can be used and they can be interconnected using ELSEIF statements. The computed control action (denoted by the `q` variable in the code), is then used to update the internal state variables according to (6).

Applying the translator to such a MATLAB code produces its HYSDEL equivalent, reported next.

```
SYSTEM closed_loop {
 INTERFACE {
   STATE {
     REAL h1, h2;
   }
   PARAMETER {
   REAL F1 = 31.8319, F2 = 31.8319;
   REAL s1 = 1, s2 = 0.9;
   REAL dT = 5, href = 76;
   REAL hmax = 100, qmax = 100;
   }
 }
 IMPLEMENTATION {
   AUX {
     REAL q;
     BOOL delta1, delta2;
   }
   AD {
     delta1 = (h2 <= href);
     delta2 = (h1 <= hmax);
   }
   DA {
     q = {IF (delta1 & delta2) THEN qmax ELSE 0};
   }
   CONTINUOUS {
     h1 = h1 + dT/F1*( q - s1*h1 );
     h2 = h2 + dT/F2*( s1*h1 - s2*h2 );
   }
 }
}
```

Applying the HYSDEL compiler to the generated model, matrices of the MLD model (5) will be generated and saved to MATLAB. The MLD model can be subsequently converted to the PWA model (1) e.g. by using the `mpt_sys` function of the Multi-Parametric Toolbox (Kvasnica et al., 2004). The translator implements Algorithms 1 and 2 in the PHP language and is provided as a free web-based service available at `http://necron.sk/xant/`. Notice that the translator is under an active development and is subject to frequent changes in the following months.

4. SAFETY VERIFICATION VIA REACHABILITY ANALYSIS

To verify safety properties of closed-loop systems described as hybrid systems, one can solve the following problem.

Problem 4.1. Given is a hybrid system either in PWA or MLD form, a polyhedral set of initial conditions $\mathcal{X}_0$, a time horizon T, and a polyhedral set of "unsafe" states $\mathcal{X}_f$. Find an initial condition $x(0) \in \mathcal{X}_0$ for which the evolution of states reaches $\mathcal{X}_f$ in, at most, T steps, or determine that no such initial condition exists.

A feasible solution to Problem 4.1 constitutes at least one "unsafe" initial condition for which the control rules fail to meet a given safety goal. Infeasibility of Problem 4.1, on the other hand, provides a certificate that the system will always evolve in a safe manner.

Remark 4.2. Problem 4.1 can be easily extended to cover cases where the set of "unsafe" states $\mathcal{X}_f$ is a non-convex set represented by a finite number of polyhedra. Moreover, instead of verifying safety with respect to a fixed horizon

Algorithm 1 Lexer algorithm

```
INPUT: MATLAB code
OUTPUT: Cleaned MATLAB code with identified tokens
 1: take MATLAB code as string
 2: for each operator OR keyword in string do
 3:   if operator then
 4:     put space before and after operator
 5:   end if
 6:   if keyword then
 7:     put semicolon before and after keyword
 8:   end if
 9: end for
10: for each character in string do
11:   if it is TAB then
12:     replace it with space
13:   end if
14:   if it is carriage return OR line feed then
15:     replace it with semicolon
16:   end if
17:   if it is per cent sign then
18:     while it is NOT (semicolon OR carriage return
        OR line feed) do
19:       shift to the next character
20:     end while
21:   end if
22:   if it is space as previous character then
23:     shift to the next character
24:   end if
25:   if it is semicolon as previous character then
26:     shift to the next character
27:   end if
28: end for
29: for each semicolon do
30:   break the rest of the string into new line
31: end for
```

T, one can look for the minimal value of T for which the system violates safety conditions. This can be achieved e.g. by employing bisection in conjunction with Problem 4.1.

If the hybrid system to be verified is given in the MLD form (5), Problem 4.1 can be approached by solving a feasibility mixed-integer linear program:

$$\begin{aligned}
&\text{find } x(0) && \text{(8a)}\\
&\text{s.t. } x(0) \in \mathcal{X}_0, && \text{(8b)}\\
&\quad x(k+1) = Ax(k) + B_\delta \delta(k) + B_z z(k) + B_0, && \text{(8c)}\\
&\quad E_x x(k) + E_\delta \delta(k) + E_z z(k) \leq E_0, && \text{(8d)}\\
&\quad x(T) \in \mathcal{X}_f. && \text{(8e)}
\end{aligned}$$

Here, constraints (8c)–(8d), which are defined for $k = 0, \ldots, T-1$, describe evolution of the MLD system on horizon T, cf. (5). Under the assumption that $\mathcal{X}_0$ and $\mathcal{X}_f$ are polyhedral sets, they can be described by

$$\mathcal{X}_0 = \{x \mid H_0 x \leq K_0\}, \quad \text{(9a)}$$
$$\mathcal{X}_f = \{x \mid H_f x \leq K_f\}, \quad \text{(9b)}$$

where H_0, H_f, K_0, K_f are matrices which represent the half-space representation of such sets. Therefore all constraints in (8) are linear in the decision variables $x(k)$, $\delta(k)$, and $z(k)$, for $k = 0, \ldots, T$. Since $\delta(k)$ are vectors of binary variables while $x(k)$ and $z(k)$ are real-valued vectors, it follows that problem (8) is a mixed-integer linear program (MILP) with a pure feasibility

Algorithm 2 Parser algorithm

```
INPUT: MATLAB tokens
OUTPUT: HYSDEL code
 1: declare state, input and output variables
 2: for each line do
 3:   if "if" found then
 4:     parse next line
 5:     store condition(s)
 6:     parse next line
 7:     store "if" value
 8:     parse next line
 9:     if "else" found then
10:       store "else" value
11:       parse next line
12:     end if
13:   else
14:     store parameter
15:   end if
16: end for
17: for each condition do
18:   store auxiliary variable
19:   negate condition
20:   if negated condition found then
21:     store negated aux var
22:   end if
23: end for
24: create HYSDEL pattern
25: fill pattern with stored strings
26: generate HYSDEL code
```

objective. Such MILP problems can be formulated e.g. by YALMIP (Löfberg, 2004) and solved efficiently using state-of-the-art solvers, such as with GLPK (Makhorin, 2001) or CPLEX (ILOG, Inc., 2003).

If the hybrid system is given in its PWA form (1), the corresponding reachability problem is formulated as follows

$$\begin{aligned}
&\text{find } x(0) && \text{(10a)}\\
&\text{s.t. } x(0) \in \mathcal{X}_0, && \text{(10b)}\\
&\quad \delta_i(k) \Leftrightarrow x(k) \in \mathcal{R}_i,\ i = 1, \ldots, n, && \text{(10c)}\\
&\quad \delta_i(k) \Leftrightarrow x(k+1) = A_i x(k) + f_i,\ i = 1, \ldots, n, && \text{(10d)}\\
&\quad \sum_{i=1}^{n} \delta_i(k) = 1, && \text{(10e)}\\
&\quad x(T) \in \mathcal{X}_f. && \text{(10f)}
\end{aligned}$$

Here, $\delta_i(k)$, $i = 1, \ldots, n$ (where n is the number of PWA regions) are binary selectors which take the value of 1 if and only if the state $x(k)$ is contained in the i-th polyhedral region $\mathcal{R}_i$, cf. (10c). The truth value of the corresponding binary selector then activates a particular state-update equation in (10d). The logic equivalence ($\Leftrightarrow$) rules can again be translated into mixed-integer inequalities using the big-M method, as shown in Section 2.2. Finally, (10e) is an exclusive-or condition which only allows the state to reside in a single polyhedral region. Again, problem (10) can be readily cast as a feasibility MILP with binary variables $\delta_i(k)$, and real variables $x(k)$, $k = 0, \ldots, T$.

Due to equivalence between PWA and MLD systems (Heemels et al., 2001), Problem 4.1 can be answered either by solving (8) or (10). The particular selection depends on the number of binary variables induced by a particular choice of the hybrid model (either PWA or MLD). In the next section we apply the described safety verification procedure to an illustrative example.

5. EXAMPLE

We revisit the two-tanks example discussed in the previous section. We reiterate that the aim is to verify that the rules-based controller (7), connected with the system (6), always meets a certain safety goal. In this example we want to verify that the liquid level in the bottom tank (h_2) stays below a pre-defined threshold, say h_{unsafe}. The set of unsafe states is hence $\mathcal{X}_f = \{h_2 \mid h_2 > h_{\text{unsafe}}\}$ with $h_{\text{unsafe}} = 84$ (all levels are expressed in centimeters). The set of initial states is $\mathcal{X}_0 = \{h_1, h_2 \mid 0 \leq h_1 \leq a,\ 0 \leq h_2 \leq a\}$. Two scenarios were considered: one with $a = 30$ and the other one with $a = 70$.

To formulate the verification problem, the HYSDEL model of the closed-loop system is first compiled by the HYSDEL compiler, which generates the corresponding MLD model description. This model was subsequently used to formulate the MILP problem (8) using YALMIP (Löfberg, 2004):

```
 1 % definition of decision variables
 2 x = {}; d = {}; z = {};
 3 for k = 1:T+1
 4  x{k} = sdpvar(nx, 1);
 5  z{k} = sdpvar(nz, 1);
 6  d{k} = binvar(nd, 1);
 7 end
 8
 9 % constraint on the initial condition
10 CON = [ 0 <= x{1}(1) <= a; 0 <= x{1}(2) <= a ];
11
12 % constraint on the final state in the unsafe set
13 CON = CON + [ x{end}(2) > 84 ];
14
15 % time evolution of the MLD model
16 for k = 1:T
17  CON = CON + [ x{k+1} == A*x{k}+Bd*d{k}+Bz*z{k}+B0 ];
18  CON = CON + [ Ex*x{k} + Ed*d{k} + Ez*z{k} <= E0 ];
19 end
20
21 % solve the fesibility problem
22 solution = solvesdp(CON, []);
23
24 % return the "unsafe" initial condition if it exists
25 if solution.problem == 0
26  xunsafe = double(x{1});
27 end
```

Here, the decision variables $x(t)$, $\delta(t)$, $z(t)$ are first defined on lines 2–7 for $t = 0, \ldots, T$. Notice that variables $\delta(t)$ are declared as binary variables on Line 6. Line 10 then specifies the set of initial conditions for both state variables (the index in $\{\cdot\}$ denotes the time step k, while the index in $(\cdot)$ specifies position of the particular element in the state vector). Similarly, the set of unsafe states is defined on Line 13. Then, constraints (8c)–(8d) are repeated for $k = 0, \ldots, T-1$ on Lines 17 and 18. Finally, the formulated verification problem is solved by calling the `solvesdp` command. If the problem is feasible for some value of T, (cf. Lines 25–27), value of the "unsafe" initial condition is returned. If the problem is infeasible for all $T \leq T_{\max}$, then there is no such unsafe starting point and therefore the controller always operates in a safe manner within of $T_{\max}$ time steps.

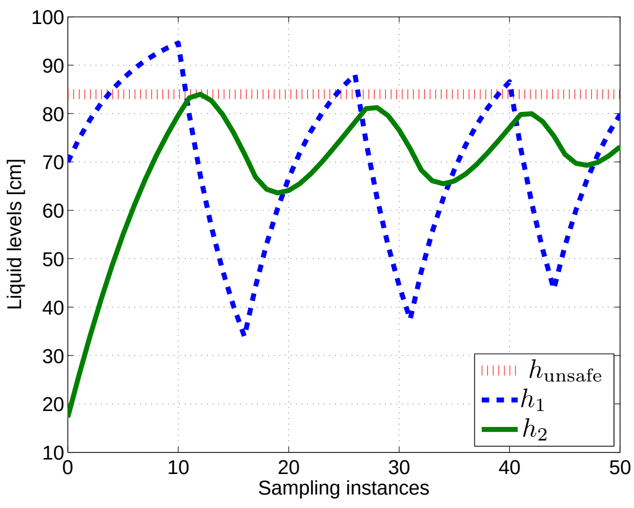

Fig. 2. Simulation scenario for an unsafe initial condition.

For $a = 30$ we have solved problem (8) for $T = 0, \ldots, 50$ (which corresponds to 250 seconds) using the GLPK MILP solver. The problem was infeasible for any value of $T \leq T_{\max}$, which certifies a safe behavior of the control system (7) for any initial condition bounded by $0 \leq h_i \leq a$, $i = 1, 2$. However, for $a = 70$ the safety verification problem was feasible for $T = 12$, which resulted into the unsafe initial condition $h_1(0) = 70$ and $h_2(0) = 17.2606$. Simulation of the closed-loop system starting from this initial condition is shown in Figure 2, which indeed confirms that the safety barrier $h_2 > 84$ is violated after 12 sampling instances.

6. CONCLUSIONS

In this paper we have proposed how to verify safety properties of logic-based control laws written in MATLAB. First, the MATLAB code was converted into its HYSDEL equivalent by means of an automated lexing and parsing procedure. The HYSDEL model was subsequently converted into a mathematical form, represented either by a PWA or by an MLD model. Finally, the verification was performed by solving a mixed-integer linear program. A motivating example was provided to illustrate individual steps. The main benefit of this work is that it allows theoretical verification algorithms to be applied to a subset of ordinary computer code, in this case represented by MATLAB. However, the translator can be easily modified to support other programming languages as well, for instance the C language or Java.

ACKNOWLEDGMENTS

The authors are pleased to acknowledge the financial support of the Scientific Grant Agency of the Slovak Republic under the grant 1/0095/11. This work was supported by the Slovak Research and Development Agency under the contracts No. VV-0029-07 and No. LPP-0092-07.

REFERENCES

Bemporad, A. and Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3), 407–427.

De Schutter, B. and Van den Boom, T. (2001). On model predictive control for max-min-plus-scaling discrete event systems. *Automatica*, 37(7), 1049–1056.

Heemels, W.P.M., De Schutter, B., and Bemporad, A. (2001). Equivalence of hybrid dynamical models. *Automatica*, 37(7), 1085–1091.

Heemels, W., Schumacher, J., and Weiland, S. (2000). Linear complementarity systems. *SIAM Journal on Applied Mathematics*, 60(4), 1234–1269.

ILOG, Inc. (2003). *CPLEX User Manual*. Gentilly Cedex, France. `http://www.ilog.fr/products/cplex/`.

Kvasnica, M., Grieder, P., and Baotić, M. (2004). Multi-Parametric Toolbox (MPT). Available from `http://control.ee.ethz.ch/~mpt/`.

Löfberg, J. (2004). YALMIP. Available from `http://users.isy.liu.se/johanl/yalmip/`.

Lygeros, J., Tomlin, C., and Sastry, S. (1999). Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3), 349–370.

Makhorin, A. (2001). *GLPK - GNU Linear Programming Kit*. `http://www.gnu.org/directory/libs/glpk.html`.

Sontag, E.D. (1981). Nonlinear regulation: The piecewise linear approach. *IEEE Trans. on Automatic Control*, 26(2), 346–358.

Torrisi, F.D. (2002). Hybrid System DEscription Language (HYSDEL). Available from `http://control.ee.ethz.ch/~hybrid/hysdel/`.

Torrisi, F. (2003). *Modeling and Reach-Set Computation for Analysis and Optimal Control of Discrete Hybrid Automata*. Ph.D. thesis, ETH Zurich.

Williams, H. (1993). *Model Building in Mathematical Programming*. John Wiley & Sons, Third Edition.