HYSDEL 2.0.5 – User Manual

Fabio Danilo Torrisi, Alberto Bemporad, Gioele Bertini,
Peter Hertach, Dominic Jost, Domenico Mignone

September 10, 2002

Abstract

This is the HYSDEL user guide. HYSDEL allows to model a class of hybrid systems described by interconnections of linear dynamic systems, automata, if-then-else and propositional logic rules. For this class of systems we present general techniques for transforming an abstract representation into a set of constrained linear difference equations involving integer and continuous variables. The resulting model can be immediately used for optimization, to solve, e.g., optimal control problems or as an intermediate step to obtain other popular representations such as piecewise affine systems.

The developer's manual [39] completes the present document with the details on the implementation of HYSDEL and is included in the source distribution.

Contents

Acknowledgements

The authors wish to acknowledge all the users of HYSDEL which provided fruitful feedback and encouraged us to further develop the tool.

The first version of the HYSDEL language was designed by Fabio Torrisi, Alberto Bemporad, and Domenico Mignone. Samarjit Chakraborty implemented a prototype of a compiler based on that preliminary language.

After a complete redesign of the language, HYSDEL was programmed by: Gioele Bertini, Peter Hertach, Dominic Jost, and Fabio Torrisi. Fabio Torrisi is currently maintaining the package and can be reached at the following email address: torrisi@aut.ee.ethz.ch.

The authors wish to thank Prof. Morari for fruitful discussions, Frank J. Christophersen and Dominik Niederberger for proof reading the manual.

This work was partially supported by the European Union through the IST project "Computation and Control" and the Swiss National Science Foundation. Dominic Jost was supported by ABB.

Preface

What are hybrid models

Mathematical models reproduce the behavior of physical phenomena. By considering the process at different levels of detail, different models of the same process are usually available in applied sciences. Models should not be too simple, otherwise they do not capture enough details of the process, but also not too complicated in order to formulate and efficiently solve interesting analysis and synthesis problems.

In the last years, several computer scientists and control theorists have investigated models describing the interaction between continuous dynamics described by differential or difference equations, and logical components described by finite state machines, if-then-else rules, propositional and temporal logic [3]. Such heterogeneous models, denoted as *hybrid* models, switch among many operating modes, where each mode is associated with a different dynamic law, and mode transitions are triggered by events, like states crossing pre-specified thresholds.

The practical relevance of hybrid models is twofold. The increasing profitability of logic controllers embedded in a continuous environment is demanding for adequate modeling, analysis and design tools, for instance in the automotive industry [5]. Moreover, many physical phenomena admit a natural hybrid description, like circuits integrating relays or diodes, biomolecular networks [1], and TCP/IP networks in [34].

Hybrid models are needed to address a number of problems, like definition and computation of trajectories, stability and safety analysis, control, state estimation, etc. The definition of trajectories is usually associated with a *simulator*, a tool able to compute the time evolution of the variables of the system. This may seem straightforward at first, however many hybrid formalisms introduce extra behaviors like Zeno effects [37], that complicate the definition of trajectories. Although simulation allows to probe the model, it certainly does not permit to assess structural properties of the model. In fact any analysis based on simulation is likely to miss the subtle phenomena that a model may generate, especially in the case of hybrid models. Tools like *reachability analysis* and *piecewise quadratic Lyapunov stability* are becoming a standard in analysis of hybrid systems. Reachability analysis (or *safety analysis* or *formal verification*) aims at detecting if a hybrid model will eventually reach an unsafe state configuration or satisfy a temporal logic formula [21]. Reachability analysis relies on a reach set computation algorithm, which is strongly related to the mathematical model of the system [48]. Piecewise quadratic Lyapunov stability [38], is a deductive way to prove the stability of an equilibrium point of a subclass of hybrid systems (piecewise linear systems), the computational burden is usually low, at the price of a convex relaxation of the problem which leads to conservative results. While for pure linear systems it exists a complete theory for the *identification* of unknown system parameters, the extension to general hybrid systems is still under investigation. *Controlling* a model (and therefore a process) means to choose the input such that the output tracks some desired reference. The control (or *scheduling*) problem can be tackled in several ways, according to the model type and control objective. Most of the control approaches are based on optimal control ideas (see e.g. [19]). The dual problem of control is *state estimation*, which amounts to compute the value of unmeasurable state variables based on the measurements of output variables.

The main applicative relevance of state estimation is for control, when direct measurements of the state vector are not possible, and for monitoring and fault detection problems.

Several classes of hybrid systems have been proposed in the literature, each class is usually tailored to solve a particular problem. Timed automata and hybrid automata have proved to be a successful modeling framework for formal verification, and have been widely used in the literature. The starting point for both models is a finite state machine equipped with continuous dynamics. In the theory of *timed automata* [4], the dynamic part is the continuous-time flow $\dot{x} = 1$. Efficient computational tools complete the theory of timed automata and allow to perform verification [17,23] and scheduling [6] of such models. Timed automata were extended to *linear hybrid automata* [2], where the dynamics is modeled by the differential inclusion $a \leq \dot{x} \leq b$. Specific tools allow to verify such models against safety and liveness requirements. Linear hybrid automata were further extended to *hybrid automata* where the continuous dynamics is governed by differential equations. Tools exist to model and analyze those systems, either directly [22,47] or by approximating the model with timed automata or linear hybrid automata [51].

In this paper we will focus on *discrete hybrid automata* (DHA). DHA result from the connection of a *finite state machine* (FSM), which provides the discrete part of the hybrid system, with a *switched affine system* (SAS), which provides the continuous part of the hybrid dynamics. The interaction between the two is based on two connecting elements: The *event generator* (EG) and the *mode selector* (MS). The EG extracts logic signals from the continuous part. Those logic events and other exogenous logic inputs trigger the switch of the state of the FSM. The MS combines all the logic variables (states, inputs, and events) to choose the mode (=continous dynamics) of the SAS. Continuous dynamics and reset maps are expressed as linear affine difference equations. DHA models are a mathematical abstraction of the features provided by other computational oriented and domain specific hybrid frameworks: *Mixed logical dynamical* (MLD) *models* [12], *piecewise affine* (PWA) *systems* [49], *linear complementarity* (LC) *systems* [20, 31, 32, 46, 52], *extended linear complementarity* (ELC) *systems* [24, 33], and *max-min-plus-scaling* (MMPS) *systems* [25, 33]. In particular, as shown first in [50] and then, with different arguments, in [8, 33] all those modeling frameworks are equivalent and it is possible represent the same system with models of each class.

DHA are formulated in discrete time. Despite the fact that the effects of sampling can be neglected in most applications, subtle phenomena such as Zeno behaviors do not appear in discrete time. Although it is possible to consider hybrid automata in continuous-time, computation is efficiently tractable only for discrete time models[1]. As anticipated DHA generalize many computational oriented models for hybrid systems and therefore represent the starting point for solving complex analysis and synthesis problems for hybrid systems.

In particular the MLD and PWA frameworks allow to recast reachability/observability analysis, optimal control, and receding horizon estimation as mixed-integer linear/quadratic optimization problems. Reachability analysis algorithms were developed in [14] for MLD and PWA hybrid systems, extended in [15] for stability and performance analysis of hybrid control systems, and in [16] to perform parametric verification. In [10] the authors presented a novel approach for solving scheduling problems using combined reachability analysis and quadratic optimization for MLD and PWA models. For feedback control, in [12] the authors propose a model predictive control scheme which is able to stabilize MLD systems on desired reference trajectories while fulfilling operating constraints, and possibly take into account previous qualitative knowledge in the form of heuristic rules. Similarly, the dual problem of

[1] Many tools for continuous-time hybrid models perform internally a time discretization of the model in order to execute the computations.

state estimation admits a receding horizon solution scheme [11].

In [31, 52] (linear) complementarity systems in *continuous* time have been studied. Applications include constrained mechanical systems, electrical networks with ideal diodes or other dynamical systems with piecewise linear relations, variable structure systems, constrained optimal control problems, projected dynamical systems and so on [31, Ch. 2]. Issues related to modeling, well-posedness (existence and uniqueness of solution trajectories), simulation and discretization have been of particular interest.

Finally, we mention that identification techniques for piecewise affine systems were recently developed [13, 28, 40], that allow to derive models (or parts of models) from input/output data.

In this manual we present a theoretical framework for DHA systems. We will go through the steps needed for modeling a system as DHA. We will first detail the process of translating propositional logic involving Boolean variables and linear threshold events over continuous variables into mixed-integer linear inequalities, generalizing several results available in the literature [12, 44, 53], in order to get an equivalent MLD form of a DHA system, which is later used to obtain the equivalent PWA, LC, ELC, and MMPS system. We will present the tool HYSDEL (=HYbrid Systems DEscription Language), that allows describing the hybrid dynamics in a textual form, and a related compiler which provides different model representations of the given hybrid dynamics.

The latest version of the HYSDEL compiler is available at `http://control.ethz.ch/~hybrid/hysdel`. Applications of HYSDEL can be found at `http://control.ethz.ch/~hybrid/`.

1 Discrete Hybrid Automata

Discrete hybrid automata (DHA) are the interconnection of a finite state machine and a switched linear dynamic system through a mode selector and an event generator (see Figure 1.1).

In the following we will use the fact that any discrete variable $\alpha \in \{\alpha_1, \ldots, \alpha_j\}$, admits a Boolean encoding $a \in \{0,1\}^{d(j)}$. From now on we will refer to either the variable or its encoding with the same name.

1.1 Switched Affine System (SAS)

A switched affine system is a collection of linear affine systems:

$$x_r'(k) = A_{i(k)}x_r(k) + B_{i(k)}u_r(k) + f_{i(k)} \tag{1.1a}$$
$$y_r(k) = C_{i(k)}x_r(k) + D_{i(k)}u_r(k) + g_{i(k)}, \tag{1.1b}$$

where $k \in \mathbb{Z}^+$ is the time indicator, $'$ denotes the successor operator ($x_r'(k) = x_r(k+1)$), $x_r \in \mathcal{X}_r \subseteq \mathbb{R}^{n_r}$ is the continuous state vector, $u_r \in \mathcal{U}_r \subseteq \mathbb{R}^{m_r}$ is the exogenous continuous input vector, $y_r \in \mathcal{Y}_r \subseteq \mathbb{R}^{p_r}$ is the continuous output vector, $\{A_i, B_i, f_i, C_i, D_i, g_i\}_{i\in\mathcal{I}}$ is a collection of matrices of suitable dimensions, and the mode $i(k) \in \mathcal{I}$ is an input signal that chooses the linear state update dynamics. We denote by $\sharp\mathcal{I} = s$ the number of elements in $\mathcal{I}$. A SAS of the form (1.1) preserves the value of the state when a switch occurs, however it is possible to implement reset maps on a SAS, as we will show later in this section. A SAS can be rewritten as the combination of linear terms and *if-then-else* rules: the state-update Equation (1.1a) is equivalent to

$$z_1(k) = \begin{cases} A_1x_r(k) + B_1u_r(k) + f_1, & \text{if } (i(k)=1), \\ 0, & \text{otherwise,} \end{cases} \tag{1.2a}$$

$$\vdots$$

$$z_s(k) = \begin{cases} A_sx_r(k) + B_su_r(k) + f_s & \text{if } (i(k)=s), \\ 0 & \text{otherwise,} \end{cases} \tag{1.2b}$$

$$x_r'(k) = \sum_{i=1}^{s} z_i(k) \tag{1.2c}$$

where $z_i(k) \in \mathbb{R}^{n_r}, i = 1, \ldots, s$, and (1.1b) admits a similar transformation.

1.2 Event Generator (EG)

An event generator is a mathematical object that generates a logic signal according to the satisfaction of a linear affine constraint:

$$\delta_e(k) = f_{\mathrm{H}}(x_r(k), u_r(k), k), \tag{1.3}$$

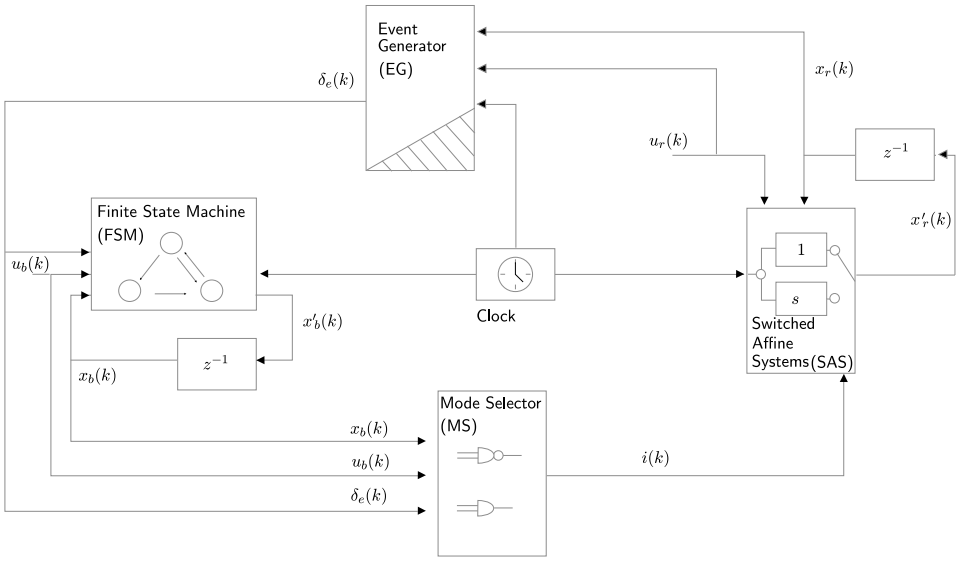


Figure 1.1: A discrete hybrid automaton (DHA) is the connection of a finite state machine (FSM) and a switched affine system (SAS), through a mode selector (MS) and an event generator (EG). The output signals are omitted for clarity

where $f_{\mathrm{H}} : \mathbb{R}^{n_r} \times \mathbb{R}^{m_r} \times \mathbb{Z}_{\geq 0} \to \mathcal{D} \subseteq \{0,1\}^{n_e}$ is a vector of descriptive functions of a linear hyperplane, and $\mathbb{Z}_{\geq 0} \triangleq \{0, 1, \dots\}$ is the set of nonnegative integers. In particular, *time events* are modeled as: $[\delta_e^i(k) = 1] \leftrightarrow [kT_s \geq t_0]$, where T_s is the sampling time, while *threshold events* are modeled as: $[\delta_e^i(k) = 1] \leftrightarrow [a^T x_r(k) + b^T u_r(k) \leq c]$, where the superscript i denotes the i-th component of a vector.

1.3 Finite State Machine (FSM)

A finite state machine[1] (or automaton) is a discrete dynamic process that evolves according to a logic state update function:

$$x'_b(k) = f_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k)), \tag{1.4a}$$

where $x_b \in \mathcal{X}_b \subseteq \{0,1\}^{n_b}$ is the Boolean state, $u_b \in \mathcal{U}_b \subseteq \{0,1\}^{m_b}$ is the exogenous Boolean input, $\delta_e(k)$ is the endogenous input coming from the EG, and $f_{\mathrm{B}} : \mathcal{X}_b \times \mathcal{U}_b \times \mathcal{D} \to \mathcal{X}_b$ is a deterministic logic function. A FSM can be conveniently represented using an oriented graph. A FSM may also have an associated Boolean output

$$y_b(k) = g_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k)), \tag{1.4b}$$

where $y_b \in \mathcal{Y}_b \subseteq \{0,1\}^{p_b}$.

Example 1 Figure 1.2 shows a finite state machine where $u_b = [u_{b1}\ u_{b2}]^T$ is the input vector, and $\delta = [\delta_1 \dots \delta_4]^T$ ia a vector of signals coming from the event generator. The state transition function

[1] In this paper we will only refer to synchronous finite state machines, where the transitions may happen only at sampling times. The adjective synchronous will be omitted for brevity.

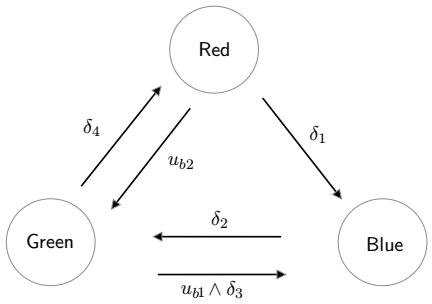


Figure 1.2: Example of finite state machine

is:

$$x_b'(k) = \begin{cases} \mathsf{Red} & \text{if} \quad ((x_b(k) = \mathsf{Green}) \wedge d_4) \vee ((x_b(k) = \mathsf{Red}) \wedge \neg u_{b2} \wedge \neg \delta_1), \\ \mathsf{Green} & \text{if} \quad ((x_b(k) = \mathsf{Red}) \wedge u_{b2}) \vee ((x_b(k) = \mathsf{Blue}) \wedge \delta_2) \vee \\ & \quad\;\; ((x_b(k) = \mathsf{Green}) \wedge \neg \delta_4 \wedge \neg (u_{b1} \wedge \delta_3)), \\ \mathsf{Blue} & \text{if} \quad ((x_b(k) = \mathsf{Red}) \wedge \delta_1) \vee ((x_b(k) = \mathsf{Green}) \wedge (u_{b1} \wedge \delta_3)) \vee \\ & \quad\;\; ((x_b(k) = \mathsf{Blue}) \wedge \neg \delta_2)). \end{cases} \tag{1.5}$$

By associating a Boolean vector $x_b = \left[\begin{smallmatrix} x_{b1} \\ x_{b2} \end{smallmatrix}\right]$ to each state ($\mathsf{Red} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$, $\mathsf{Green} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, and $\mathsf{Blue} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$), one can rewrite (1.5) as:

$$\begin{aligned} x_{b1}' &= \neg((\neg x_{b1} \wedge x_{b2} \wedge d_4) \vee (\neg x_{b1} \wedge \neg x_{b2} \wedge \neg u_{b2} \wedge \neg \delta_1) \\ &\quad \vee \neg((\neg x_{b1} \wedge \neg x_{b2} \wedge u_{b2}) \vee (x_{b1} \wedge \neg x_{b2} \wedge \delta_2) \vee (\neg x_{b1} \wedge x_{b2} \wedge \neg \delta_4 \wedge \neg (u_{b1} \wedge \delta_3))) \\ &\quad \vee ((\neg x_{b1} \wedge \neg x_{b2} \wedge \delta_1) \vee (\neg x_{b1} \wedge x_{b2} \wedge (u_{b1} \wedge \delta_3)) \vee (x_{b1} \wedge \neg x_{b2} \wedge \neg \delta_2)), \\ x_{b2}' &= \neg((\neg x_{b1} \wedge x_{b2} \wedge d_4) \vee (\neg x_{b1} \wedge \neg x_{b2} \wedge \neg u_{b2} \wedge \neg \delta_1) \\ &\quad \vee ((\neg x_{b1} \wedge \neg x_{b2} \wedge u_{b2}) \vee (x_{b1} \wedge \neg x_{b2} \wedge \delta_2) \vee (\neg x_{b1} \wedge x_{b2} \wedge \neg \delta_4 \wedge \neg (u_{b1} \wedge \delta_3))) \\ &\quad \vee \neg((\neg x_{b1} \wedge \neg x_{b2} \wedge \delta_1) \vee (\neg x_{b1} \wedge x_{b2} \wedge (u_{b1} \wedge \delta_3)) \vee (x_{b1} \wedge \neg x_{b2} \wedge \neg \delta_2)), \end{aligned}$$

where the time index, k, has been omitted for brevity.

1.4 Mode Selector (MS)

The logic state $x_b(k)$, the Boolean inputs $u_b(k)$, and the events $\delta_e(k)$ select the dynamic mode $i(k)$ of the SAS through a Boolean function $f_{\mathrm{M}} : \mathcal{X}_b \times \mathcal{U}_b \times \mathcal{D} \to \mathcal{I}$, which is therefore called *mode selector*. The output of this function

$$i(k) = f_{\mathrm{M}}(x_b(k), u_b(k), \delta_e(k)) \tag{1.6}$$

is called *active mode*. We say that a *mode switch* occurs at step k if $i(k) \neq i(k-1)$. Note that contrarily to continuous time hybrid models, where switches can occur at any time, in our discrete-time setting a mode switch can only occur at sampling instants.

1.5 Reset Maps

In correspondence with a mode switch $i(k) = j$, $i(k-1) = h \neq j$, $h, j \in \mathcal{I}$, instead of evolving $x_r'(k) = A_j x_r(k) + B_j u_r(k) + f_j$ it is possible to associate a reset of the continuous state vector

$$x_r'(k) = \phi_{\overrightarrow{hj}}(x_r(k), u_r(k)) = A_{\overrightarrow{hj}} x_r(k) + B_{\overrightarrow{hj}} u_r(k) + f_{\overrightarrow{hj}} \tag{1.7}$$

the function $\phi_{\overrightarrow{hj}}$ is called *reset map*. The reset can be considered as a special dynamics that only acts for one sampling step. Figure 1.3(a) shows how a reset affects the state evolution: At time $k = 5$ the system is in mode $i(5) = 1$, at time $k = 6$ the state $x_r(6) = A_1 x_r(5) + B_1 u_r(5) + f_1$ enters the region $x_r \geq 0$. This generates an event $\delta_e(6)$ through the EG, which in turn causes the MS to change the system dynamics to $i(6) = 2$. The mode switch $1 \to 2$ resets $x_r(7) = A_{\overrightarrow{12}} x_r(6) + B_{\overrightarrow{12}} u_r(6) + f_{\overrightarrow{12}}$. If the state $x_r(7)$ after reset belongs again to the region where the mode 2 is active, $i(7) = 2$, the successor state is $x_r(8) = A_2 x_r(7) + B_2 u_r(7) + f_2$. It might even happen that $x_r(7)$ belongs to another region, say a region where mode 3 is active, $i(7) = 3$: In this case, since $i(6) \neq i(7)$, a further reset $2 \to 3$ is applied, $x_r(8) = A_{\overrightarrow{23}} x_r(7) + B_{\overrightarrow{23}} u_r(7) + f_{\overrightarrow{23}}$.

Proposition 1 *A DHA Σ^{a} with reset conditions can be rewritten as a DHA Σ without reset conditions.*

Proof. Let the superscript a denote the variables of Σ^{a}. Let $x_b(k) = [x_b^{\mathrm{a}}(k); x_b^{\mathrm{a}}(k-1); \delta_e^{\mathrm{a}}(k-1); u_b^{\mathrm{a}}(k-1)]$, where ; denotes the concatenation of column vectors, $\delta_e(k) = \delta_e^{\mathrm{a}}(k)$, $u_b(k) = u_b^{\mathrm{a}}(k)$, $x_r(k) = x_r^{\mathrm{a}}(k)$, $u_r(k) = u_r^{\mathrm{a}}(k)$, $y_r(k) = y_r^{\mathrm{a}}(k)$, $y_b(k) = y_b^{\mathrm{a}}(k)$, and define the FSM

$$x_b'(k) = f_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k)) = \begin{bmatrix} f_{\mathrm{B}}^{\mathrm{a}}(x_b^{\mathrm{a}}(k), u_b(k), \delta_e(k)) \\ x_b^{\mathrm{a}}(k) \\ \delta_e(k) \\ u_b(k) \end{bmatrix}.$$

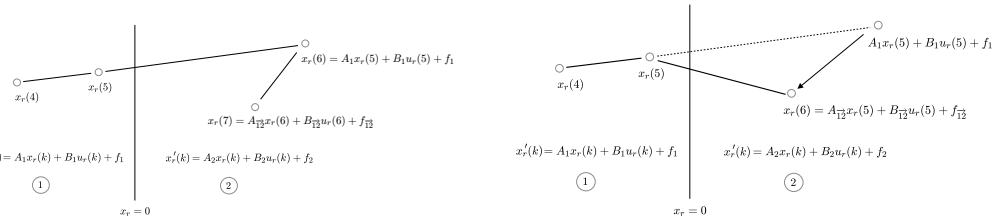
The SAS dynamics of Σ is defined as in (1.1) with $i(k) \in \{1, 2, \ldots, s^{\mathrm{a}}, s^{\mathrm{a}}+1, \ldots, s^{\mathrm{a}}+r\}$, where s^{a} is the number of modes of Σ^{a}, and $r \leq s^{\mathrm{a}}(s^{\mathrm{a}}-1)$ is the number of reset maps (we assume that when the mode switch $h \to j$ of Σ_0 does not have an associated reset map $f_{\overrightarrow{hj}}^{\mathrm{a}}$, then $f_{\overrightarrow{hj}}^{\mathrm{a}}$ defaults to the j-th state update map). The MS of Σ should internally compute $j = i^{\mathrm{a}}(k)$, $h = i^{\mathrm{a}}(k-1)$, compare them, and then choose either the j-th dynamics (if $j = h$, or $j \neq h$ and $f_{\overrightarrow{hj}}^{\mathrm{a}}$ is not specified) or the reset dynamics $f_{\overrightarrow{hj}}^{\mathrm{a}}$. Since $i^{\mathrm{a}}(k) = f_{\mathrm{M}}^{\mathrm{a}}(x^{\mathrm{a}}(k), u_b^{\mathrm{a}}(k), \delta_e^{\mathrm{a}}(k))$, $i^{\mathrm{a}}(k-1) = f_{\mathrm{M}}^{\mathrm{a}}(x^{\mathrm{a}}(k-1), u_b^{\mathrm{a}}(k-1), \delta_e^{\mathrm{a}}(k-1))$, it follows that the mode $i(k)$ is a function of $x_b(k)$, $u_b(k)$, $\delta_e(k)$. □

In some circumstances, it is desirable to predict the mode switch and to anticipate the reset by one sampling step, i.e., to reset the state *before* the guardline is actually crossed. Assume that the event $\delta_e(k)$ triggering the mode switch does not depend on the continuous input $u_r(k)$, and that the logic input $u_b(k)$ does not affect the mode selector. In this case, $i'(k) = f_{\mathrm{M}}(x_b'(k), f_{\mathrm{H}}(x_r'(k), k))$ only depends on quantities available at step k, and a mode switch $i'(k) \neq i(k)$ can be predicted already at step k. In this case, we can apply the corresponding reset directly for $x_r'(k) = A_{\overrightarrow{hj}} x_r(k) + B_{\overrightarrow{hj}} u_r(k) + f_{\overrightarrow{hj}}$ where $h = i(k)$, $j = i'(k)$. This kind of resets will be referred to as *predicted resets*, in order to distinguish them from the resets described before, that we will call *a-posteriori resets*.

Consider Figure 1.3(b). At time $k = 5$ the state $x_r(5)$ and the input $u_r(5)$ are such that $A_1 x_r(5) + B_1 x_r(5) + f_1 \geq 0$ which would generate an event δ_e at the next time step. As a consequence of the predicted mode switch, the state is reset according to the reset map $f_{\overrightarrow{12}}(x, u)$, i.e., $x(6) = A_{\overrightarrow{12}} x(5) + B_{\overrightarrow{12}} x(5) + f_{\overrightarrow{12}}$.

Proposition 2 *Assume that the event $\delta_e(k)$ does not depend on the continuous input $u_r(k)$, and that the mode $i(k)$ does not depend on the logic input $u_b(k)$. Then a DHA Σ^{a} with predicted resets can be rewritten as a DHA Σ without resets.*

Proof. Let again the superscript a denote the variables of Σ^{a}. By hypothesis, predicted resets imply that $\delta_e^{\mathrm{a}}(k+1)$ only depends on k and $x_r^{\mathrm{a}}(k+1)$, which is a function of $x_r^{\mathrm{a}}(k)$,



(a) A posteriori resets (b) Predictive resets

Figure 1.3: Reset maps

$u_r^{\rm a}(k)$, $i^{\rm a}(k)$. Define the EG for system Σ as $\delta_e(k) = [\delta_e^{\rm a}(k); \delta_e^{\rm a}(k+1)]$, and let $x_b(k) = x_b^{\rm a}(k)$, $u_b(k) = u_b^{\rm a}(k)$, $x_r(k) = x_r^{\rm a}(k)$, $u_r(k) = u_r^{\rm a}(k)$, $y_r(k) = y_r^{\rm a}(k)$, $y_b(k) = y_b^{\rm a}(k)$. Let the FSM for system Σ be equal to the FSM of $\Sigma^{\rm a}$, and define the SAS dynamics of Σ as in the proof of Proposition 1. The MS of Σ should internally compute $j = i^{\rm a}(k+1)$, $h = i^{\rm a}(k)$, compare them, and then choose either the j-th dynamics (if $j = h$, or $j \neq h$ and $f_{\overrightarrow{hj}}^{\rm a}$ is not specified) or the reset dynamics $f_{\overrightarrow{hj}}^{\rm a}$. Since $i^{\rm a}(k) = f_{\rm M}^{\rm a}(x_b^{\rm a}(k), \delta_e^{\rm a}(k))$, $i^{\rm a}(k+1) = f_{\rm M}^{\rm a}(f_{\rm B}^{\rm a}(x_b^{\rm a}(k), u_b^{\rm a}(k), \delta_e^{\rm a}(k)), \delta_e^{\rm a}(k+1))$, it follows that the mode $i(k)$ of the SAS dynamics of system Σ is a function of $x_b(k)$, $u_b(k)$, $\delta_e(k)$. □

Example 2 Let us consider a DHA with two modes:

$$\begin{aligned}
\text{SAS:} \quad & x_r'(k) = \begin{cases} x_r(k) + u_r(k) - 1, & \text{if} \quad i(k) = 1, \\ 2x_r(k), & \text{if} \quad i(k) = 2, \end{cases} \\
\text{EG:} \quad & \delta_e(k) = [x_r(k) \geq 0], \\
\text{MS:} \quad & i(k) = \begin{cases} 1, & \text{if} \quad \delta_e(k) = 0, \\ 2, & \text{if} \quad \delta_e(k) = 1. \end{cases}
\end{aligned}$$

In order to add the predictive reset map $f_{\overrightarrow{12}}(x_r, u_r) = 2$ to the model, we first consider the set $\mathcal{P} = \{x_r, u_r : a_1x_r + b_1u_r + f_1 \geq 0\}$ of all the state/input pairs that will trigger the event δ_e in one step and add an event $\delta_f = a_1x_r + b_1u_r + f_1 \geq 0$ in the EG. If the current mode is $i = 1$ and the pair (x_r, u_r) triggers the event δ_f then the state should be updated according to the reset map. Summing up, we can write the following DHA:

$$\begin{aligned}
\text{SAS:} \quad & x_r'(k) = \begin{cases} x_r(k) + u_r(k) - 1, & \text{if} \quad i(k) = 1, \\ 2x_r(k), & \text{if} \quad i(k) = 2, \\ 2, & \text{if} \quad i(k) = 3, \end{cases} \\
\text{EG:} \quad & \begin{cases} \delta_e(k) = [x_r(k) \geq 0], \\ \delta_f(k) = [x_r(k) + u_r(k) - 1 \geq 0], \end{cases} \\
\text{MS:} \quad & i(k) = \begin{cases} 1, & \text{if} \quad \left[\begin{smallmatrix} \delta_e(k) \\ \delta_f(k) \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right], \\ 2, & \text{if} \quad \delta_e(k) = 1, \\ 3, & \text{if} \quad \left[\begin{smallmatrix} \delta_e(k) \\ \delta_f(k) \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right] \end{cases}
\end{aligned} \tag{1.8}$$

which admits a PWA (3.1) representation (Figure 1.4) that clearly shows that the reset condition is another dynamical mode.

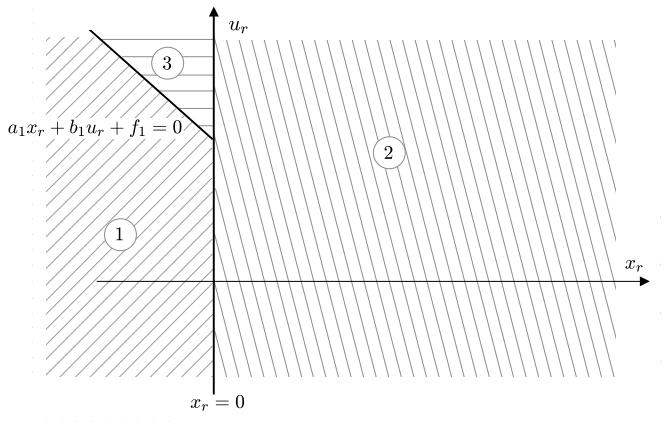


Figure 1.4: Equivalent PWA system

1.6 DHA Trajectories

For a given initial condition $\begin{bmatrix} x_r(0) \\ x_b(0) \end{bmatrix} \in \mathcal{X}_r \times \mathcal{X}_b$, and input $\begin{bmatrix} u_r(k) \\ u_b(k) \end{bmatrix} \in \mathcal{U}_r \times \mathcal{U}_b$, $k \in \mathbb{Z}_{\geq 0}$, the state trajectory $x(k)$, $k \in \mathcal{Z}_{\geq 0}$, of the system is recursively computed as follows:

1. Initialization: $x(0) = \begin{bmatrix} x_r(0) \\ x_b(0) \end{bmatrix}$;
2. Recursion:
   a) $\delta_e(k) = f_{\mathrm{H}}(x_r(k), u_r(k), k)$;
   b) $i(k) = f_{\mathrm{M}}(x_b(k), u_b(k), \delta_e(k))$;
   c) $y_r(k) = C_{i(k)}x_r(k) + D_{i(k)}u_r(k) + g_{i(k)}$;
   d) $y_b(k) = g_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k))$;
   e) $x_r'(k) = A_{i(k)}x_r(k) + B_{i(k)}u_r(k) + f_{i(k)}$;
   f) $x_b'(k) = f_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k))$.

Definition 1 *A DHA is* well-posed *on $\mathcal{X}_r \times \mathcal{X}_b$, $\mathcal{U}_r \times \mathcal{U}_b$, $\mathcal{Y}_r \times \mathcal{Y}_b$, if for all initial conditions $x(0) = \begin{bmatrix} x_r(0) \\ x_b(0) \end{bmatrix} \in \mathcal{X}_r \times \mathcal{X}_b$, and for all inputs $u(k) = \begin{bmatrix} u_r(k) \\ u_b(k) \end{bmatrix} \in \mathcal{U}_r \times \mathcal{U}_b$, for all $k \in \mathbb{Z}_{\geq 0}$, the state trajectory $x(k) \in \mathcal{X}_r \times \mathcal{X}_b$ and output trajectory $y(k) = \begin{bmatrix} y_r(k) \\ y_b(k) \end{bmatrix} \in \mathcal{Y}_r \times \mathcal{Y}_b$ are uniquely defined.*

Definition 1 will be used for other types of hybrid models that we will introduce later. In general a hybrid model may not be well-posed, either because the trajectories stop after a finite time (for instance, the state vector leaves the set $\mathcal{X}_r \times \mathcal{X}_b$) or because of bifurcations (the successor $x_r'(k)$, $x_b'(k)$ may be multiply defined). In general a DHA can be "not well-posed" only if its trajectories are not defined after finite time. This means that DHA can not bifurcate.

2 HYSDEL Models

We designed a modeling language to describe DHA models, called HYbrid System DEscription Language (HYSDEL). In this section we will detail the language capabilities and we will show how DHA systems can be modeled within HYSDEL. As we will explain better in the next section, the HYSDEL description is an abstract modeling step. The associated HYSDEL compiler then translates the description into several computational models, in particular into MLD and PWA form.

2.1 HYSDEL List

A HYSDEL list is composed by two parts INTERFACE and IMPLEMENTATION. The declaration of an empty system is as follows:

```
SYSTEM name {
/* C-style comments */
    INTERFACE {
    }
    IMPLEMENTATION {
    }
}
```

HYSDEL has a syntax based on C-language, it allows C-style comments (`/* ... */`) and uses the curly braces ("`{`" and "`}`") delimiters. All the sections start with the section name in capital letters followed by the content of the section enclosed in curly braces. In the informal description of the grammar that follows, we use `courier` font to specify keywords and language constructs (or terminals tokens), *slanted roman* font to denote user defined content (or intermediate lexical tokens). We denote by the subscript $_{\text{opt}}$ optional parts and by $_{+}$ a declaration that can be repeated one or more times. We will omit the definitions of some tokens, if intuitive. A formal description of the grammar is reported in [39] and the file hys.y of the source distribution contains the bison description of the grammar.

2.1.1 INTERFACE Section

The INTERFACE section contains the declarations, divided into subsections called STATE, INPUT, OUTPUT and PARAMETER, the order in which those sections are declared is not relevant. The first three declarations are referred to as variable declaration, while the last is a parameter declaration. A real variable declaration has the general form: `REAL` *name* `[`*min*`, `*max*`]`$_{\text{opt}}$`;` where the optional declaration `[`*min*`, `*max*`]` contains the bounds of the variable. The type specifier `REAL` can be replaced with the keyword `BOOL` if the variable is Boolean, in such a case the bound declaration is forbidden and defaults to `[0, 1]`. Two or more variable declarations can share the type specifier by using a "`,`" in the declaration: `BOOL` *name1*`, `*name2*$_{+}$`;`.
There are three kinds of parameter declarations:

Boolean parameters: `BOOL parname = value` where `value;` can be either `TRUE` or `FALSE`; any further occurrence of the parameter is then replaced with the corresponding value.

Operator	HYSDEL representation
$\cdot$	`*`
$/$	`/`
$+$	`+`
$-$	`-`
a^b	`a ^ b`
$e^b = \exp(b)$	`exp(b)`
$\sqrt{a}$	`sqrt(a)`
$\log a$	`log(a)`
$\cos(a)$	`cos(a)`
$\sin(a)$	`sin(a)`

Table 2.1: List of operators supported on parameter declarations

Operator	HYSDEL representation
$\wedge$	`&` or `&&`
$\vee$	`|` or `||`
$\Rightarrow$	`->`
$\Leftarrow$	`<-`
$\Leftrightarrow$	`<->`
$\neg$	`~` or `!`

Table 2.2: List of operators supported on Boolean expressions

Real numeric parameters: `REAL parname = number;`, any further occurrence of the parameter is then replaced with the corresponding value.

Real symbolic parameters: `REAL parname;`, the parameter is handled symbolicaly.

Any parameter can be defined in terms of other parameters declared before using the operators and functions in Table 2.1, and can be used thereinafter as it was a number. HYSDEL predefines two real parameters `pi` $= \pi$ and the tolerance `MLD_epsilon` $= 10^{-6}$ used for the relations (3.10c). Both the parameters can be redefined to different values.

Currently HYSDEL supports only scalar data types, there is no support for vectors and matrices.

Before describing in detail the IMPLEMENTATION section, we recall some definitions.

Boolean-expr, affine-expr, linear-expr

A *Boolean-expr* is the combination of Boolean variables and the operators reported in Table 2.2.

An *affine-expr* is linear affine combination of real variables

$$a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n, \tag{2.1}$$

where a_i is a function of parameters, and x_i are real (state, input, output, and auxiliary) variables. If $a_0 = 0$ then (2.1) is a *linear-expr*.

2.1.2 IMPLEMENTATION Section

The second part, IMPLEMENTATION, is composed of specialized sections describing the relations among the variables. The IMPLEMENTATION section starts with an optional AUX section which contains the declarations of the internal signals of the DHA system, called also auxiliary variables. The declaration follows the general syntax of the variable declaration.

OUTPUT Section

The OUTPUT section allows specifying static linear and logic relations for the output vector $y = \left[\begin{smallmatrix} y_r \\ y_b \end{smallmatrix}\right]$.
The general syntax is:

OUTPUT{*output-item*$_+$}

and each *output-item* is one of the following:

var = *affine-expr* ;
var = *Boolean-expr* ;

AD Section

The HYSDEL section AD allows to define Boolean variables from continuous ones, and is based exactly on the same semantics of the event generator (EG) described earlier.
The general syntax is:

AD{*ad-item*$_+$}

and each *ad-item* is one of the following:

var = *affine-expr* <= *real-number* ;
var = *affine-expr* >= *real-number* ;

and the syntax (used by the former version of the HYSDEL compiler)

var = *affine-expr* <= 0 [*min*, *max*, *epsilon*] ;

is still accepted but not recommended because of the error prone specification of the bounds [*min*, *max*, *epsilon*] on *affine-expr*[1]. The variable *epsilon* is the tolerance used for the translation in MLD form according to (3.10c), if omitted, the value of MLD_epsilon is used.

HYSDEL does not provide explicit access to the time instance, however this limitation can be easily overcome by adding a continuous state variable t such that $t' = t + T_s$, where T_s is the sampling time. Examples include level indicator variables, operational alarms, etc.

Example 3 In a water tank with inflow Q, the sensor δ provides the signal 1 if and only if the liquid level $h \geq 0.5h_{\max}$.

```
SYSTEM tank {
   INTERFACE {
      PARAMETER {
         REAL hmax = 1; }
      INPUT {
         REAL h [0, hmax]; }
      OUTPUT {
         BOOL y; }
   } /* end interface */
   IMPLEMENTATION {
      AUX {
         BOOL d; }
      AD {
         d = 0.5 * hmax <= h; }
      OUTPUT {
         y = d;}
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0001.hys.

[1] The old bounds declaration [*max*, *min*, *epsilon*] is still accepted but is deprecated and will raise a warning.

LOGIC Section

The section LOGIC allows to specify arbitrary functions of Boolean variables: In particular the mode selector is a Boolean function and therefore it can be modeled in this section.
The general syntax is:

LOGIC{*logic-item*$_+$}

and each *logic-item* is as follows:

var = *Boolean-expr* ;

Example 4 The passengers of a train can pull the handle of the emergency stop, generating the binary signal $u_{\text{alarm}} = 1$. Usually, this causes an emergency stop of the train, by setting the emergency braking signal $u_{\text{brake}} = 1$. However, the brake should not be activated if fire is detected while the train is in a tunnel. We define the indicator variables for fire as $s_{\text{fire}} = 1$ and tunnel transition as $s_{\text{tunnel}} = 1$. Then we can describe this system as:

$$u_{\text{brake}} = u_{\text{alarm}} \wedge (\neg s_{\text{tunnel}} \vee \neg s_{\text{fire}}) . \tag{2.2}$$

```
SYSTEM train {
   INTERFACE {
      INPUT {
         BOOL alarm, tunnel, fire; }
      OUTPUT {
         BOOL brake;}
   } /* end interface */
   IMPLEMENTATION {
      AUX {
         BOOL decision; }
      LOGIC {
         decision =  alarm & (~tunnel | ~fire); }
      OUTPUT {
         brake = decision; }
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0003.hys.

DA Section

The HYSDEL section DA defines continuous variables according to if-then-else conditions on Boolean variables. This section models part of the switched affine system (SAS), namely the variables z_i defined in (1.2a)–(1.2b).
The general syntax is:

DA{*da-item*$_+$}

each *da-item* is one of the following:

var = { IF *Boolean-expr* THEN *affine-expr* [*min*, *max*, *epsilon*]$_{\text{opt}}$};
var = { IF *Boolean-expr* THEN *affine-expr* [*min*, *max*, *epsilon*]$_{\text{opt}}$
ELSE *affine-expr* [*min*, *max*, *epsilon*]$_{\text{opt}}$};

here, the variable *epsilon* is a placeholder and is introduced for symmetry with the AD section[2]. If the `ELSE` part is omitted, is is assumed to be 0.

Example 5 The command signal u to an electric motor is passed through a nonlinear amplifier, which has two modes of operation: high gain and low gain. In "low gain" the amplifier completely rejects the noise d, while in "high gain" it only attenuates it. The Boolean input $l = 1$ selects the low gain mode. The actual signal u_{comp} applied to the motor is $u_{\text{comp}} = u$ if $l = 1$, otherwise $u_{\text{comp}} = 2.3u + 0.4d$.

```
SYSTEM motor {
   INTERFACE {
      OUTPUT {
         REAL ucomp; }
      INPUT {
         REAL u [0, 10], d [0, 10];
         BOOL l; }
   } /* end interface */
   IMPLEMENTATION {
      AUX {
         REAL unl;}
      DA {
         unl = { IF l THEN u ELSE 2.3*u + 0.4*d};}
      OUTPUT {
         ucomp = unl; }
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0002.hys.

CONTINUOUS Section

The CONTINUOUS section describes the linear dynamics, expressed as difference equations. This section models the remaining Equation (1.2c) of the SAS.
The general syntax is:

`CONTINUOUS{`*cont-item*$_+$`}`

and each *cont-item* has the form:

var `=` *affine-expr* `;`

LINEAR Section

HYSDEL allows also to define a continuous variable as an affine function of continuous variables in the LINEAR section.
The general syntax is:

`LINEAR{`*lin-item*$_+$`}`

and each *lin-item* has the form:

var `=` *affine-expr* `;`

[2]The old bounds declaration `[`*max*`,` *min*`,` *epsilon*`]` is still accepted but is deprecated and will raise a warning.

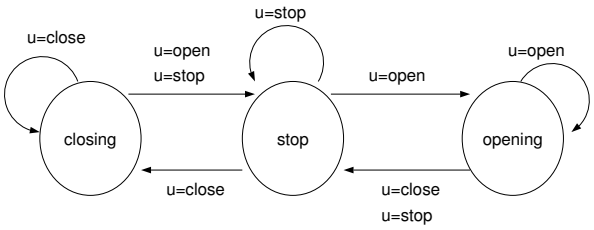


Figure 2.1: Finite state machine of an outflow unit

This section, together with the CONTINUOUS and AD sections allows more flexibility when modeling the SAS. This extra flexibility allows algebraic loops that may render undefined trajectories of the model. The HYSDEL compiler integrates a semantic checker that is able to detect and report such abnormal situations.

AUTOMATA Section

The AUTOMATA section specifies the state transition equations of the finite state machine (FSM) as Boolean functions $x'_b(k) = f_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k))$.
The general syntax is:

AUTOMATA{*automata-item*$_+$}

and each *automata-item* is as follows:

var = *Boolean-expr* ;

Example 6 An outflow unit of a hydroelectric power plant is described by the automaton of Figure 2.1.

```
SYSTEM outflow {
   INTERFACE {
      STATE {
         BOOL closing, stop, opening; }
      INPUT {
         BOOL uclose, uopen, ustop; }
   } /* end of interface */
   IMPLEMENTATION {
      AUTOMATA {
         closing = (uclose & closing) | (uclose & stop);
         stop    = ustop | (uopen & closing) | (uclose & opening);
         opening = (uopen & stop) | (uopen & opening);  }
      MUST {
         ~(uclose & uopen);
         ~(uclose & ustop);
         ~(uopen  & ustop); }
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0006.hys.

MUST Section

The MUST section specifies constraints on continuous and Boolean variables, i.e., linear constraints and Boolean formulas, and therefore it allows defining the sets $\mathcal{X}_r$, $\mathcal{X}_b$, $\mathcal{U}_r$, $\mathcal{U}_b$, $\mathcal{Y}_r$, $\mathcal{Y}_b$ (more generally, the MUST section allows also mixed constraints on states, inputs, and outputs).
The general syntax is:

MUST{*must-item*$_+$}

and each *must-item* is one of the following:

affine-expr <= *affine-expr* ;
affine-expr >= *affine-expr* ;
Boolean-expr ;

Example 7 The level h in a water tank must be nonnegative and below the maximum height h_{max} of the tank, $0 \leq h \leq h_{max}$.

```
SYSTEM watertank {
   INTERFACE {
      STATE {
         REAL h; }
      INPUT {
         REAL Q; }
      PARAMETER {
         REAL hmax = 0.3;
         REAL k    = 1; }
   } /* end interface */
   IMPLEMENTATION {
      CONTINUOUS {
         h = h + k*Q; }
      MUST {
          h - hmax <= 0;
         -h        <= 0; }
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0007.hys.

Symbolic manipulation, and Cast operator

HYSDEL implements a simple symbolic computation tool. The simple symbolic manipulator handles efficiently linear and affine expression but cannot detect that an expression is linear if it involves cancellation on nonlinear terms.

Example 8 HYSDEL will report an error with the following source:

```
SYSTEM outflow {
   INTERFACE {
      OUTPUT {
         REAL y1,y2,y3;}
      INPUT {
         REAL x1,x2,x3; }
   } /* end of interface */
```

```
   IMPLEMENTATION {
      OUTPUT {
         y1 = x1 + 2 * x2 + 3 * x3 + 4 * x1;
              /* OK! = 5 * x1 + 2 * x2 + 3 * x3 */
         y2 = x1 + log(2) * x2 - log(2) * x2;
              /* OK! = x1 */
         y3 = x1 + x2 * x3 - x3 * x2;
              /* ERROR! although linear, HYSDEL will*/
              /* report an error */}
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0009.hys.

A Boolean expression *Boolean-expr* has a conventional value of 1 if true and 0 otherwise, the value of a Boolean expression can be used in affine expressions by using the cast operator: `(REAL` *Boolean-expr* `)`.

Example 9 The flow through a pipe is 10 l/s if both valve v1 and valve v2 are opened, 0 otherwise

```
SYSTEM outflow {
   INTERFACE {
      OUTPUT {
         REAL flow; }
      INPUT {
         BOOL v1, v2; }
   } /* end of interface */
   IMPLEMENTATION {
      OUTPUT {
         flow = 10 * (REAL v1 & v2);}
   } /* end implementation */
} /* end system */
```

This file is available in the distribution as l0008.hys.

2.1.3 HYSDEL Compiler

Once the HYSDEL model of a system is available, the companion HYSDEL compiler is able to generate the equivalent MLD model.

Mixed Logical Dynamical (MLD) Systems

In [12], the authors proposed discrete-time hybrid systems denoted as mixed logical dynamical (MLD) systems. An MLD system is described by the following relations:

$$\begin{aligned}\begin{bmatrix} x_r(t+1) \\ x_b(t+1) \end{bmatrix} &= \begin{bmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} B_{1rr} & B_{1rb} \\ B_{1br} & B_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} \\ &\quad + \begin{bmatrix} B_{2rb} \\ B_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} B_{3rr} \\ B_{3br} \end{bmatrix} z(t), \end{aligned} \tag{2.3a}$$

$$\begin{aligned}\begin{bmatrix} y_r(t) \\ y_b(t) \end{bmatrix} &= \begin{bmatrix} C_{rr} & C_{rb} \\ C_{br} & C_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} D_{1rr} & D_{1rb} \\ D_{1br} & D_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} \\ &\quad + \begin{bmatrix} D_{2rb} \\ D_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} D_{3rr} \\ D_{3br} \end{bmatrix} z(t), \end{aligned} \tag{2.3b}$$

$$E_2 d(t) + E_3 z(t) \leq E_1 \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + E_2 \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + E_5, \tag{2.3c}$$

where $x_r \in \mathbb{R}^{n_{xr}}$, $x_b \in \{0,1\}^{n_{xb}}$, $u_r \in \mathbb{R}^{n_{ur}}$, $u_b \in \{0,1\}^{n_{ub}}$, $y_r \in \mathbb{R}^{n_{yr}}$, $y_b \in \{0,1\}^{n_{yb}}$, $z \in \mathbb{R}^{n_z}$, and $\delta \in \{0,1\}^{n_\delta}$. All the E_i matrices have n_e rows.

A more general form of the MLD systems is:

$$\begin{bmatrix} x_r(t+1) \\ x_b(t+1) \end{bmatrix} = \begin{bmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} B_{1rr} & B_{1rb} \\ B_{1br} & B_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + \begin{bmatrix} B_{2rb} \\ B_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} B_{3rr} \\ B_{3br} \end{bmatrix} z(t) + \begin{bmatrix} B_{5r} \\ B_{5b} \end{bmatrix}, \tag{2.4a}$$

$$\begin{bmatrix} y_r(t) \\ y_b(t) \end{bmatrix} = \begin{bmatrix} C_{rr} & C_{rb} \\ C_{br} & C_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} D_{1rr} & D_{1rb} \\ D_{1br} & D_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + \begin{bmatrix} D_{2rb} \\ D_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} D_{3rr} \\ D_{3br} \end{bmatrix} z(t) + \begin{bmatrix} D_{5r} \\ D_{5b} \end{bmatrix}, \tag{2.4b}$$

$$E_2 d(t) + E_3 z(t) \leq E_1 \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + E_2 \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + E_5, \tag{2.4c}$$

where $x_\bullet$, $u_\bullet$, $y_\bullet$, z, and δ are as before, we denote this variant as extended MLD form. Note that the matrices B_{5c}, B_{5d}, D_{5c}, and D_{5d} do not introduce any further expressivity in the MLD model but are useful to keep the size of the models contained.

The parameters declared in the system form the coefficients of the E matrices. If the system has symbolic parameters, then HYSDEL generates a symbolic output and assumes a structure called *params* width a field with the same name as each symbolic parameter.

Example 10 The interface declaration:

```
INTERFACE {
    PARAMETER {
        REAL p1,p2;
    }
}
```

assumes that the fields *params.p1* and *params.p2* are defined in the structure *params*.

Given the current state $x(k)$ and input $u(k)$, the time-evolution of (2.4) could be determined by solving $\delta(k)$ and $z(k)$ from (2.4c), and then updating $x'(k)$ and $y(k)$ from (2.4a)–(2.4b).

HYSDEL can also produce a function to simulate the hybrid system, this function directly uses the definitions in the HYSDEL source and does not rely on any mixed integer solver.

```
function [xn, d, z, y] = sys_name (x, u, params)
```

x is the current state, *u* the input, and *params* the structure specifying the values for every symbolic parameter and can be omitted if there is no symbolic parameter. The function computes one time step and returns the new state as *xn* and the output as *y*. It also provides the auxiliary variables *d* and *z*. *xn*, *d*, *z*, *y*, *x* and *u* are vectors. The order of the variables corresponds to the order in which the real variables are stored before the Boolean ones, otherwise the order corresponds to the order in which they are declared in the HYSDEL.

Command Line

HYSDEL accepts the following command line options:

`-h,--help` Print help and exit.

`-V,--version` Print version and exit.

`-iSTRING,--input=STRING` HYSDEL input file name (default=stdin).

`-mSTRING,--MLDoutput=STRING` MLD output file name (without extension) (default=stdout).

`-sSTRING,--SIMoutput=STRING` Simulator file name (without extension).

`-p, --parametric` Force all parameters to be handled as symbolic parameters.

`-a, --allow-affine` Allow affine state and output functions. Former version of HYSDEL did not allows constant terms a_0 in the affine state and output functions. This limitation comes directly from equation (2.3a) and (2.3b). The option `-a` overrides that limitation by adding an auxiliary variable $z = a_0$. The output generated is backwards compatible but somehow redundant. See also the next `-5` option.

`-5, --allow-B5-D5` Use Extended MLD form (2.4) (implies -a). Note that the Extended MLD form may produce wrong results with old algorithms.

`--no-symbol-table` Omit the symbol table information.

`--no-row-info` Omit row information.

`--no-params-checks` Omit checks for symbolic parameters.

`--matlab-symbolic` Support for Matlab symbolic toolbox for parameters.

`-vINT, --verbose=INT` Verbosity level (0=silent, 3=max) (default=2).

HYSDEL stores the output in a file which describes a MLD-structure according to Matlab syntax. If the system contains symbolic parameters then the generated file assumes that the structure *params* is defined in the workspace.

In the next section we will present the complete format of the MLD-structure.

2.2 MLD structure

The structure S contains the following fields:

- $Arr = A_{rr}$, $Arb = A_{rb}$, $Abr = A_{br}$, $Abb = A_{bb}$, A = [Arr, Arb; Abr, Abb].
- $B1rr = B_{1rr}$, $B1rb = B_{1rb}$, $B1br = B_{1br}$, $B1bb = B_{1bb}$, $B1$ = [B1rr, B1rb; B1br, B1bb].
- $B2rb = B_{2rb}$, $B2bb = B_{2bb}$, $B2$ = [B2rb; B2bb].
- $B3rr = B_{3rr}$, $B3br = B_{3br}$, $B3$ = [B3rr; B3br].
- $B5r = B_{5r}$, $B5b = B_{5b}$, $B5$ = [B5r; B5b].
- $Crr = C_{rr}$, $Crb = C_{rb}$, $Cbr = C_{br}$, $Cbb = C_{bb}$, C = [Crr, Crb; Cbr, Cbb].
- $D1rr = D_{1rr}$, $D1rb = D_{1rb}$, $D1br = D_{1br}$, $D1bb = D_{1bb}$, $D1$ = [D1rr, D1rb; D1br, D1bb].
- $D2rb = D_{2rb}$, $D2bb = D_{2bb}$, $D2$ = [D2rb; D2bb].
- $D3rr = D_{3rr}$, $D3br = D_{3br}$, $D3$ = [D3rr; D3br].
- $D5r = D_{5r}$, $D5b = D_{5b}$, $D5$ = [D5r; D5b].
- $E1 = E_1$, $E2 = E_2$, $E3 = E_3$, $E4 = E_4$, $E5 = E_5$.

- *nxr* = n_{xr}, *nxb* = n_{xb}, *nx* = *nxr* + *nxb*, *nur* = n_{ur}, *nub* = n_{ub}, *nu* = *nur* + *nub*, *nyr* = n_{yr}, *nyb* = n_{yb}, *ny* = *nyr* + *nyb*, *nd* = n_{δ}, *nz* = n_z, *ne* = n_e.
- *ul, uu* lower and upper bound on u, same for *xl, xu, yl, yu, dl, du*, and *zl, zu*.
- *MLDisvalid* 1 if the model has passed a validity check (sizes of the matrices, null entries,...). User functions can trust this information.
- *name* string containing the system name as specified in the HYSDEL declaration.
- *MLDstructver* integer containing the protocol version, needed for future extentions. All the protocol versions should be backwards compatible. User's function should check if the protocol version is new enough. The protocol version presented in this document is 2. HYSDEL version 1.3.0 and higher support protocol version 2.
- *MLDsymtable* 1 if the structure includes symbol table informations.
- *MLDrowinfo* 1 if the structure includes debugging information for the generated constraints.
- *symtable* Cell array containing the symbols.
  - *name* string containing the name of the symbol
  - *kind* {'x'|'u'|'y'|'z'|'d'|'p'} name of the array containing the variable, note that they are lower case. The letter 'p' stands for parameter, it is just a place holder as there is no parameter vector.
  - *type* {'r'|'b'} type of the variable, note that they are lower case.
  - *index* position of the variable in the array $kind_{type}$. In case of parameters (kind = 'p') the content of this variable is of no relevance.
  - *line_of_declaration* line where the variable was declared in the HYSDEL code. This index is -1 for the predefined parameters.
  - *line_of_first_use* line where the variable was used for the first time in the HYSDEL code.
  - *defined* group of constraints containing the definition of the variable (see the following *rowinfo* entry).
  - *computable_order* priority for computation. This information can be used as branching priority if the mixed integer solver supports this option.
  - *min* minimum of the variable.
  - *min_computed* 1 if HYSDEL computed the minimum.
  - *max* maximum of the variable.
  - *max_computed* 1 if HYSDEL computed the maximum.
  - *value* numerical value of the parameter (this field is present only if kind = 'p').
- *rowinfo* Structure containing the information on the matrices.
  - *state_upd* Cell array containing the information on the rows in the state update matrices (2.4a). The cell array has the same number of items as the number of rows of the matrices, the i-th element of the array provides information on the i-th constraint.

  - *section* Name of the section where the state update was declared.
  - *item_type* Actual section the item belong to.
  - *defines* Name of the variable that is defined.
  - *depends* Cell array of variable names the definition. depends on.
  - *group*, *subgroup*, *subindex* unique identification of the source declaration. Each statement in HYSDEL gets a sequential number, this statement is stored in *group*. A statement may correspond to more than one basic element of HYSDEL (i.e. in case of automatic introduction of variables), each such a constituting element gets an sequential number, stored in *subgroup*. Each element may correspond to more than one inequality or equality constraint, each constraint gets a sequential number. Summing up, all the constraints coming from a HYSDEL statement share the same *group* index, all the constraints coming from a basic element share the same *subgroup* index and within the same *group*,*subgroup* each constraint gets a sequential *subindex*.
  - *source* Code snippet of the original HYSDEL declaration.
  - *sourceline* Line in the source code.
  - *human* Human readable expression.

- *output* Cell array containing the information on the rows in the output matrices (2.4b). As before the i-th element of the array provides information on the i-th constraint and each item has the same fields as in the previous point.
- *ineq* Cell array containing the information on the rows of the inequality constraints (2.4c). As before the i-th element of the array provides information on the i-th constraint. The item has the same fields as described in the previous point plus the following fields:
  - *aff_min* value of m used for the transformations (3.12) and (3.10).
  - *aff_min_computed* 1 if m was computed, 0 if provided.
  - *aff_max* value of M used for the transformations (3.12) and (3.10).
  - *aff_max_computed* 1 if M was computed, 0 if provided.
  - *aff_eps* value of the tolerance ϵ used for the transformations (3.10).
  - *aff_eps_computed* 1 if a default value for ϵ was used, 0 if provided.

3 From DHA to Computational Models

3.1 DHA and Piecewise Affine Systems

This section highlights the relationships between the DHA introduced above and the class of Piecewise Affine (PWA) systems [49].

PWA systems [33] are defined by partitioning the state space into polyhedral regions, and associating with each region a different linear state-update equation

$$\begin{aligned} x'(k) &= A_{i(k)}x(k) + B_{i(k)}u(k) + f_{i(k)}, && \text{(3.1a)} \\ y(k) &= C_{i(k)}x(k) + D_{i(k)}u(k) + g_{i(k)}, && \text{(3.1b)} \\ & i(k) \text{ such that } \; H_{i(k)}x(k) + J_{i(k)}u(k) \leq K_{i(k)}, && \text{(3.1c)} \end{aligned}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n$, $u \in \mathcal{U} \subseteq \mathbb{R}^m$, $y \in \mathcal{Y} \subseteq \mathbb{R}^p$, $\{H_i x + J_i u \leq K_i\}_{i=1}^s$, is a polyhedral partition[1] of the set $\mathcal{X} \times \mathcal{U}$, the matrices A_i, B_i, f_i, C_i, D_i, g_i, H_i, J_i, K_i are constant and have suitable dimensions, the inequality in (3.1c) should be interpreted componentwise. For PWA systems, well-posedness is defined similarly to Definition 1.

Definition 2 *Let Σ_1, Σ_2 be hybrid models, whose inputs are $u_1(k), u_2(k) \in \mathcal{U}$, and outputs $y_1(k), y_2(k) \in \mathcal{Y}$, $k \in \mathbb{Z}_{\geq 0}$. Let $x_1(k) \in \mathcal{X}_1$ be the state of Σ_1 and $x_2(k) \in \mathcal{X}_2$ the state of Σ_2, $k \in \mathbb{Z}_{\geq 0}$. The hybrid model Σ_2 is* equivalent *to Σ_1 on $\mathcal{X}_1$,$\mathcal{U}$,$\mathcal{Y}$, $\Sigma_2 \overset{\mathcal{X}_1,\mathcal{U},\mathcal{Y}}{\rightsquigarrow} \Sigma_1$, if there exists a mapping $T : \mathcal{X}_1 \mapsto \mathcal{X}_2$ such that for all initial conditions $x_1(0) \in \mathcal{X}_1$, and for all $u_1(k) = u_2(k) = u(k) \in \mathcal{U}$, the output trajectories $y_1(k)$ and $y_2(k)$ coincide and $x_2(k) = Tx_1(k)$ at all steps $k \in \mathbb{Z}_{\geq 0}$.*

Lemma 1 *Let Σ_{PWA} be a well-posed PWA model defined on a set of states $\mathcal{X} \subseteq \mathbb{R}^n$, a set of inputs $\mathcal{U} \subseteq \mathbb{R}^m$, and a set of outputs $\mathcal{Y} \subseteq \mathbb{R}^p$. Then there exists a well-posed DHA model Σ_{DHA} such that $\Sigma_{\text{DHA}} \overset{\mathcal{X},\mathcal{U},\mathcal{Y}}{\rightsquigarrow} \Sigma_{\text{PWA}}$ under the identity state transformation $T(x) = x$.*

Proof. Equations (3.1a)–(3.1b) are the modes of the SAS, the constraints $H_i x + J_i u \leq K_i$, $i = 1, \ldots, s$ are the defining hyperplanes $f_{\text{H}}(\cdot)$ of the EG, and the MS is defined by Equation (3.1c), namely if all the events associated to the hyperplanes of $H_j x + J_j u \leq K_j$ are satisfied then $i(k) = j$. □

PWA systems can model a large number of physical processes, such as systems with static nonlinearities, and can approximate nonlinear dynamics via multiple linearizations at different operating points.

[1] The double definition of the state-update function over common boundaries of the partition (the boundaries will also be referred to as *guardlines*) is a technical issue that can be resolved by allowing a part of the inequalities in (3.1) to be strict. However, from a numerical point of view, this issue is not relevant.

3.2 DHA and Mixed Logical Dynamical Systems

This section describes how to transform a DHA into linear mixed integer equations and inequalities, by generalizing several results already appeared in the literature [12, 16, 35, 43, 44, 53], and the equivalence between DHA and Mixed Logical Dynamical (MLD) systems [12].

3.2.1 Logical Functions

Boolean functions can be equivalently expressed by inequalities. This technique allows us to translate both the LOGIC and AUTOMATA sections of a HYSDEL model into inequalities.

In order to introduce our notation, we recall here some basic definitions of Boolean algebra. A variable X is a *Boolean variable* if $X \in \{0, 1\}$. A *Boolean expression* is inductively defined[2] by the grammar

$$\phi ::= X|\neg\phi_1|\phi_1 \vee \phi_2|\phi_1 \oplus \phi_2|\phi_1 \wedge \phi_2|\phi_1 \leftarrow \phi_2|\phi_1 \rightarrow \phi_2|\phi_1 \leftrightarrow \phi_2|(\phi_1) \tag{3.2}$$

where X is a Boolean variable, and the logic operators $\neg$ (not), $\vee$ (or), $\wedge$ (and), $\leftarrow$ (implied by), $\rightarrow$ (implies), $\leftrightarrow$ (iff) have the usual semantics. A Boolean expression is a *conjunctive normal form* (CNF) or *product of sums* if it can be written according to the following grammar:

$$\phi ::= \psi|\phi \wedge \psi \tag{3.3}$$

$$\psi ::= \psi_1 \vee \psi_2|\neg X|X \tag{3.4}$$

where ψ are called *terms of the product*, and X are the *terms of the sum* ψ. A CNF is minimal if it has the minimum number of terms of product and each term has the minimum number of terms of sum. Every Boolean expression can be rewritten as a minimal CNF.

A Boolean expression f will be also called *Boolean function* when is used to define a literal X_n as a function of $X_1, \ldots, X_{n-1}$:

$$X_n = f(X_1, X_2, \ldots, X_{n-1}). \tag{3.5}$$

More in general, we can define relations among Boolean variables $X_1, \ldots, X_n$ through a *Boolean formula*

$$F(X_1, \ldots, X_n) = 1, \tag{3.6}$$

where $X_i \in \{0, 1\}$, $i = 1, \ldots, n$. Note that each Boolean function is also a Boolean formula, but not vice versa. Boolean formulas can be equivalently translated into a set of integer linear inequalities. For instance, $X_1 \vee X_2 = 1$ is equivalent to $X_1 + X_2 \geq 1$ [53]. The translation can be performed either using an *symbolical* method or a *geometrical* method.

Symbolical Method

The symbolical method consists of first converting (3.5) or (3.6) into CNF, a task that can be performed automatically by using one of the several techniques available, see e.g. [42, 45]. Let the CNF have the form $\bigwedge_{j=1}^{m} \left(\bigvee_{i \in P_j} X_i \bigvee \bigvee_{i \in N_j} \neg X_i \right)$, $N_j, P_j \subseteq \{1, \ldots, n\}$ $\forall j = 1, \ldots, m$. Then, the corresponding set of integer linear inequalities is

$$\begin{cases} 1 \leq \sum_{i \in P_1} X_i + \sum_{i \in N_1} (1 - X_i), \\ \vdots \\ 1 \leq \sum_{i \in P_m} X_i + \sum_{i \in N_m} (1 - X_i). \end{cases} \tag{3.7}$$

[2] In the sake of simplicity we are neglecting precedence.

With these inequalities we can define the set P_{CNF} for any Boolean formula F as:

$$P_{CNF} = \{x \in [0,1]^n : (3.7) \text{ are satisfied with } x = [X_1, \ldots, X_n]^T\}. \tag{3.8}$$

Geometrical Method

The geometrical method consists of two steps (see e.g. [41]). First, the set of points satisfying (3.5) or (3.6) is computed (for this reason, the method was also called *truth table* method in [41]). Each row of the truth table is associated with a vertex of the hypercube $\{0,1\}^n$. The vertices are collected in a set V of *valid* points, all the other points $\{0,1\}^n \setminus V$ are called *invalid*. The inequalities representing the Boolean formula are obtained by computing the convex hull of V, for which several tools are available (see e.g. [29]). We therefore define

$$P_{CH} = \{x \in [0,1]^n : x \in \text{conv}(V)\} \tag{3.9}$$

In general it holds that $P_{CH} \subseteq P_{CNF}$, since $\text{conv}(V)$ is the smallest set containing all integer feasible points. However, there exist Boolean formulas, for which $P_{CH} \neq P_{CNF}$[3]. Conditions for which $P_{CH} = P_{CNF}$ are currently a topic of research.

3.2.2 Continuous-Logic Interfaces

Events of the form (1.3) can be equivalently expressed as

$$\begin{aligned} f^i_{\mathrm{H}}(x_r(k), u_r(k), k) &\leq M^i(1 - \delta^i_e), && \text{(3.10a)} \\ f^i_{\mathrm{H}}(x_r(k), u_r(k), k) &> m^i \delta^i_e, \qquad i = 1, \ldots, n_e, && \text{(3.10b)} \end{aligned}$$

where M^i, m^i are upper and lower bounds, respectively, on $f^i_{\mathrm{H}}(x_r(k), u_r(k), k)$. As we will point out in Section 3.2.4, sometimes from a computational point of view, it may be convenient to have a system of inequalities without strict inequalities. In this case we will follow the common practice [53] to replace the strict Inequality (3.10b) as

$$f^i_{\mathrm{H}}(x_r(k), u_r(k), k) \geq \epsilon + (m - \epsilon)\delta \tag{3.10c}$$

where ϵ is a small positive scalar, e.g. the machine precision, although the equivalence does not hold for $0 \leq f^i_{\mathrm{H}}(x_r(k), u_r(k), k) < \epsilon$.

The most common *logic to continuous* interface is the if-then-else construct

$$\text{IF } \delta \text{ THEN } z = a_1^T x + b_1^T u + f_1 \text{ ELSE } z = a_2^T x - b_2^T u + f_2 \tag{3.11}$$

which can be translated into [16]

$$\begin{aligned} (m_2 - M_1)\delta + z &\leq a_2 x + b_2 u + f_2, && \text{(3.12a)} \\ (m_1 - M_2)\delta - z &\leq -a_2 x - b_2 u - f_2, && \text{(3.12b)} \\ (m_1 - M_2)(1 - \delta) + z &\leq a_1 x + b_1 u + f_1, && \text{(3.12c)} \\ (m_2 - M_1)(1 - \delta) - z &\leq -a_1 x - b_1 u - f_1, && \text{(3.12d)} \end{aligned}$$

where M_i, m_i are upper and lower bounds on $a_i x + b_i u + f_i$, $i = 1, 2$, $\delta \in \{0, 1\}$, $z \in \mathbb{R}$, $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$. Note that (3.11)–(3.12) are a generalization of the real product $z = \delta \cdot (ax + bu + f)$ described in [53].

[3] For example $(X_1 \vee X_2) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3)$.

3.2.3 Continuous Dynamics

As already mentioned, we will deal with dynamics described by linear affine difference equations

$$x'(k) = \sum_{i=1}^{s} z_i(k). \tag{3.13}$$

3.2.4 Mixed Logical Dynamical (MLD) Systems

In [12], the authors proposed discrete-time hybrid systems denoted as mixed logical dynamical (MLD) systems. An MLD system is described by the following relations:

$$x'(k) = Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k) + B_5, \tag{3.14a}$$
$$y(k) = Cx(k) + D_1u(k) + D_2\delta(k) + D_3z(k) + D_5, \tag{3.14b}$$
$$E_2\delta(k) + E_3z(k) \leq E_1u(k) + E_4x(k) + E_5, \tag{3.14c}$$
$$\tilde{E}_2\delta(k) + \tilde{E}_3z(k) < \tilde{E}_1u(k) + \tilde{E}_4x(k) + \tilde{E}_5. \tag{3.14d}$$

where $x \in \mathbb{R}^{n_r} \times \{0,1\}^{n_b}$ is a vector of continuous and binary states, $u \in \mathbb{R}^{m_r} \times \{0,1\}^{m_b}$ are the inputs, $y \in \mathbb{R}^{p_r} \times \{0,1\}^{p_b}$ the outputs, $\delta \in \{0,1\}^{r_b}$, $z \in \mathbb{R}^{r_r}$ represent auxiliary binary and continuous variables, respectively, and A, B_1, B_2, B_3, C, D_1, D_2, D_3, E_1,...,E_5 and $\tilde{E}_1$,...,$\tilde{E}_5$ are matrices of suitable dimensions. Given the current state $x(k)$ and input $u(k)$, the time-evolution of (3.14) is determined by solving $\delta(k)$ and $z(k)$ from (3.14c)–(3.14d), and then updating $x'(k)$ and $y(k)$ from (3.14a)–(3.14b). The equations and inequalities obtained with methods presented in Sections 3.2.1, 3.2.2, 3.2.3 can be represented using the MLD framework. When the problems of synthesis and analysis of MLD models are tackled by optimization techniques, it is convenient to replace the strict inequalities as in (3.10c). We will therefore consider an MLD model where the matrices $\tilde{E}_1$,...,$\tilde{E}_5$ are embedded in (3.14c) as nonstrict inequalities. For MLD systems, well-posedness is defined similarly to Definition 1.

Lemma 2 *Let Σ_{DHA} be a well-posed DHA model defined on a set of states $\mathcal{X} \subseteq \mathbb{R}^n$, a set of inputs $\mathcal{U} \subseteq \mathbb{R}^m$, and a set of outputs $\mathcal{Y} \subseteq \mathbb{R}^p$. Then there exists a well-posed MLD model Σ_{MLD} such that $\Sigma_{\text{MLD}} \overset{\mathcal{X},\mathcal{U},\mathcal{Y}}{\rightsquigarrow} \Sigma_{\text{DHA}}$ under the identity state transformation $T(x) = x$.*

Proof. Directly follows from Sections 3.2.1, 3.2.2, 3.2.3. □

3.3 Other Computational Models

In the previous section we showed the equivalence relations between DHA and PWA and MLD systems. In this section, we review other existing models of linear hybrid systems and show further relationships with DHA.

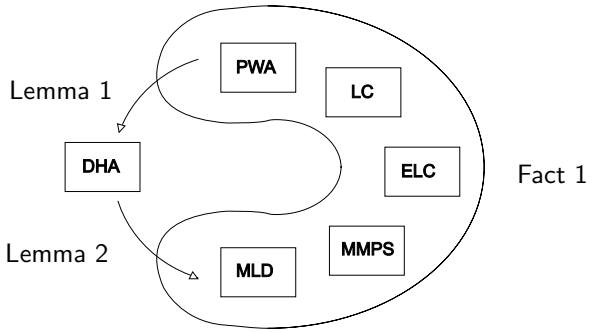


Figure 3.1: Conceptual scheme for the proof of Theorem 1

3.3.1 Linear Complementarity (LC) Systems

Linear complementarity (LC) systems are given in discrete-time by the equations

$$\begin{aligned} x'(k) &= Ax(k) + B_1u(k) + B_2w(k), && \text{(3.15a)} \\ y(k) &= Cx(k) + D_1u(k) + D_2w(k), && \text{(3.15b)} \\ v(k) &= E_1x(k) + E_2u(k) + E_3w(k) + E_4, && \text{(3.15c)} \\ 0 \le v(k) &\perp w(k) \ge 0 && \text{(3.15d)} \end{aligned}$$

with $v(k), w(k) \in \mathbb{R}^q$ and where $\perp$ denotes the orthogonality of vectors (i.e. $v(k) \perp w(k)$ means that $v^T(k)w(k) = 0$). We call $v(k)$ and $w(k)$ the complementarity variables. A, B_i, C, D_i and E_i are real matrices [20,31,32,46,52].

In [33] the relationships among the model classes mentioned above and two others, *min-max-plus-scaling* (MMPS) and *extended linear complementarity* (ELC) systems, were discussed. As ELC systems are of similar nature as LC systems, we will not define them here, but refer to [24,33]. MMPS systems are obtained by choosing the state-update function, the output function, and constraints as (nested) combinations of the operations maximization, minimization, addition and scalar multiplication. More detail on this class can be found in [25,33].

Fact 1 *PWA, MLD, LC, ELC, and MMPS models are equivalent classes of hybrid models (certain equivalences require assumptions on the boundedness of input, state, and auxiliary variables or on well-posedness).*

Proof. See [33] for full detail on assumptions, relationships, and a constructive proof. □

Theorem 1 *Let $\mathcal{X}$, $\mathcal{U}$, $\mathcal{Y}$ be sets of states, inputs, and outputs respectively, and assume that $\mathcal{X}$, $\mathcal{U}$ are bounded. Then DHA, PWA, MLD, LC, ELC, and MMPS well-posed models are equivalent to each other on $\mathcal{X}$, $\mathcal{U}$, $\mathcal{Y}$.*

Proof. By referring to Figure 3.1, mutual equivalences among PWA, MLD, LC, ELC, and MMPS on bounded $\mathcal{X}, \mathcal{U}, \mathcal{Y}$ follows from Fact 1. The equivalence $\Sigma_{\text{DHA}} \overset{\mathcal{X},\mathcal{U},\mathcal{Y}}{\rightsquigarrow} \Sigma_{\text{PWA}}$ follows by Lemma 1, while the equivalence $\Sigma_{\text{MLD}} \overset{\mathcal{X},\mathcal{U},\mathcal{Y}}{\rightsquigarrow} \Sigma_{\text{DHA}}$ follows by Lemma 2. Therefore, any equivalence relation can be stated for any ordered pairs of models.

□

Thanks to the equivalences mentioned above, it is clear that HYSDEL is a tool that allows generating several different hybrid models of a given hybrid system. In particular, HYSDEL generates MLD models, which can be immediately (and efficiently) translated into PWA systems [7], or LC/ELC/MMPS systems using the constructive methods reported in [30,33]. Note that by Propositions 1 and 2 also DHA models with resets are equivalent to any of the other classes of hybrid models.

4 Applications

In this paper we introduced Discrete Hybrid Automata as a general modeling framework for obtaining hybrid models oriented to the solution of analysis and synthesis problems. The language HYSDEL describes DHA at a high level and its associated compiler generates the corresponding computational models. This simplifies the use of the whole theory and set of tools available for different classes of hybrid systems for solving control, state estimation and verification problems.

The effectiveness of HYSDEL was shown on an automotive case study in [26]. In [16] HYSDEL was used to model a batch evaporator plant and the associate PLC controller. HYSDEL has been successfully used in several industrial applications. In [18] the authors modeled the hybrid behavior of a vehicle/tyre system and designed a traction controller that helps the driver to control a vehicle under adverse external conditions such as wet or icy roads. Another automotive application was presented in [9], where the focus is on the application of hybrid modeling and optimal control to the problem of air-to-fuel ratio and torque control in advanced gasoline direct injection stratified charge (DISC) engines. In both cases, the control design leaded to a control law that can be implemented on automotive hardware as a piecewise affine function of the measured and estimated quantities. In [27] the economic optimization of a combined cycle power plant was accomplished by modeling the system in HYSDEL (turning on/off the gas and steam turbine, operating constraints, different modalities start up of the turbines), and then using the generated MLD model in a mixed integer linear optimization algorithm [36].

The latest version of the HYSDEL compiler is available on-line at `http://control.ethz.ch/~hybrid/hysdel`.

A Errors and Warnings

Syntactic errors are reported verbatim from bison. All other errors are listed in Table A.1. Warnings are stated in Table A.2.

error message	location
left hand side variable xxx is not auxiliary	AD, DA, LINEAR, LOGIC
left hand side variable xxx is not state	AUTOMATA, CONTINUOUS
left hand side variable xxx is not output	OUTPUT
left hand side variable xxx is not Boolean	AD, AUTOMATA, LOGIC, OUTPUT
left hand side variable xxx is not real	CONTINUOUS, OUTPUT, DA, LINEAR
recursive definition of variable xxx	AD, DA, LINEAR, LOGIC
expression must be affine	AD, CONTINUOUS, MUST, OUTPUT, LINEAR
then-expression must be affine	DA
else-expression must be affine	DA
constant term in CONTINUOUS	CONTINUOUS
constant term in OUTPUT	OUTPUT
division by zero	Expr
log of non positive	Expr
sqrt of negative	Expr
MLD_epsilon required but is not a parameter	AD
Logic expression in equality is always true, resulting in a constant term	AUTOMATA, OUTPUT
bounds must be constant	AD, DA
epsilon must be constant	AD
parameter xxx is defined using variable yyy	PARAMETER
expression contains output variable	Expr
variable xxx redefined	
failed to compute bounds: no definition for variable xxx	AD, DA
failed to compute bounds: circular definition of variable xxx	AD, DA

Table A.1: Errors generated by HYSDEL

warning message	location
Upper bound not tight	AD, DA
Lower bound not tight	AD, DA
Variable xxx is never used	
Parameter xxx is never used	
Variable xxx is never defined	

Table A.2: Warnings generated by HYSDEL

B Matlab Functions

HYSDEL comes with a number of Matlab functions. This section reports verbatim the Matlab help for those commands.

B.1 Main Routines

B.1.1 hysdel

```
% hysdel(filename,simname,options)
% Compiles the HYSDEL list <filename.hys>,
% generates the M-file <filename.m>
%
% INPUT:
% filename: the hysdel source
% simname : if not empty, generates the HYSDEL simulator (see manual)
% options : a string of space separated command line switches to append to
%           HYSDEL call (see HYSDEL manual, it includes: -p -a -5
%           --no-symbol-table --no-row-info --no-params-checks
%           --matlab-symbolic -v[0-3])
%
% OUTPUT:
% filename.m and simname.m on disk
%
% (C) 2000--2002 F.D. Torrisi,
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% torrisi@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.2 hybsim

```
% [XX,DD,ZZ,YY]=HYBSIM(X0,UU,FILE,PARS,OPT)
% Simulates a hybrid system
% if sys is a simulator-file-name, uses the Matlab simulator (see HYSDEL Manual)
% if sys is a MLDstructure, uses MLDSIM which solves a Mixed Integer Program
% (see MLDSIM)
%
% INPUT:
% X0  : is the initial condition;
% UU  : is the input vector u(t) = UU(:,t);
% SYS : system either the filename of the Matlab simulator
%               or the Name of the MLD struct
% PARS: is the parameters structure passed to the one step
%               Matlab simulator (see HYSDEL Manual, Simulator)
% OPT : are the options passed to the one step MLDsimulator MLDSIM (see MLDSIM)
%
% OUTPUT:
% XX  : is the state trajectory
% DD  : is the auxiliary bool variables trajectory
% ZZ  : is the auxiliary real variables trajectory
% YY  : is the output trajectory
%
```

```
% REMARK:
% using the matlab simulator is much faster. The results produced by the two
% approaches should be the same.
%
% SEE:
% matlab simulator (in the HYSDEL manual, simulator)
% MLDSIM
% PANMIP
% MIQP
%
% (C) 2002 F.D. Torrisi
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% torrisi@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.3 hylink

```
%HYLINK: HYSDEL models in simulink via S-function
%
% use: drag a simulink block of type 'S-function' in your simulink model
%      set the function name to hylink and the filename as parameter.
%      The optional parameter x0 sets the initial state.
%      Use multiplex/demultiplex to access all the inputs/outputs
%
% (C) 2000--2002 F.D. Torrisi,
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% torrisi@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.4 syminfo

```
% info = syminfo(S, vname, vtype, vkind, vindex)
% extract infos from the symble table
% look up the symble table for entries matching the query
% item.name == name AND item.type == type AND item.kind == kind
% empty field ('') matches all entries
%
% INPUT:
% S     : the system name
% vname : var name as in the Hysdel source
% vtype : var type ('b','r')
% vkind : var kind ('x','u','d','z')
% vindex: var index
%
% OUTPUT
% info  : cell array of info records saisfying the query
%
% REMARK
% the contents of the syminfo records are detailed in the HYSDEL manual in the
% section describing the MLD-structure
%
% (C) 2001 by F.D. Torrisi
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% torrisi@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.5 gen_log

```
% gen_log(S, log)
```

```
% generates log-file of all variables
%
% INPUT:
% S  : structure containing the MLD-model
% log: structure containing the log-data of all variables (d, z, u, x, y)
%      obtained by runing 'hybsim.m' i.e.: log.x has the dimensions
%      nx times nm (nx=number of states, nm=number of time-steps)
%
% OUTPUT:
% a logfile ''systemname''.log on disk
%
% (C) 2002 Tobias Geyer
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% geyer@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.6 mldsim

```
% function [xt1, dt, zt, yt, fl, eps] = mldsim( m, xt, ut, Options )
% Simulates an MLD System. Given the MLD system and the current state and
% input, computes the next state and the current output.
%
% INPUT:
% m      : MLD system
% xt     : current state
% ut     : current input
% Options: 1. Specify with Options.solver the miqp solver desired, e.g. like:
%             Options.solver      = 'miqp';
%             SEE PANMIP
%          2. Specify options for the choosen solver e.g. like:
%             Options.miqp.solver = 'linprog';
%             SEE PANMIP
%          3. Specify options for the simulation
%             Options.large        : specifies the magnitude of the box bounds
%                                    for the solver
%          4. Specify options to debug models (see implementation for details)
%             Options.relaxIneq = 1: relax inequalities to maintain feasibility
%             Options.relaxU = 1   : relax inputs
%
% OUTPUT:
% xt1    : next time step
% dt     : delta vector
% zt     : zeta vector
% yt     : output vector
% fl     : flags from the MIP solver
% eps    : epsilon needed in relaxation (see implementation for details)
%
% REMARK:
% Requires panmip as solver interface and some solver supported by panmip
%
% (C) 1998--2002 D. Mignone
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% mignone@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.7 hysdel_old

```
% hysdel_old(filename)
% THIS FILE GENERATES THE DEPRECATED MLD STRUCTURE 1.0
% IT IS PROVIDED FOR BACKWARDS COMPATIBILITY.
```

```
%
% INPUT:
% filename: the hysdel source
%
% OUTPUT:
% filename.mat on disk
%
% REMARK:
% the M-file <hout.m> contains the system in the new format. Use
% load filename.mat if you want to access the system in the old format.
%
% (C) 2000--2002 by F.D. Torrisi, A. Bemporad, T. Geyer
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% torrisi|bemporad|geyer@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.1.8 new2old

```
% Sold = new2old(S)
% THIS FILE TRANSLATES THE MLD STRUCTURE INTO THE DEPRECATED MLD STRUCTURE 1.0
% IT IS PROVIDED FOR BACKWARDS COMPATIBILITY.
%
% INPUT:
% S   : a MLD-structure 2.0
%
% OUTPUT:
% Sold: a MLD-structure 1.0
%
% (C) 2002 by F.D. Torrisi
% Automatic Control Laboratory, ETH Zentrum, CH-8092 Zurich, Switzerland
% torrisi@aut.ee.ethz.ch
%
% see license.txt for the terms and conditions.
```

B.2 Support Routines

This is the list of additional routines that come with HYSDEL: mld2mat.m (extracts matrices from the MLD structure), panmip.m (common MIP interface), miqp.m (free MIQP solver).

B.3 Contributed Routines

The subdirectory contrib of the HYSDEL distribution, contains routines contributed by the users: pwa2hys.m (Piecewise affine to HYSDEL), con_list6.m (adjacency map for a polyhedral partition)

Bibliography

[1] R. Alur, C. Belta, F. Ivančić, V. Kumar, M. Mintz, G.J. Pappas, H. Rubin, and J. Schug. Hybrid modeling and simulation of biomolecular networks. In *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 19–33. Springer Verlag, 2001.

[2] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer Verlag, 1993.

[3] P.J. Antsaklis. A brief introduction to the theory and applications of hybrid systems. *Proc. IEEE, Special Issue on Hybrid Systems: Theory and Applications*, 88(7):879–886, July 2000.

[4] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In O. Maler, editor, *Hybrid and Real-Time Systems*, volume 1201 of *Lecture Notes in Computer Science*, pages 346–360. Springer Verlag, 1997.

[5] A. Balluchi, L. Benvenuti, M. Di Benedetto, C. Pinello, and A. Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: Challenges and opportunities. *Proc. IEEE Vol. 88, No. 7, July 2000*, 88(7):888–912, 2000.

[6] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–162. Springer Verlag, 2001.

[7] A. Bemporad. An efficient technique for translating mixed logical dynamical systems into piecewise affine systems. In *Proc. 41th IEEE Conf. on Decision and Control*, Submitted CDC'02.

[8] A. Bemporad, G. Ferrari-Trecate, and M. Morari. Observability and controllability of piecewise affine and hybrid systems. *IEEE Transactions on Automatic Control*, 45(10):1864–1876, 2000.

[9] A. Bemporad, N. Giorgetti, I.V. Kolmanovsky, and D. Hrovat. A hybrid systems approach to modeling and optimal control of disc engines. In *Proc. 41th IEEE Conf. on Decision and Control*, Submitted CDC'02.

[10] A. Bemporad, L. Giovanardi, and F.D. Torrisi. Performance driven reachability analysis for optimal scheduling and control of hybrid systems. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 969–974, Sydney, Australia, December 2000.

[11] A. Bemporad, D. Mignone, and M. Morari. Moving Horizon Estimation for Hybrid Systems and Fault Detection. In *Proceedings of the American Control Conference*, 1999.

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

[13] A. Bemporad, J. Roll, and L. Ljung. Identification of hybrid systems via mixed-integer programming. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 786–792, 2001.

[14] A. Bemporad, F.D. Torrisi, and M. Morari. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In *Proceedings 3rd International Workshop on Hybrid Systems, Pittsburgh, PA, USA*, volume 1790 of *Lecture Notes in Computer Science*, pages 45–58. Springer Verlag, 2000.

[15] A. Bemporad, F.D. Torrisi, and M. Morari. Performance analysis of piecewise linear systems and model predictive control systems. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 4957–4962, Sydney, Australia, December 2000.

[16] A. Bemporad, F.D. Torrisi, and M. Morari. Discrete-time hybrid modeling and verification of the batch evaporator process benchmark. *European Journal of Control*, 7(4):382–399, 2001.

[17] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, Y. Wang, and C. Weise. New Generation of Uppaal. In *Int. Workshop on Software Tools for Technology Transfer*, June 1998.

[18] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat. A hybrid approach to traction control. In *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 162 – 174. Springer Verlag, 2001.

[19] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[20] M.K. Çamlıbel, W.P.M.H. Heemels, and J.M. Schumacher. Consistency of a time-stepping method for a class of piecewise linear networks. To appear in *IEEE Trans. Circuits Syst. – I*, 2002.

[21] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[22] T. Dang and O. Maler. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*, pages 96–109. Springer Verlag, 1998.

[23] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The Tool Kronos. In *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*. Springer, 1997.

[24] B. De Schutter and B. De Moor. The extended linear complementarity problem and the modeling and analysis of hybrid systems. In P. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry, editors, *Hybrid Systems V*, volume 1567 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 1999.

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

[26] A. Bemporad F. Torrisi. Hysdel — a tool for generating computational hybrid models for analysis and synthesis problems. Technical Report AUT02-03, ETH, Zurich, March 2002.

[27] G. Ferrari-Trecate, E. Gallestey, P. Letizia, M. Spedicato, M. Morari, and M. Antoine. Modeling and control of co-generation power plants: A hybrid system approach. In *Hybrid Systems Computation and Control*, volume To appear of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[28] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. In A. Sangiovanni-Vincentelli and M.D. Di Benedetto, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 218–231. Springer Verlag, 2001.

[29] K. Fukuda. *cdd/cdd+ Reference Manual*. Institute for Operations Research, ETH-Zentrum, CH-8092 Zurich, Switzerland, 0.61 (cdd) 0.75 (cdd+) edition, December 1997.

[30] W.P.H.M Heemels, B. de Schutter, and A. Bemporad. On the equivalence of classes of hybrid dynamical models. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 364–369, Orlando, Florida, 2001.

[31] W.P.M.H. Heemels. *Linear complementarity systems: a study in hybrid dynamics*. PhD thesis, Dept. of Electrical Engineering, Eindhoven University of Technology, The Netherlands, 1999.

[32] W.P.M.H. Heemels, J.M. Schumacher, and S. Weiland. Linear complementarity systems. *SIAM Journal on Applied Mathematics*, 60(4):1234–1269, 2000.

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

[34] J. P. Hespanha, S. Bohacek, K. Obraczka, and J. Lee. Hybrid modeling of TCP congestion control. In A. Sangiovanni-Vincentelli and M.D. Di Benedetto, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 291–304. Springer Verlag, 2001.

[35] T. Huerlimann. *Reference Manual for the LPL Modeling Language, Version 4.42*. Departement for Informatics, Université de Fribourg, Switzerland, `http://www2-iiuf.unifr.ch/tcs/lpl/TonyHome.htm`, 2001.

[36] ILOG, Inc. *CPLEX 7.1 User Manual*. Gentilly Cedex, France, 2001.

[37] K. H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry. On the regularization of Zeno hybrid automata. *System & Control Letters*, 38:141–150, 1999.

[38] M. Johansson and A. Rantzer. Computation of Piecewise Quadratic Lyapunov Functions for Hybrid Systems. *IEEE Transactions on Automatic Control*, 43(4):555–559, April 1998.

[39] D. Jost and Torrisi F. Hysdel 2.0.5 - programmer manual. Technical Report AUT02-28, ETH, Zurich, August 2002.

[40] P. Julián, M. Jordan, and A. Desages. Canonical piecewise-linear approximation of smooth functions. *IEEE Trans. Circuits Syst. I*, 45(5):567–571, May 1998.

[41] D. Mignone, A. Bemporad, and M. Morari. A Framework for Control, Fault Detection, State Estimation and Verification of Hybrid Systems. *Proceedings of the American Control Conference*, 1999.

[42] R. E. Miller. *Switching Theory. Volume 1: Combinational Circuits*. John Wiley, 1965.

[43] G. Mitra, C. Lucas, and Moody S. Tool for reformulating logical forms into zero-one mixed integer programs. *European Journal of Operational Research*, 71:262–276, 1994.

[44] R. Raman and I.E. Grossmann. Relation between MILP modeling and logical inference for chemical process synthesis. *Computers and Chemical Engineering*, 15(2):73–84, 1991.

[45] T. Sasao. *Switching Theory for Logic Synthesis*. Kluwer Academic, 1999.

[46] A.J. van der Schaft and J.M. Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251 of *Lecture Notes in Control and Information Sciences*. Springer, London, 2000.

[47] B.I. Silva, B. Richerson, B. Krogh, and Chutinan A. Modeling and verifying hybrid dynamic systems using Checkmate. In *Proc. 4^{th} Int. Conf. on Automation of Mixed Processes*, pages 323–328, 2000.

[48] B.I. Silva, O. Stursberg, B.H. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 2867–2874, 2001.

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

[50] E.D. Sontag. Interconnected automata and linear systems: A theoretical framework in discrete-time. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III - Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 436–448. Springer-Verlag, 1996.

[51] O. Stursberg, S. Kowalewsky, J. Preussig, and Treseler H. Block-diagram based modelling and analysis of hybrid processes under discrete control. *Journal Europ. des Syst. Automatises*, 32(9-10):1097–1118, 1998.

[52] A.J. van der Schaft and J.M. Schumacher. Complementarity modelling of hybrid systems. *IEEE Transactions on Automatic Control*, 43:483–490, 1998.

[53] H.P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, Third Edition, 1993.

Index