HYSDEL—A Tool for Generating Computational Hybrid Models for Analysis and Synthesis Problems

Fabio Danilo Torrisi and Alberto Bemporad, *Member, IEEE*

***Abstract*—This paper presents a computational framework for modeling hybrid systems in discrete-time. We introduce the class of discrete hybrid automata (DHA) and show its relation with several other existing model paradigms: piecewise affine systems, mixed logical dynamical systems, (extended) linear complementarity systems, min-max-plus-scaling systems. We present HYSDEL (hybrid systems description language), a high-level modeling language for DHA, and a set of tools for translating DHA into any of the former hybrid models. Such a multimodeling capability of HYSDEL is particularly appealing for exploiting a large number of available analysis and synthesis techniques, each one developed for a particular class of hybrid models. An automotive example shows the modeling capabilities of HYSDEL and how the different models allow to use several computational tools.**

***Index Terms*—Analysis, hybrid systems, modeling, optimal control, verification.**

I. INTRODUCTION

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 [1]. 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 prespecified thresholds.

The practical relevance of hybrid models is twofold. The profitability of logic controllers embedded in a continuous environment is increasing the demand for adequate modeling, analysis and design tools, for instance in the automotive industry [1]. Moreover, many physical phenomena admit a natural hybrid description, like circuits integrating relays or diodes, biomolecular networks [2], and TCP/IP networks in [3].

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 behavior like Zeno effects [4], that complicate the definition of trajectories. Although simulation allows one to probe the model, it certainly does not permit structural properties of the model to be assessed. 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 [5]. Reachability analysis relies on a reach set computation algorithm, which is strongly related to the mathematical model of [6].

Piecewise quadratic Lyapunov stability [7] 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 there 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 choosing 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 [8]. 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 (see [6] and the references contained therein) and have been widely used in the literature. The starting point for both models

Manuscript received May 21, 2002; received in final form April 1, 2003. Recommended by Associate Editor P. J. Mosterman. This work supported in part by the European Union through the IST project "Computation and Control" and by the Swiss National Science Foundation.

F. D. Torrisi is with the Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), CH-8092 Zürich, Switzerland (e-mail: torrisi@control.ee.ethz.ch).

A. Bemporad is with the Dipartimento Ingegneria dell'Informazione, University of Siena, Siena 53100, Italy (e-mail: bemporad@dii.unisi.it).

Digital Object Identifier 10.1109/TCST.2004.824309

is a finite state machine equipped with continuous dynamics. In the theory of *timed automata*, the dynamic part is the continuous-time flow $\dot{x} = 1$. Efficient computational tools complete the theory of timed automata and allow one to perform verification and scheduling of such models. Timed automata were extended to *linear hybrid automata* [5], where the dynamics is modeled by the differential inclusion $a \leq \dot{x} \leq b$. Specific tools allow one 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 or by approximating the model with timed automata or linear hybrid automata [6].

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 (=continuous 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* [9], *piecewise affine* (PWA) *systems* [10], *linear complementarity* (LC) *systems*, *extended linear complementarity* (ELC) *systems*, and *max–min–plus-scaling* (MMPS) *systems* [11]. In particular, as shown in [11] all those modeling frameworks are equivalent (possibly under some hypothesis) and it is possible to 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 behavior do not appear in discrete-time. Although it is possible to consider hybrid automata in continuous-time, several computational tools profit from the discretization of time.[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 one to recast reachability/observability analysis, optimal control, and receding horizon estimation as mixed-integer linear/quadratic optimization problems. Reachability analysis algorithms were developed in [12] and [13] for stability and performance analysis of hybrid PWA systems. In [12], the authors also presented a novel approach for solving scheduling problems using combined reachability analysis and quadratic optimization for MLD and PWA models. For feedback control, in [9], 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 state estimation admits a receding horizon solution scheme [14]–[16].

Finally, we mention that identification techniques for piecewise affine systems were recently developed [17]–[20], that allow one to derive models (or parts of models) from input–output data.

We remark that, among all the equivalent discrete-time modeling framework mentioned above, DHA are the closest to modeling practice in the sense that they provide both convenient submodels (like finite state machines and connections) and a convenient textual representation as input description for the tool HYSDEL. On the other hand, MLD and PWA models embed and conceal the sub-parts in a computational convenient collection of equalities and inequalities that often are hard to determine by hand.

In this paper, 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 [21], [22], [9], 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), which allows describing the hybrid dynamics in a textual form, and a related compiler which provides different model representations of the given hybrid dynamics. We will detail a complete automotive case study. We will first derive a model of the car engine and power train. Then, we will analyze a controller defined using heuristics, and finally we will synthesize a PWA optimal controller. In order to perform such tasks, we will use the compiler HYSDEL, the piecewise linear toolbox [19], by applying reachability analysis [13], and by using explicit model predictive control techniques [23].

The HYSDEL compiler is available at http://control.ee.ethz.ch/~hybrid/hysdel. Additional related software in Matlab is available at http://www.dii.unisi.it/hybrid/tools.html.

II. DHA

DHA are the interconnection of a finite state machine and a switched linear dynamic system through a mode selector and an event generator (see Fig. 1).

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

A. SAS

An SAS is a collection of linear affine systems

$$x'_r(k) = A_{i(k)}x_r(k) + B_{i(k)}u_r(k) + f_{i(k)} \quad (1a)$$

$$y_r(k) = C_{i(k)}x_r(k) + D_{i(k)}u_r(k) + g_{i(k)} \quad (1b)$$

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 con-

[1]Also some tools for continuous-time hybrid models perform internally a time discretization of the model in order to execute the computations [6].

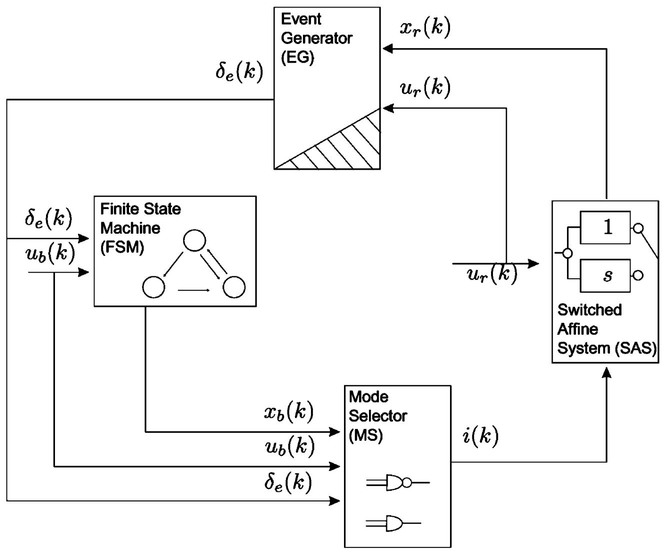


Fig. 1. DHA is the connection of an FSM and an SAS, through an MS and an EG. The output signals are omitted for clarity.

tinuous 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 opportune dimensions, and the mode $i(k) \in \mathcal{I} \triangleq \{1, \ldots, s\}$ is an input signal that chooses the affine state update dynamics. A SAS of the form (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 Section II.E. A SAS can be rewritten as the combination of linear terms and IF–THEN–ELSE rules: The state-update equation (1a) is equivalent to

$$z_1(k) = \begin{cases} A_1 x_r(k) + B_1 u_r(k) + f_1, & \text{if } (i(k) = 1) \\ 0, & \text{otherwise} \end{cases} \quad (2a)$$

$$\vdots$$

$$z_s(k) = \begin{cases} A_s x_r(k) + B_s u_r(k) + f_s, & \text{if } (i(k) = s) \\ 0, & \text{otherwise} \end{cases} \quad (2b)$$

$$x_r'(k) = \sum_{i=1}^{s} z_i(k) \quad (2c)$$

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

B. EG

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

$$\delta_e(k) = f_{\mathrm{H}}(x_r(k), u_r(k), k) \quad (3)$$

where $f_{\mathrm{H}} : \mathcal{X}_r \times \mathcal{U}_r \times \mathbb{Z}_{\geq 0} \to \mathcal{D} \subseteq \{0,1\}^{n_e}$ is a vector of descriptive functions of a linear hyperplane, and $\mathbb{Z}_{\geq 0} \triangleq \{0, 1, \ldots\}$ 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 and t_0 is a given 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 ^{i} denotes the ith component of a vector.

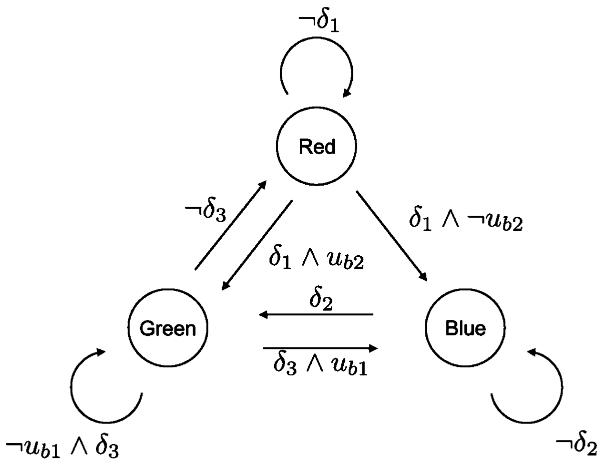


Fig. 2. Example of an FSM.

C. FSM

An FSM[2] (or automaton) is a discrete dynamic process that evolves according to a logic state update function

$$x_b'(k) = f_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k)) \quad (4a)$$

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. An FSM can be conveniently represented using an oriented graph. An FSM may also have an associated Boolean output

$$y_b(k) = g_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k)) \quad (4b)$$

where $y_b \in \mathcal{Y}_b \subseteq \{0,1\}^{p_b}$ and $g_b : \mathcal{X}_r \times \mathcal{U}_r \times D \mapsto \mathcal{Y}_b$. The idea of transforming a well-posed FSM into a set of Boolean equalities was already presented in [24] where the authors performed model checking using (mixed) integer optimization on an equivalent set of integer inequalities.

Example 1: Fig. 2 shows an FSM where $u_b = [u_{b1}\, u_{b2}]^T$ is the input vector, and $\delta = [\delta_1 \ldots \delta_3]^T$ is a vector of signals coming from the event generator. The logic state update function or *state transition function* is

$$x_b'(k) = \begin{cases} \text{Red if } ((x_b(k) = \text{Green}) \wedge \neg\delta_3) \vee \\ \quad ((x_b(k) = \text{Red}) \wedge \neg\delta_1) \\ \text{Green if} ((x_b(k) = \text{Red}) \wedge \delta_1 \wedge u_{b2}) \vee \\ \quad ((x_b(k) = \text{Blue}) \wedge \delta_2) \vee \\ \quad ((x_b(k) = \text{Green}) \wedge \neg u_{b1} \wedge \delta_3), \\ \text{Blue if } ((x_b(k) = \text{Red}) \wedge \delta_1 \wedge \neg u_{b2}) \vee \\ \quad ((x_b(k) = \text{Green}) \wedge (\delta_3 \wedge u_{b1})) \vee \\ \quad ((x_b(k) = \text{Blue}) \wedge \neg\delta_2)). \end{cases} \quad (5)$$

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

By associating a Boolean vector $x_b = [\begin{smallmatrix} x_{b1} \\ x_{b2} \end{smallmatrix}]$ to each state (Red $= [\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}]$, Green $= [\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}]$, and Blue $= [\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}]$), one can rewrite (5) as

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

where the time index (k) was omitted for brevity.

Note that since the logic state update function is deterministic, for each state the conditions associated to all the outgoing arcs are mutually exclusive.

D. 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 $f_{\mathrm{M}} : \mathcal{X}_b \times \mathcal{U}_b \times \mathcal{D} \mapsto \alpha(\mathcal{I})$, which is therefore called MS, where α is the set of the binary codings of the elements of $(\mathcal{I})$. The output of this function

$$i(k) = f_{\mathrm{M}}(x_b(k), u_b(k), \delta_e(k)) \tag{6}$$

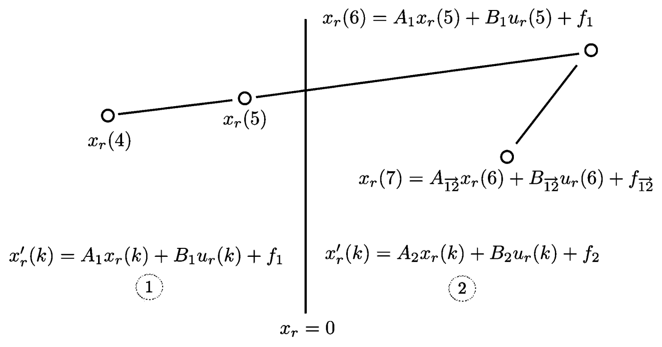
is called *active mode*. We say that a *mode switch* occurs at step k if $i(k) \neq i(k-1)$. Note that, in contrast to continuous-time hybrid models, where switches can occur at any time, in our discrete-time setting a mode switch can only occur at sampling instants.

E. 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

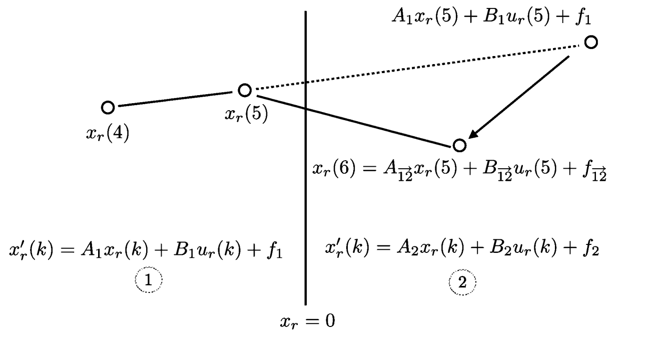
$$\begin{aligned} x'_r(k) &= \phi_{\overrightarrow{hj}}(x_r(k), u_r(k)) \\ &= A_{\overrightarrow{hj}} x_r(k) + B_{\overrightarrow{hj}} u_r(k) + f_{\overrightarrow{hj}} \end{aligned} \tag{7}$$

the function $\phi_{\overrightarrow{hj}} : \mathcal{X}_r \times \mathcal{U}_r \mapsto \mathcal{X}_r$ is called *reset map*. The reset can be considered as a special dynamics that only acts for one sampling step. Fig. 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, $x_r(8) = A_{\overrightarrow{23}} x_r(7) + B_{\overrightarrow{23}} u_r(7) + f_{\overrightarrow{23}}$.



$x_r(6) = A_1 x_r(5) + B_1 u_r(5) + f_1$
$x_r(4)$
$x_r(5)$
$x_r(7) = A_{\overrightarrow{12}} x_r(6) + B_{\overrightarrow{12}} u_r(6) + f_{\overrightarrow{12}}$
$x'_r(k) = A_1 x_r(k) + B_1 u_r(k) + f_1$
$x'_r(k) = A_2 x_r(k) + B_2 u_r(k) + f_2$
1
2
$x_r = 0$

(a) A posteriori resets



$A_1 x_r(5) + B_1 u_r(5) + f_1$
$x_r(4)$
$x_r(5)$
$x_r(6) = A_{\overrightarrow{12}} x_r(5) + B_{\overrightarrow{12}} u_r(5) + f_{\overrightarrow{12}}$
$x'_r(k) = A_1 x_r(k) + B_1 u_r(k) + f_1$
$x'_r(k) = A_2 x_r(k) + B_2 u_r(k) + f_2$
1
2
$x_r = 0$

(b) Predictive resets

Fig. 3. Reset maps.

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

Proof: Let the superscript $^{\mathrm{a}}$ 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

$$\begin{aligned} x'_b(k) &= f_{\mathrm{B}}(x_b(k), u_b(k), \delta_e(k)) \\ &= \begin{bmatrix} f_{\mathrm{B}}^{\mathrm{a}}\left(x_b^{\mathrm{a}}(k), u_b(k), \delta_e(k)\right) \\ x_b^{\mathrm{a}}(k) \\ \delta_e(k) \\ u_b(k) \end{bmatrix}. \end{aligned}$$

The SAS dynamics of Σ is defined as in (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 $r \leq (s^{\mathrm{a}})^2$ is the number of reset maps (we assume that when the mode switch $h \to j$ of Σ does not have an associated reset map $\phi^{\mathrm{a}}_{\overrightarrow{hj}}$, then $\phi^{\mathrm{a}}_{\overrightarrow{hj}}$ defaults to the jth 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 jth dynamics (if $j = h$, or $j \neq h$ and $\phi^{\mathrm{a}}_{\overrightarrow{hj}}$ is not specified) or the reset dynamics $\phi^{\mathrm{a}}_{\overrightarrow{hj}}$. Since $i^{\mathrm{a}}(k) = f^{\mathrm{a}}_{\mathrm{M}}(x_b^{\mathrm{a}}(k), u_b^{\mathrm{a}}(k), \delta_e^{\mathrm{a}}(k))$, $i^{\mathrm{a}}(k-1) = f^{\mathrm{a}}_{\mathrm{M}}(x^{\mathrm{a}}(k-1), u_b^{\mathrm{a}}(k-1), \delta_e^{\mathrm{a}}(k-1))$, 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, $i'(k) = f_M(x'_b(k), f_H(x'_r(k), k))$ 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 *predictive resets*, in order to distinguish them from the resets described before, that we will call *a posteriori resets*.

Consider Fig. 3(b). At time $k = 5$ the state $x_r(5)$ and the input $u_r(5)$ are such that $A_1 x_r(5) + B_1 u_r(5) + f_1 \geq 0$ which would generate an event $\delta_e(6) = [x_r(6) \geq 0]$ at the next time step. As a consequence of the predicted mode switch, the state is reset according to the reset map $\phi_{\overrightarrow{12}}(x, u)$, i.e., $x(6) = A_{\overrightarrow{12}} x(5) + B_{\overrightarrow{12}} u(5) + f_{\overrightarrow{12}}$.

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 predictive resets can be rewritten as a DHA Σ without resets.

Proof: Let again the superscript $^{\mathrm{a}}$ denote the variables of Σ^{a}. By hypothesis, predictive 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 $x_r^{\mathrm{a}}(k), u_r^{\mathrm{a}}(k), i^{\mathrm{a}}(k)$. Define the EG for system Σ as $\delta_e(k) = [\delta_e^{\mathrm{a}}(k); \delta_e^{\mathrm{a}}(k+1)]$, and let $x_b(k) = x_b^{\mathrm{a}}(k)$, $u_b(k) = u_b^{\mathrm{a}}(k), x_r(k) = x_r^{\mathrm{a}}(k), u_r(k) = u_r^{\mathrm{a}}(k), y_r(k) = y_r^{\mathrm{a}}(k), y_b(k) = y_b^{\mathrm{a}}(k)$. Let the FSM for system Σ be equal to the FSM of Σ^{a}, and define the SAS dynamics of Σ as in the proof of Proposition 1. The MS of Σ should internally compute $j = i^{\mathrm{a}}(k+1), h = i^{\mathrm{a}}(k)$, compare them, and then choose either the jth dynamics (if $j = h$, or $j \neq h$ and $\phi^{\mathrm{a}}_{\overrightarrow{hj}}$ is not specified) or the reset dynamics $\phi^{\mathrm{a}}_{\overrightarrow{hj}}$. Since $i^{\mathrm{a}}(k) = f_M^{\mathrm{a}}(x_b^{\mathrm{a}}(k), \delta_e^{\mathrm{a}}(k)), i^{\mathrm{a}}(k+1) = f_M^{\mathrm{a}}(f_B^{\mathrm{a}}(x_b^{\mathrm{a}}(k), u_b^{\mathrm{a}}(k), \delta_e^{\mathrm{a}}(k)), \delta_e^{\mathrm{a}}(k+1))$, it follows that the mode $i(k)$ of the SAS dynamics of system Σ is a function of $x_b(k), u_b(k), \delta_e(k)$. ■

Note that in our discrete-time setting resets only occur at sampling instants, and therefore it cannot account for model artifacts typical of continuous-time models like *live-locks*, where infinite switches/resets occur at the same time instant. Clearly this limitation does not prevent to capture with enough detail most of systems of interest in the applications.

Example 2: Let us consider a DHA with two modes

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

In order to add the predictive reset map $\phi_{\overrightarrow{12}}(x_r, u_r) = 2$ to the model, we first consider the set $\mathcal{P} = \{x_r, u_r : a_1 x_r + b_1 u_r + f_1 \geq 0\}$ of all the state/input pairs that will trigger the event δ_e in one step and add an event $\delta_f = a_1 x_r + b_1 u_r + f_1 \geq 0$ 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:

$$\mathrm{SAS}: x'_r(k) = \begin{cases} x_r(k) + u_r(k) - 1, & \text{if } i(k) = 1 \\ 2x_r(k), & \text{if } i(k) = 2 \\ 2, & \text{if } i(k) = 3 \end{cases} \quad (8a)$$
$$\mathrm{EG}: \begin{cases} \delta_e(k) = [x_r(k) \geq 0], \\ \delta_f(k) = [x_r(k) + u_r(k) - 1 \geq 0], \end{cases} \quad (8b)$$
$$\mathrm{MS}: i(k) = \begin{cases} 1, & \text{if } \begin{bmatrix} \delta_e(k) \\ \delta_f(k) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ 2, & \text{if } \delta_e(k) = 1 \\ 3, & \text{if } \begin{bmatrix} \delta_e(k) \\ \delta_f(k) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \end{cases} \quad (8c)$$

As will be described in Section III, (8) admits the piecewise affine representation depicted in Fig. 4, that clearly shows that the reset condition is another dynamical mode.

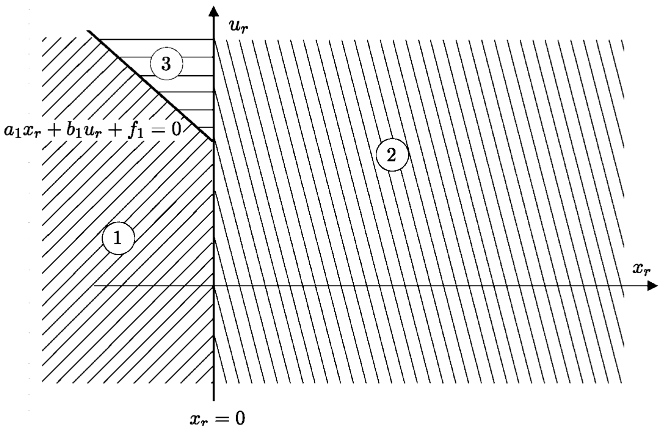


Fig. 4. Equivalent piecewise affine representation of (8), showing where each mode is active.

F. 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 $x(k), k \in \mathbb{Z}_{\geq 0}$ 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) $\delta_e(k) = f_H(x_r(k), u_r(k), k)$;
   b) $i(k) = f_M(x_b(k), u_b(k), \delta_e(k))$;
   c) $y_r(k) = C_{i(k)} x_r(k) + D_{i(k)} u_r(k) + g_{i(k)}$;
   d) $y_b(k) = g_B(x_b(k), u_b(k), \delta_e(k))$;
   e) $x'_r(k) = A_{i(k)} x_r(k) + B_{i(k)} u_r(k) + f_{i(k)}$;
   f) $x'_b(k) = f_B(x_b(k), u_b(k), \delta_e(k))$.

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 nondeterminism (the successor $x'_r(k), x'_b(k)$ may be multiply defined). We remark that trajectories of DHA are deterministic as a consequence of their definition.

DHA modes are related to *hybrid automata* (HA) [5], the main difference is in the time model, DHA admit time in the natural numbers, while in HA the time is a real number. Moreover, DHA models do not allow instantaneous transitions, and are deterministic, opposed to HA where any enabled transition may occur in zero time. This has two consequences: 1) DHA do not admit live-locks (infinite switches in zero time), and 2) DHA do not admit Zeno behaviors (infinite switches in finite time). Finally, in DHA models, guards, reset maps, and continuous dynamics are limited to linear affine functions. Moreover, contrarily to HA, in DHA the continuous dynamics is not a property of the state of the automaton but is selected by the MS according also to discrete inputs and events. However, working with discrete-time models allows the development of several analysis and synthesis tools, as later reported in Section VII.

III. DHA and PWA Systems

This section highlights the relationships between the DHA introduced above and the class of PWA systems [10]. PWA systems [11] are defined by partitioning the state space into polyhedral regions, and associating with each region a different affine state-update equation

$$x'(k) = A_{i(k)}x(k) + B_{i(k)}u(k) + f_{i(k)} \tag{9a}$$
$$y(k) = C_{i(k)}x(k) + D_{i(k)}u(k) + g_{i(k)} \tag{9b}$$
$$i(k) \text{ such that } H_{i(k)}x(k) + J_{i(k)}u(k) \le K_{i(k)} \tag{9c}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n, u \in \mathcal{U} \subseteq \mathbb{R}^m, y \in \mathcal{Y} \subseteq \mathbb{R}^p, \{H_i x + J_i u \le K_i\}_{i=1}^s$, is a polyhedral partition[3] 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 (9c) should be interpreted componentwise. For PWA systems, well-posedness is defined similarly to Definition 1. An exact definition is available in [11].

Definition 2: Let Σ_1, Σ_2 be hybrid models, whose inputs are $u_1(k) \in \mathcal{U}_1 \subseteq \mathcal{U}, u_2(k) \in \mathcal{U}_2 \subseteq \mathcal{U}$ and outputs $y_1(k) \in \mathcal{Y}_1 \subseteq \mathcal{Y}, y_2(k) \in \mathcal{Y}_2 \subseteq \mathcal{Y}, k \in \mathbb{Z}_{\ge 0}$. Let $x_1(k) \in \mathcal{X}_1 \subseteq \mathcal{X}$ be the state of Σ_1 and $x_2(k) \in \mathcal{X}_2 \subseteq \mathcal{X}$ the state of $\Sigma_2, k \in \mathbb{Z}_{\ge 0}$. The hybrid models Σ_1 and Σ_2 are *equivalent* on $\bar{\mathcal{X}}, \bar{\mathcal{U}}, \bar{\mathcal{Y}}, \bar{\mathcal{X}} \subseteq \mathcal{X}_1 \cap \mathcal{X}_2, \bar{\mathcal{U}} \subseteq \mathcal{U}_1 \cap \mathcal{U}_2, \bar{\mathcal{Y}} \subseteq \mathcal{Y}_1 \cap \mathcal{Y}_2$ if for all initial conditions $x_1(0) = x_2(0) \in \bar{\mathcal{X}}$, and for all $u_1(k) = u_2(k) \in \bar{\mathcal{U}}$, the output trajectories coincides, i.e., $y_1(k) = y_2(k)$ and $x_2(k) = x_1(k)$ at all steps $k \in \mathbb{Z}_{\ge 0}$.

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, it can be rewritten as an equivalent well-posed DHA model Σ_{DHA} on $\mathcal{U}, \mathcal{X}, \mathcal{Y}$.

Proof: Equations (9a)–(9b) are the modes of the SAS, the constraints $H_i x + J_i u \le K_i, i = 1, \ldots, s$ are the defining hyperplanes $f_{\text{H}}(\cdot)$ of the EG, and the MS is defined by (9c), namely if all the events associated to the hyperplanes of $H_j x + J_j u \le K_j$ 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.

IV. DHA and MLD 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 [9], [21], [22], [25], [26], and the equivalence between DHA and mixed logical dynamical (MLD) systems [9].

A. Logical Functions

Boolean functions can be equivalently expressed by inequalities [27].

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[4] by the grammar

$$\phi ::= X \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \oplus \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \leftarrow \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2 \mid (\phi_1) \tag{10}$$

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 in *conjunctive normal form* (CNF) or *product of sums* if it can be written according to the following grammar:

$$\phi ::= \psi \mid \phi \wedge \psi \tag{11}$$
$$\psi ::= \psi_1 \vee \psi_2 \mid \neg X \mid X \tag{12}$$

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 $X_1, \ldots, X_{n-1}$ as follows:

$$X_n = f(X_1, X_2, \ldots, X_{n-1}). \tag{13}$$

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

$$F(X_1, \ldots, X_n) = 1 \tag{14}$$

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 $X_1 + X_2 \ge 1$ [21]. The translation can be performed either using an *symbolical* method or a *geometrical* method.

[3]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 (9) to be strict. However, from a numerical point of view, this issue is not relevant.

[4]For the sake of simplicity, we are neglecting precedence.

1) Symbolical Method: The symbolical method consists of first converting (13) or (14) into CNF, a task that can be performed automatically by using one of the several standard techniques available. Let the CNF have the form $\bigwedge_{j=1}^{m}(\bigvee_{i\in P_j} X_i \bigvee_{i\in N_j} \neg X_i), N_j, P_j \subseteq \{1,\ldots,n\} \forall j = 1,\ldots,m$. Then, the corresponding set of integer linear inequalities is

$$\begin{cases} 1 \le \sum_{i\in P_1} X_i + \sum_{i\in N_1}(1-X_i) \\ \vdots \\ 1 \le \sum_{i\in P_m} X_i + \sum_{i\in N_m}(1-X_i) \end{cases}. \tag{15}$$

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

$$P_{\text{CNF}} = \{x \in [0,1]^n\text{: (15) are satisfied with } x = [X_1,\ldots,X_n]^T\}. \tag{16}$$

2) Geometrical Method: The geometrical method consists of two steps (see, e.g., [28]). First, the set of points satisfying (13) or (14) is computed (for this reason, the method was also called *truth table* method in [28]). 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]). Therefore, we define

$$P_{\text{CH}} = \{x \in [0,1]^n\text{: } x \in \text{conv}(V)\}. \tag{17}$$

Although P_{CH} and P_{CNF} contain the same integer points, i.e., $(P_{\text{CH}} \cap \{0,1\}^n) = (P_{\text{CNF}} \cap \{0,1\}^n)$, in general the set $P_{\text{CH}} \subseteq P_{\text{CNF}}$, since $\text{conv}(V)$ is the smallest set containing all integer feasible points. However, there exist Boolean formulas, for which $P_{\text{CH}} \neq P_{\text{CNF}}$[5]. Conditions for which $P_{\text{CH}} = P_{\text{CNF}}$ are currently a topic of research.

B. Continuous-Logic Interfaces

Events of the form (3) can be equivalently expressed as

$$f_{\text{H}}^i(x_r(k), u_r(k), k) \le M^i(1-\delta_e^i) \tag{18a}$$
$$f_{\text{H}}^i(x_r(k), u_r(k), k) > m^i\delta_e^i, \qquad i = 1,\ldots,n_e \tag{18b}$$

where M^i, m^i are upper and lower bounds, respectively, on $f_{\text{H}}^i(x_r(k), u_r(k), k)$. As we will point out in Section IV-D, 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 [21] to replace the strict inequality (18b) as

$$f_{\text{H}}^i(x_r(k), u_r(k), k) \ge \epsilon + (m^i - \epsilon)\delta_e^i \tag{18c}$$

where ϵ is a small positive scalar, e.g., the machine precision, although the equivalence does not hold for $0 < f_{\text{H}}^i(x_r(k), u_r(k), k) < \epsilon$, the numbers in the interval $(0,\epsilon)$ cannot be represented in a computer.

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

The most common *logic to continuous* interface is the IF–THEN–ELSE construct

$$\begin{aligned} \text{IF } \delta \text{ THEN } z &= a_1^T x + b_1^T u + f_1 \\ \text{ELSE } z &= a_2^T x + b_2^T u + f_2 \end{aligned} \tag{19}$$

which can be translated into [13]

$$(m_2 - M_1)\delta + z \le a_2 x + b_2 u + f_2 \tag{20a}$$
$$(m_1 - M_2)\delta - z \le -a_2 x - b_2 u - f_2 \tag{20b}$$
$$(m_1 - M_2)(1-\delta) + z \le a_1 x + b_1 u + f_1 \tag{20c}$$
$$(m_2 - M_1)(1-\delta) - z \le -a_1 x - b_1 u - f_1 \tag{20d}$$

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 when a_2, b_2, f_2 are zero, (19)–(20) coincide with the product $z = \delta \cdot (ax + bu + f)$ described in [21].

C. Continuous Dynamics

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

$$x'(k) = \sum_{i=1}^{s} z_i(k). \tag{21}$$

D. MLD Systems

In [9], the authors proposed discrete-time hybrid systems denoted as MLD systems. An MLD system is described by the following relations:

$$\begin{aligned} x'(k) &= Ax(k) + B_1 u(k) + B_2\delta(k) \\ &\quad + B_3 z(k) + B_5 \end{aligned} \tag{22a}$$
$$\begin{aligned} y(k) &= Cx(k) + D_1 u(k) + D_2\delta(k) \\ &\quad + D_3 z(k) + D_5 \end{aligned} \tag{22b}$$
$$E_2\delta(k) + E_3 z(k) \le E_1 u(k) + E_4 x(k) + E_5 \tag{22c}$$
$$\bar{E}_2\delta(k) + \bar{E}_3 z(k) = \bar{E}_1 u(k) + \bar{E}_4 x(k) + \bar{E}_5 \tag{22d}$$

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 $A, B_1, B_2, B_3, C, D_1, D_2, D_3, E_1, \ldots, E_5$, and $\bar{E}_1, \ldots, \bar{E}_5$ are matrices of suitable dimensions. Given the current state $x(k)$ and input $u(k)$, the time-evolution of (22) is determined by solving $\delta(k)$ and $z(k)$ from (22c)–(22d), and then updating $x'(k)$ and $y(k)$ from (22a)–(22b). The equations and inequalities obtained with methods presented in Sections IV-A–C can be represented using the MLD framework. Since the problems of synthesis and analysis of MLD models are tackled by optimization techniques, we have replaced strict inequalities as in (18b) by nonstrict inequalities as in (18c).[6] For MLD systems, well-posedness is defined similarly to Definition 1. A formal definition of well-posedness for MLD systems and a test to assess the well-posedness have been presented in [9]. Finally, we recall that the MLD model is similar to the model presented in [30] for verification of safety properties as they

[6]One may also explicitly include in (22) strict inequality constraints $\tilde{E}_2\delta(k) + \tilde{E}_3 z(k) < \tilde{E}_1 u(k) + \tilde{E}_4 x(k) + \tilde{E}_5$.

both aim at translating a hybrid system in a set of mixed integer linear equalities and inequalities using similar techniques.

Lemma 2: Let $\Sigma_{\rm DHA}$ 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, for any bounded $\bar{\mathcal{X}}, \bar{\mathcal{U}}$, there exists a well posed MLD model $\Sigma_{\rm MLD}$ equivalent to $\Sigma_{\rm DHA}$ on $\bar{\mathcal{X}}, \bar{\mathcal{U}}, \mathcal{Y}$.

Proof: Directly follows from Sections IV-A–C.

V. Other Computational Models and Further Equivalences

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.

A. Other Discrete-Time Modeling Frameworks

In [11], the relationships among the model classes mentioned previously and three others: LC, MMPS, and ELC systems, were discussed. ELC systems and LC systems are linear systems where two vectors are linked by an orthogonality constraint, see [11] for details. 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 details on this class can be also found in [11].

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 [11] for full details 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: Mutual equivalences among PWA, MLD, LC, ELC, and MMPS on bounded $\mathcal{X}, \mathcal{U}, \mathcal{Y}$ follows from Fact 1. By Lemma 1, any PWA model $\Sigma_{\rm PWA}$ can be rewritten as an equivalent DHA model $\Sigma_{\rm DHA}$, while any $\Sigma_{\rm DHA}$ can be rewritten as an MLD model $\Sigma_{\rm MLD}$ by Lemma 2. Therefore, any equivalence relation can be stated for any ordered pairs of models. ■

Note that by Propositions 1 and 2 also DHA models with resets are equivalent to any of the other classes of hybrid models.

We remark once more that all the models are equivalent. While there is no difference in modeling capability among the models, the same task can be solved substantially more efficiently by picking the proper model. Table I summarizes the advised model for several typical engineering tasks, according to the authors' knowledge of the state of the art.

TABLE I
Advised Model for Each Task

Task	Model
Modeling	DHA
Simulation	DHA
Control	MLD,PWA,MMPS
Stability	PWA
Verification	PWA
Identification	PWA
Fault Detection	MLD
Estimation	MLD

VI. HYSDEL Description of DHA Models

We designed a modeling language to describes DHA models, called HYSDEL. The HYSDEL description of a DHA is an abstract modeling step. The associated HYSDEL compiler then translates the description into several computational models, in particular into a MLD using the technique presented in Section IV, and PWA form using either the direct approach of [31] or the indirect approach of [32] that translates the MLD into a PWA. It is also possible to generate LC/ELC/MMPS systems using the constructive methods reported in [11]. HYSDEL can generate also a simulator that runs as a function in Matlab.

TABLE II
Sample HYSDEL List of System (8).

```
SYSTEM sample {
INTERFACE {
  STATE {
    REAL xr [-10, 10]; }
  INPUT {
    REAL ur [-2, 2]; }
}
IMPLEMENTATION {
  AUX {
    REAL z1, z2, z3;
    BOOL de, df, d1, d2, d3; }
  AD {
    de = xr >= 0;
    df = xr + ur - 1 >= 0; }
  LOGIC {
    d1 = ~de & ~df;
    d2 = de;
    d3 = ~de & df; }
  DA {
    z1 = {IF d1 THEN xr + ur - 1 };
    z2 = {IF d2 THEN 2 * xr };
    z3 = {IF (~de & df)  THEN 2 }; }
  CONTINUOUS {
    xr = z1 + z2 + z3; }
}}
```

In this section, we show how a DHA system can be modeled in HYSDEL by analyzing the HYSDEL description of the DHA (8). A complete description of the syntax of HYSDEL is available in the manual accompanying the compiler [33], and an example of realistic size is presented in Section VIII.

Consider the HYSDEL list of Table II. As any HYSDEL list, it is composed of two parts. The first one, called INTERFACE, contains the declaration of all variables and parameters, so that it is possible to make the proper type checks. The second part, IMPLEMENTATION, is composed of specialized sections where the relations among the variables are described. These sections are described next.

AUX Section The HYSDEL section AUX contains the declaration of the auxiliary variables used in the model. These variables will become the δ and z variables in the MLD model (22).

AD Section The HYSDEL section AD allows one to define Boolean variables from continuous ones, and is based exactly on the same semantics of the EG described earlier.

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.

LOGIC SECTION The section LOGIC allows one 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.

DA SECTION The HYSDEL section DA defines continuous variables according to IF–THEN–ELSE conditions on Boolean variables. This section models part of the SAS, namely the variables z_i defined in (2a)–(2b). Note that, as the definition of `z3` suggests, HYSDEL can handle compound logic formulas in the DA section, therefore, there is no need to explicitly define a Boolean variable for each mode.

CONTINUOUS SECTION The CONTINUOUS section describes the linear dynamics, expressed as difference equations. This section models (2c).

An HYSDEL description may have additional sections that are not part of the sample code of Table II, that we describe below. For examples and the detailed syntax, we refer the interested reader to [33].

LINEAR SECTION HYSDEL allows also one to define a continuous variable as an affine function of continuous variables in the LINEAR section. 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 the 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 FSM as a collection of Boolean functions $x'_{bi}(k) = f_{\mathrm{B}i}(x_b(k), u_b(k), \delta_e(k)), i = 1, \ldots, n_b$.

OUTPUT SECTION The OUTPUT section allows one to specify static linear and logic relations for the output vector $y = \begin{bmatrix} y_r \\ y_b \end{bmatrix}$.

Finally, HYSDEL allows one more section.

MUST SECTION This section specifies arbitrary linear and logic constraints on continuous and Boolean variables, and therefore it allows for 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).

VII. APPLICATIONS

The hybrid models automatically generated by HYSDEL can be used for solving several analysis (stability, observability, safety/reachability) and design (control, state estimation) tasks. In particular, in this section we focus our attention by recalling tools for reachability analysis and controller synthesis.

A. Reachability Analysis

The *reachability analysis* of hybrid dynamical systems is a tool for the verification of *safety properties*: For a given set of initial conditions and exogenous signals, verify that the set of unsafe states cannot be entered, or provide a counterexample. More precisely, we define the following.

REACHABILITY ANALYSIS PROBLEM Given a PWA system Σ, a polyhedral set of initial conditions $\mathcal{X}(0)$, a collection of disjoint target polyhedral sets $\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_L$, a bounded set of inputs $\mathcal{U}$, and a time horizon $k \le K_{\max}$, determine i) if $\mathcal{Z}_j$ is reachable from $\mathcal{X}(0)$ within $k \le K_{\max}$ steps for some sequence $\{u(0), \ldots, u(k-1)\} \subseteq \mathcal{U}$ of exogenous inputs; ii) if yes, the subset of initial conditions $\mathcal{X}_{\mathcal{Z}_j}(0)$ of $\mathcal{X}(0)$ from which $\mathcal{Z}_j$ can be reached within $K_{\max}$ steps; iii) for any $x_1 \in \mathcal{X}_{\mathcal{Z}_j}(0)$ and $x_2 \in \mathcal{Z}_j$, the input sequence $\{u(0), \ldots, u(k-1)\} \subseteq \mathcal{U}, k \le K_{\max}$, which drives x_1 to x_2. The justification for focusing on finite-time reachability is that states which are not reachable in less than $K_{\max}$ steps are considered, in practice, unreachable. Although finite-time reachability analysis can only guarantee finite-time liveness properties (for instance, it cannot check if $\mathcal{Z}_i$ will be ever reached), the reachability problem stated above is clearly decidable. Nevertheless, the problem is $\mathcal{NP}$-hard. An algorithm for solving this problem was presented in [13].

B. Receding Horizon Control of Hybrid Systems

Receding horizon hybrid optimal control [9], [23], also known as model predictive control, can be usefully employed to synthesize control laws for hybrid systems. The main idea is to setup a finite-horizon optimal control problem for the hybrid MLD system (22) by optimizing a performance index under operating constraints:

$$\min_{\xi} J(\xi, x(t)) \triangleq \sum_{k=0}^{N-1} \|Q(y_k - r(k))\|_p + \|Ru_k\|_p \quad (23a)$$

$$\text{subj. to} \begin{cases} x_0 = x(k) \\ x_{k+1} = Ax_k + B_1u_k + B_2\delta_k + B_3z_k \\ y_k = Cx_k + D_1u_k + D_2\delta_k + D_3z_k \\ E_2\delta_k + E_3z_k \le E_1u_k + E_4x_k + E_5 \end{cases} \quad (23b)$$

where $x(k)$ is the state of the MLD system at step $k, r(k)$ is the current desired reference, $\xi \triangleq [u_0; \delta_0; z_0; \ldots; u_{N-1}; \delta_{N-1}; z_{N-1}]$ is the optimization vector, and Q and R are weighting matrices. The subscript p denotes the standard one-norm for $p = 1, \infty$-norm for $p = \infty$, and the squared Euclidean norm for $p = 2$.

In (23), we assume that possible physical and/or logical constraints on the variables of the hybrid system are already included in the mixed-integer linear constraints of the MLD model, as they can be conveniently modeled in the MUST section of the HYSDEL model. It is in fact extremely useful to specify constraints over state and output variables that must be fulfilled by the trajectories of the closed-loop system directly in terms of the HYSDEL variables.

Receding horizon control (RHC) amounts to repeatedly computing the optimal solution to (23) at each time t, and applying only the first optimal control move u_0^* as the input $u(t)$ to the system. In [9] the authors have shown that the resulting closed-loop system (hybrid model + RHC controller) is asymptotically stable (provided that an end-point constraint on the final state x_N and weights on states and auxiliary variables are present).

Problem (23) can be translated into a mixed integer quadratic program (MIQP) ($p = 2$), or a mixed integer linear program (MILP) ($p = 1$ or $p = \infty$), i.e., into the minimization of a quadratic/linear cost function subject to linear constraints, where some of the variables are constrained to be binary; see [9] and [23] for details.

1) Implementation as a Piecewise Affine Control Law: In some cases online mixed-integer optimization may be prohibitive, for example in fast sampling applications. In such cases, the design of the controller is performed in two steps. First, the RHC controller based on the optimal control problem (23) is tuned in simulation using MILP solvers, until the desired performance is achieved. Then, for implementation, in the second phase the explicit piecewise affine form of the RHC law is computed offline by using a multiparametric mixed integer programming solver as shown in [23]

$$u(k) = \Phi_i \begin{bmatrix} x(k) \\ r(k) \end{bmatrix} + \gamma_i$$
$$\text{if } \Theta_i \begin{bmatrix} x(k) \\ r(k) \end{bmatrix} \le \kappa_i, \qquad i = 1, \dots, N_r \tag{24}$$

where $x(k), r(k)$ are the current state and desired reference, respectively, and $\Phi_i, \gamma_i, \Theta_i, \kappa_i$ are matrices of suitable dimension. We remark that the resulting piecewise affine control action (24) is *identical* to the RHC designed in the first phase.

VIII. Analysis and Control of a Cruise Control System

In this section we use HYSDEL to obtain a hybrid model of a car with robotized manual gear shift, and show how such a model can be directly used i) to formally verify certain safety and liveness properties of a simple cruise controller based on PI control and a set of gear-switching rules and ii) to synthesize a cruise control system that is piecewise affine and optimal with respect to a certain performance index.

A. Car Model

We focus on a car equipped with manual transmission, and we assume that the gear command is robotized, namely that a slave control system takes care of releasing the clutch, shifting the gear, and engaging the clutch. We only consider the longitudinal dynamics of the car: the continuous variables are the scalar position $x(\text{m})$ and the speed $v = \dot{x}$ (m·s^{-1}). The continuous inputs are the engine torque u_t (Nm), the braking force u_b (N), and the sinus of the road slope u_s, plus six binary inputs $g_1, g_2, g_3, g_4,$ and g_5, and $g_R \in \{0, 1\}$ corresponding to the selected gear. Denoting by ω the engine speed (rad·s^{-1}), we have the kinematic relation

$$v = \frac{k_{\text{loss}} r_{\text{wheel}}}{R_g(i) R_{\text{fin}}} \omega \tag{25}$$

where $R_g(i)$ is the gear ratio corresponding to the ith gear, R_{fin} is the final drive ratio, r_{wheel} is the wheel radius, and k_{loss} is the drive train efficiency level [34]. Note that by using a kinematic relation for the speed engine, we are neglecting the clutch, the motor dynamic behavior, and we are assuming that the time spent for gear shift is negligible.

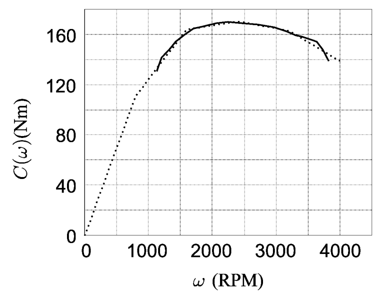


Fig. 5. Maximum torque of the engine of the Clio 1.9 DTI RXE (solid line) and PWL approximation (dotted line).

The dynamic equation of motion of the car is $m\ddot{x} = F_e - F_b - F_r - F_s$ where m (kg) is the vehicle mass, F_e (N) is the traction force, $F_b = u_b$ is the braking force (N), F_r (N) is the friction force, and F_s (N) takes into account the slope of the road. $F_s = mgu_s$ (N), where $u_s = \sin\alpha$ and α is the slope of the road and as a first approximation, we assume that F_r is linear in $v, F_r = \beta v$, where β (kg·m·s^{-1}) is a constant that takes into account all the frictions (i.e., aerodynamic, tires deformation, drive train). From the conservation of mechanical power, we have $F_e v = \omega u_1$, which gives $F_e = (k_{\text{loss}})/(R_g(i))u_1$. The commanded torque u_t is upper-bounded by the maximum torque deliverable at a certain engine speed $\omega, u_t \le C_e^+(\omega)$ where $C_e^+(\omega)$ is a nonlinear function typically reported in the data sheets of the car and $C_e^-(\omega)$ is the maximum braking force that the engine can provide when the throttle is fully released. In order to derive a hybrid model of the car as described in Section II, we piecewise-linearize $C_e^+(\omega)$ into four regions using the PWL toolbox [35], which requires the introduction of three event variables $\delta_{\text{PWL1}}, \delta_{\text{PWL2}},$ and δ_{PWL3}, and as a first approximation, we assume $C_e^-(\omega) = -\alpha_1 - \beta_1\omega$, cf. Fig. 5.

The engine speed ω can be written as a SAS, and by (2), as the sum of auxiliary continuous variables, $\omega = \omega_1 + \omega_2 + \omega_3 + \omega_4 + \omega_5 + \omega_R$, where

$$\omega_i = \begin{cases} \frac{k_s}{R_g(i)}\dot{x}, & \text{if } g_i = 1 \\ 0, & \text{otherwise.} \end{cases}$$

To validate the model, we took the parameters of the Renault Clio 1.9 DTI RXE from http://www.renault.com/. The simulated acceleration and max speed tests gave the same results as the experimental counterpart, reported in the technical documentation. For the reader's convenience we report the main parameters of the car under consideration: $R_g(1) = 3.7271$, $R_g(2) = 2.048, R_g(3) = 1.321, R_g(4) = 0.971, R_g(5) = 0.756$, $R_g(R) = -3.545, R_{\text{fin}} = 3.294, k_{\text{loss}} = 0.925$, $r_{\text{wheel}} = 0.2916$ m, $\beta = 25$ kg·m·s^{-1}, $m = 1020$ kg, $\alpha_1 = 10$ Nm, $\beta_1 = 0.3$ kg·m^2·s^{-1}. Fig. 5 reports the measured maximum torque and the piecewise affine approximation, the maximum error is 5.7 Nm. Finally, the dynamics is discretized with sampling time $T_s = 0.5$ s using forward finite differences to obtain the DHA model. The corresponding HYSDEL model of the car is reported in Appendix. The resulting MLD model contains two continuous states (vehicle position x and speed v), three continuous inputs (engine torque F_e, breaking force

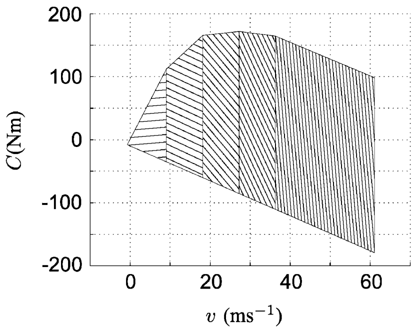


Fig. 6. PWA system equivalent to the MLD model obtained through HYSDEL from the list reported in Appendix in the (velocity,torque) space for position $x = 0$, braking force $F_b = 0, \text{gear inputvector} = [0\ 0\ 0\ 0\ 1\ 0]^T$ (4th gear).

F_b, and slope u_s), 6 binary inputs (gears $g_R, g_1, \ldots, g_5$), one continuous output (speed v), 16 auxiliary continuous variables (six for the traction force, six for the engine speed, and four for the piecewise linearization of the maximum engine torque), four auxiliary binary variables (breakpoints for the piecewise linearization of the maximum engine torque), and 96 mixed-integer inequalities. The DHA model can be then transformed into a PWA model using the approach presented in [31][7] or, equivalently, the MLD model can be converted in a PWA model by running the algorithm proposed in [32][8]. The total number of binary variables is $0 + 6 + 4 = 10$, which gives a worst-case number of possible regions in the PWA system equals to $2^{10} = 1024$, while the PWA equivalent to the hybrid MLD model has 30 regions, and is computed in 7.5 s starting from the DHA and in 72.66 s starting from the MLD using Matlab on a Pentium III 650-MHz machine. Fig. 6 shows a section of the resulting PWA model.

B. Reachability Analysis

We want to show how the HYSDEL model can be successfully employed to verify safety properties. To this end, we assume we have a simple cruise controller from a previous design. We want to verify that the controlled system will never exceed the target speed by some tolerance, for instance the speed limits imposed by local authorities. The complete hybrid system under examination is now composed of two subsystems: the car dynamic model described in Section VIII, and the cruise controller. For a detailed description of compositional DHA models, refer to [31].

1) Model of the Cruise Controller: The controller commands throttle position, braking force, and selected gear, based on the desired vehicle speed and measurements of the actual car speed.

The automaton reported in Fig. 7 selects the gear. If the engine speed is faster than $\omega_u = 5000$ RPM then the gear is shifted up. Similarly, if the speed is lower than $\omega_l = 3000$ RPM, the gear is shifted down. The two thresholds are chosen by looking at the max torque plot in Fig. 5. To track the desired speed reference $v_r(k)$, the throttle and the brakes are op-

[7]The corresponding software is available for download from http://control.ee.ethz.ch/~hybrid/hysdel as a plug-in for HYSDEL.

[8]The corresponding software is available for download from http://www.dii.unisi.it/~bemporad/tools

$$\begin{tikzcd} 1 \arrow[r, shift left, "w \geq w_u"] & 2 \arrow[l, shift left, "w < w_l"] \arrow[r, shift left, "w \geq w_u"] & 3 \arrow[l, shift left, "w < w_l"] \arrow[r, shift left, "w \geq w_u"] & 4 \arrow[l, shift left, "w < w_l"] \arrow[r, shift left, "w \geq w_u"] & 5 \arrow[l, shift left, "w < w_l"] \end{tikzcd}$$

Fig. 7. Gear shift logic controller.

erated by a PI controller. Let $e(k)$ be the integral error, $e'(k) = e(k) + T_s \Delta v(k), \Delta v(k) = v_r(k) - v(k)$. The controller is

$$u_t(k) = \begin{cases} k_t \Delta v(k) + i_t e(k), & \text{if } v(k) \leq v_r(k) + 1 \\ 0, & \text{otherwise} \end{cases} \tag{26a}$$

$$u_b(k) = \begin{cases} k_b \Delta v(k), & \text{if } v(k) > v_r(k) + 1 \\ 0, & \text{otherwise.} \end{cases} \tag{26b}$$

The control variables u_t and u_b are saturated against the maximum torque and braking force, respectively. The integrator in the PI controller uses an anti-windup scheme: $e(k)$ is integrated only when the control inputs u_t and u_b are not saturated. Note that, the threshold in (26) is 1 ms^{-1} over the target speed, therefore the fine tracking of the speed reference is performed using only the command coming for the throttle. By fixing the gear ratio in fifth gear we calibrate the parameters k_t, k_b, and i_t on the resulting linear system ($k_t = 70, k_b = 20$, and $i_t = 10$). The HYSDEL model of the car is reported in the Appendix, and it is available together with the cruise control system in the HYSDEL distribution [33]. The corresponding MLD model (22) has 173 MLD constraints, $x \in \mathbb{R}^2 \times \{0,1\}^5, d \in \{0,1\}^{15}, z \in \mathbb{R}^{19}$.

2) Verification: The HYSDEL compiler is used to generate a PWA model of the cruise control system. The verification is performed using the algorithm presented in [13]. We check the above mentioned safety requirement, namely that the cruise control will never accelerate the car over the speed limits. As the safety specification is independent of the car position we omit this from the model and we use the following initial set $\mathcal{X}(0) = \{x = \begin{bmatrix} v \\ e \end{bmatrix} : v \in [0,1], e \in [0,1]\}$ and target set: $\mathcal{Z}_1 = \{v : v > v_r + r_{\text{toll}}\}$ where r_{toll} is a tolerance, in this example we set $r_{\text{toll}} = 1.3889$ m·s^{-1} (5 km/h) that is for instance the tolerance of the speed limit enforcement devices adopted in Switzerland. Moreover, we check the liveness of the controller by adding the set $\mathcal{Z}_2 = \{v, t : v \leq v_r - 2r_{\text{toll}}, t > 10/T_s\}$, where we require that the controller reaches the target speed minus the tolerance $2r_{\text{toll}}$ in 10 s (a controller that stops the car would be safe against fines, but not at all desirable!). We perform parametric verification [13] for a class of constant references $v_r \in [8.333, 19.444]$ m/s (=[30, 70] km/h). The exploration horizon is fixed to $T_{\max} = 15.5$ s ($K_{\max} = T_{\max}/T_s = 31$ steps).

3) Verification Results: The result of the verification algorithm is that the controlled system satisfies both the specifications: It does not enter the unsafe region $\mathcal{Z}_1$ (over the speed limit) and guarantees the liveness of the control action (within 10 s the speed v is in a bounded set around the target speed v_r. The verification required 9109 s on a PC Pentium 650 MHz running Matlab 5.3.

The algorithm was run also for the same initial and target sets and for an extended range of the parameter $v_r \in [30, 120]$ km/h. The algorithm reported the first counterexample in 415 s: for $v_r = 105.0012$ km/h) the liveness condition is not satisfied.

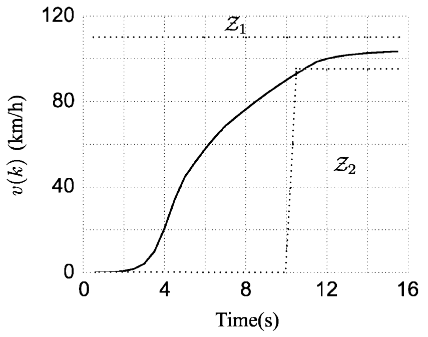


Fig. 8. Counterexample to the liveness property.

In fact, by examining the plot of the counterexample reported in Fig. 8, one can see that the controller fails to reach the requested vehicle speed within the specified time frame.

C. Cruise Control Design

Now, we replace the heuristic controller with an optimal one. We use the hybrid model of the car to synthesize a cruise control system that commands the gear ratio (discrete input) and gas pedal and brakes (continuous inputs) in order to track a desired speed and minimize fuel consumption. To this end, we design a receding horizon controller and derive its equivalent explicit piecewise affine form [23], so that the cruise controller becomes a look-up table of affine functions of the measured velocity and reference signals, that can be easily implemented in real time. Since the controller does not depend on the position of the car, we will remove x from the model.

The main idea of the approach is to setup a finite-horizon optimal control problem for the hybrid MLD system (22) by optimizing a performance index under operating constraints. In particular, we minimize

$$\min_{\xi} J(v(k), v_d(k)) \triangleq |v'(k) - v_d(k)| + \rho|\omega(k)| \quad (27a)$$

$$\text{subj. to} \begin{cases} v'(k) = Av(k) + B_1u(k) \\ \qquad +B_2\delta(k) + B_3z(k), \\ E_2\delta(k) + E_3z(k) \leq E_1u(k) \\ \qquad +E_4x(k) + E_5, \end{cases} \quad (27b)$$

where $v(k)$ is the measured velocity of the car at time $t = kT_s$, and $\xi \triangleq [u^T(k), \delta^T(k), z^T(k)]^T$ is the optimization vector.

As remarked above, the design of the controller is performed in two steps. First, the RHC controller based on the optimal control problem (27) is tuned in simulation using MILP solvers [36], until the desired performance is achieved. The RHC controller is not directly implementable, as it would require a MILP to be solved on-line, which is clearly prohibitive on standard automotive control hardware. Therefore, for implementation, in the second phase the explicit piecewise affine form of the RHC law is computed off-line by using a multiparametric mixed integer linear programming (mp-MILP) solver, according to the approach of [23], which provides the optimal control action as a piecewise affine function of the measured (or estimated) state vector of the hybrid system and reference signals. As a result, the state + reference space is partitioned into polyhedral sets, where an affine control law is defined in each polyhedron.

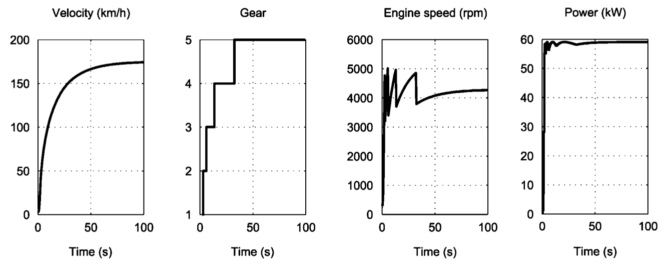


Fig. 9. Maximum acceleration profiles.

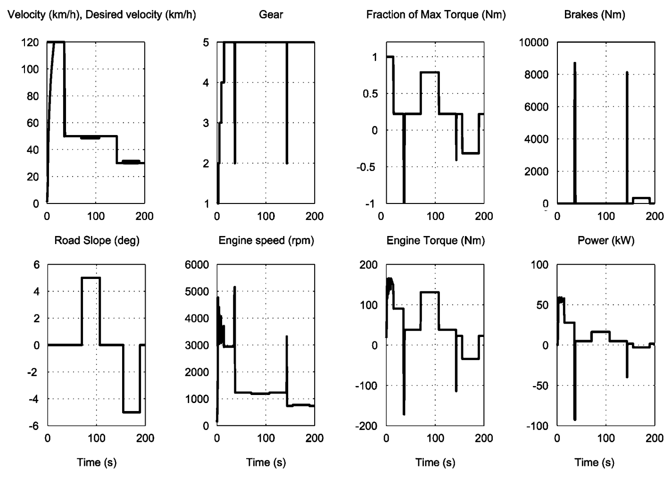


Fig. 10. Closed-loop profiles: aggressive control action.

As a first design step, we choose $\rho = 0.001$ m/rad in (27a). The corresponding multiparametric mixed-integer linear programming has 98 linear inequalities, 19 continuous variables, ten binary variables, two parameters $(v(k), v_d(k))$, and is solved in 27 m on a Sun Ultra 10 running Matlab 5.3 and Cplex. The corresponding piecewise affine control law consists of 49 regions.

For a commanded speed $v_d = 250$ km/h, which cannot be reached by the car, the cruise controller leads to the maximum acceleration curves depicted in Fig. 9, that are very close to those reported in the data sheets.

Fig. 10 shows the closed-loop trajectories for a few changes of the velocity set-point. During the shift from 0 to 120 km/h, the cruise controller commands the gears similarly to what shown in Fig. 9, with fullthrottle and no action on the brakes. When the set-point changes from 120 km/h to 50 km/h, the cruise controller does its best to slow down the car: switch to second gear, use full brakes, release the gas pedal. As soon as the set-point is recovered, the weight ρ on the fuel consumption leads back to fifth gear. The simulation also includes a nonzero road slope, which acts as an unmeasured and unmodeled disturbance to be rejected by the cruise controller.

Clearly, the controller is too aggressive during the set-point transition. This can be easily fixed by adding in (27) the constraint

$$|v'(k) - v(k)| < T_s a_{\max}$$

where $a_{\max}$ is the maximum acceleration tolerated. The resulting MILP problem has 100 linear inequalities, and is solved multiparametrically in 28 m, leading to a partition of the (v, v_d) space into 54 regions. The corresponding closed-loop

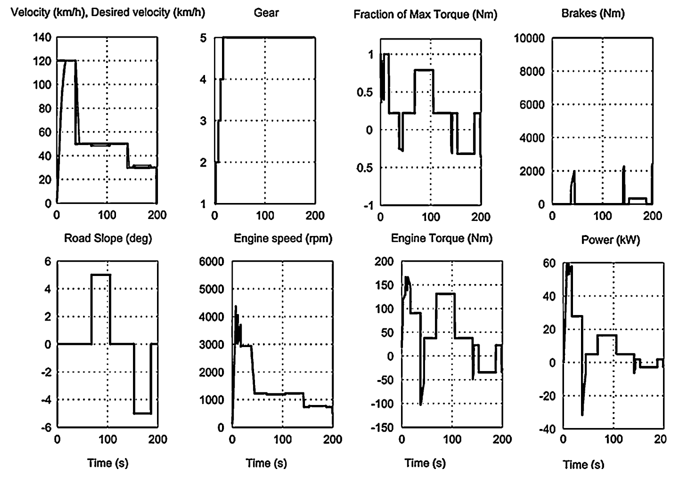


Fig. 11. Closed-loop profiles: smoother control action.

trajectories are depicted in Fig. 11, where a better drive comfort is clearly apparent.

We remark that the cruise control system described in this section has to be considered as a pure exercise of modeling and control synthesis for hybrid systems, and there is no claim that it is sensible, as it is, in a real application. For instance, it is apparent the rotation speed of the engine $\omega \approx 800$ rpm depicted in Fig. 10 would not be realistic for most commercial vehicles. A more comprehensive study for the synthesis of a supervisor for automatic gear shifting is currently under investigation in collaboration with the Fiat Research Center, Italy [37].

IX. Conclusion

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.

HYSDEL has been successfully used in several industrial applications. In [38], the authors modeled the hybrid behavior of a vehicle/tyre system and designed a traction controller that improves a driver's ability to control a vehicle under adverse external conditions such as wet or icy roads. Another automotive application was presented in [39], 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 [40], the economic optimization of a combined cycle power plant was accomplished by modeling the system in HYSDEL (turning on/off gas and steam turbines, operating constraints, different modalities for starting up of the turbines), and then using the generated MLD model in a mixed integer linear optimization algorithm [36].

The HYSDEL compiler is freely available for download at http://control.ee.ethz.ch/~hybrid/hysdel.

Appendix
HYSDEL Code—Car Model

```
SYSTEM car {INTERFACE {
  STATE {
    REAL position [-1000, 1000];
    REAL speed [-50 * 1000/3600, 220 * 1000/3600]; }
  INPUT {
    REAL torque [-300, 300]; /* Nm */
    REAL F_brake [0,9000]; /* N */
    REAL slope [0, 1];
    BOOL gear1, gear2, gear3, gear4, gear5, gearR; }
  OUTPUT {
    REAL position_y, speed_y, w_y; }
  PARAMETER {
Parameters omitted for brevity, full list available in [33].
  }}
IMPLEMENTATION {
  AUX {
    REAL Fe1, Fe2, Fe3, Fe4, Fe5, FeR, w1, w2, w3, w4,
        w5, wR, DCe1, DCe2, DCe3, DCe4;
    BOOL dPWL1, dPWL2, dPWL3, dPWL4; }
  AD {
    dPWL1 = wPWL1 - (w1 + w2 + w3 + w4 + w5 + wR) <= 0;
    dPWL2 = wPWL2 - (w1 + w2 + w3 + w4 + w5 + wR) <= 0;
    dPWL3 = wPWL3 - (w1 + w2 + w3 + w4 + w5 + wR) <= 0;
    dPWL4 = wPWL4 - (w1 + w2 + w3 + w4 + w5 + wR) <= 0; }
  DA {
    Fe1 = {IF gear1 THEN torque / sf * Rgear1};
    Fe2 = {IF gear2 THEN torque / sf * Rgear2};
    Fe3 = {IF gear3 THEN torque / sf * Rgear3};
    Fe4 = {IF gear4 THEN torque / sf * Rgear4};
    Fe5 = {IF gear5 THEN torque / sf * Rgear5};
    FeR = {IF gearR THEN torque / sf * RgearR};
    w1 = {IF gear1 THEN speed / sf * Rgear1};
    w2 = {IF gear2 THEN speed / sf * Rgear2};
    w3 = {IF gear3 THEN speed / sf * Rgear3};
    w4 = {IF gear4 THEN speed / sf * Rgear4};
    w5 = {IF gear5 THEN speed / sf * Rgear5};
    wR = {IF gearR THEN speed / sf * RgearR};
    DCe1 = {IF dPWL1 THEN (aPWL2 - aPWL1)+
      (bPWL2 - bPWL1) * (w1 + w2 + w3 + w4 + w5 + wR)};
    DCe2 = {IF dPWL2 THEN (aPWL3 - aPWL2)+
      (bPWL3 - bPWL2) * (w1 + w2 + w3 + w4 + w5 + wR)};
    DCe3 = {IF dPWL3 THEN (aPWL4 - aPWL3)+
      (bPWL4 - bPWL3) * (w1 + w2 + w3 + w4 + w5 + wR)};
    DCe4 = {IF dPWL4 THEN (aPWL5 - aPWL4)+
      (bPWL5 - bPWL4) * (w1 + w2 + w3 + w4 + w5 + wR)}; }
  CONTINUOUS {
    position = position + Ts * speed;
    speed = speed + Ts / mass * (Fe1 + Fe2 + Fe3 + Fe4 + Fe5
      + FeR - F_brake - beta_friction * speed) - g * slope; }
  OUTPUT {
    position_y = position;
    speed_y = speed;
    w_y = (w1 + w2 + w3 + w4 + w5 + wR); }
```

```
 MUST {
  w1 >= wemin; w1 <= wemax; w2 >= wemin; w2 <= wemax;
  w3 >= wemin; w3 <= wemax; w4 >= wemin; w4 <= wemax;
  w5 >= wemin; w5 <= wemax; wR >= wemin; wR <= wemax;
  -F_brake <= 0; F_brake <= max_brake_force;
  -torque - (alpha1 + beta1 * (w1 + w2 + w3 + w4 + w5+
    wR)) <= 0;
  torque - (aPWL1 + bPWL1 * (w1 + w2 + w3 + w4 + w5 + wR)
    +DCe1 + DCe2 + DCe3 + DCe4) - 1 <= 0;
  -((REAL gear1) + (REAL gear2) + (REAL gear3) + (REAL
    gear4) + (REAL gear5) + (REAL gearR)) <= -0.9999;
  (REAL gear1) + (REAL gear2) + (REAL gear3) + (REAL
    gear4) + (REAL gear5) + (REAL gearR) <= 1.0001;
  dPWL4- > dPWL3; dPWL4- > dPWL2; dPWL4- > dPWL1;
  dPWL3- > dPWL2; dPWL3- > dPWL1; dPWL2- > dPWL1; }
}}
```

Acknowledgment

The authors would like to thank Dr. D. Mignone for contributing many ideas and suggestions, Prof. M. Morari for fruitful discussions, Dr. F. Borrelli for providing his software for multiparametric mixed-integer linear programming, the more than 800 users that reported feedback on the HYSDEL compiler, Mr. G. Bertini, Mr. P. Hertach, and Mr. D. Jost, for helping in the implementation of HYSDEL, and Ms. F. Porro for drawing some of the figures reported in this paper.

References

[1] P. Antsaklis, Ed., *Special Issue on Hybrid Systems: Theory and Applications*, ser. Proc. IEEE, July 2000, vol. 88.

[2] R. Alur, C. Belta, F. Ivančić, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug, "Hybrid modeling and simulation of biomolecular networks," in *Hybrid Systems: Comp. and Contr.*. ser. Lecture Notes in Comp. Sc., M. Di Benedetto and A. S. Vincentelli, Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 19–33.

[3] J. P. Hespanha, S. Bohacek, K. Obraczka, and J. Lee, "Hybrid modeling of TCP congestion control," in *Hybrid Systems: Comp. and Contr.*. ser. Lecture Notes in Comp. Sc., M. Di Benedetto and A. S. Vincentelli, Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 291–304.

[4] K. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, "On the regularization of Zeno hybrid automata," *Syst. Control Lett.*, vol. 38, pp. 141–150, 1999.

[5] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*. ser. Lecture Notes in Comp. Sc., R. Grossman, A. Nerode, A. Ravn, and H. Rischel, Eds. New York: Springer-Verlag, 1993, vol. 736, pp. 209–229.

[6] B. Silva, O. Stursberg, B. Krogh, and S. Engell, "An assessment of the current status of algorithmic approaches to the verification of hybrid systems," in *Proc. 40th IEEE Conf. Decision Control*, Orlando, FL, Dec. 2001, pp. 2867–2874.

[7] M. Johansson and A. Rantzer, "Computation of piecewise quadratic lyapunov functions for hybrid systems," *IEEE Trans. Automat. Contr.*, vol. 43, pp. 555–559, Apr. 1998.

[8] X. Xu and P. Antsaklis, "Results and perspectives on computational methods for optimal control of switched systems," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sc., O. Maler and A. Pnueli, Eds. New York: Springer-Verlag, 2003, pp. 540–555.

[9] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, Mar. 1999.

[10] E. Sontag, "Nonlinear regulation: The piecewise linear approach," *IEEE Trans. Automat. Contr.*, vol. AC-26, pp. 346–358, Apr. 1981.

[11] W. Heemels, B. D. Schutter, and A. Bemporad, "Equivalence of hybrid dynamical models," *Automatica*, vol. 37, no. 7, pp. 1085–1091, July 2001.

[12] F. Torrisi, A. Bemporad, and L. Giovanardi. (2003) Reach-Set computation for analysis and optimal control of discrete hybrid automata. Automatic Control Laboratory, ETH, Zurich, Switzerland. [Online]. Available: http://control.ee.ethz.ch/

[13] A. Bemporad, F. Torrisi, and M. Morari, "Discrete-time hybrid modeling and verification of the batch evaporator process benchmark," *Eur. J. Control*, vol. 7, no. 4, pp. 382–399, 2001.

[14] A. Bemporad, D. Mignone, and M. Morari, "Moving horizon estimation for hybrid systems and fault detection," presented at the Amer. Control Conf., Albuquerque, NM, 1999.

[15] G. Ferrari-Trecate, D. Mignone, and M. Morari, "Moving horizon estimation for hybrid systems," *IEEE Trans. Automat. Contr.*, vol. 47, pp. 1663–1676, Oct. 2002.

[16] D. Mignone, "Control and estimation of hybrid systems with mathematical optimization," Ph.D. dissertation, Automatic Control Labotatory, ETH, Zurich, Switzerland, 2002.

[17] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari, "A clustering technique for the identification of piecewise affine systems," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sci., M. Di Benedetto and A. S. Vincentelli, Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 218–231.

[18] A. Bemporad, J. Roll, and L. Ljung, "Identification of hybrid systems via mixed-integer programming," in *Proc. 40th IEEE Conf. Decision Control*, Orlando, FL, Dec. 2001, pp. 786–792.

[19] P. Julián, M. Jordan, and A. Desages, "Canonical piecewise-linear approximation of smooth functions," *IEEE Trans. Circuits Syst. I*, vol. 45, pp. 567–571, May 1998.

[20] A. Bemporad, A. Garulli, S. Paoletti, and A. Vicino, "A greedy approach to identification of piecewise affine models," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sci., O. Maler and A. Pnueli, Eds. New York: Springer-Verlag, 2003, vol. 2623, pp. 97–112.

[21] H. Williams, *Model Building in Mathematical Programming*, 3rd ed. New York: Wiley, 1993.

[22] R. Raman and I. Grossmann, "Relation between MILP modeling and logical inference for chemical process synthesis," *Comput. Chem. Eng.*, vol. 15, no. 2, pp. 73–84, 1991.

[23] A. Bemporad, F. Borrelli, and M. Morari, "Piecewise linear optimal controllers for hybrid systems," in *Proc. American Control Conf.*, Chicago, IL, June 2000, pp. 1190–1194.

[24] T. Park and P. Barton, "Implicit model checking of logic-based control systems," *Aiche J.*, vol. 43, no. 9, pp. 2246–2260, 1997.

[25] G. Mitra, C. Lucas, S. Moody, and E. Hadjiconstantinou, "Tool for reformulating logical forms into zero-one mixed integer programs," *Eur. J. Oper. Res.*, vol. 72, pp. 262–276, 1994.

[26] T. Huerlimann, Reference manual for the LPL modeling language, Dept. Informatics, Université de Fribourg, Fribourg, Switzerland, 2001. Version 4.42.

[27] V. Chandru and J. Hooker, *Optimization Methods for Logical Inference*. New York: Wiley, 1999.

[28] D. Mignone, A. Bemporad, and M. Morari, "A framework for control, fault detection, state estimation and verification of hybrid systems," in *Amer. Control Conf.*, Albuquerque, NM, 1999.

[29] K. Fukuda, Cdd/cdd + Reference Manual, Inst. Operations Research, ETH, Zurich, Switzerland, Dec. 1997.

[30] V. Dimitriadis, N. Shah, and C. Pantelides, "Modeling and safety verification of discrete/continuous processing systems," *Aiche J.*, vol. 43, pp. 1041–1059, 1997.

[31] T. Geyer, F. Torrisi, and M. Morari, "Efficient mode enumeration of compositional hybrid models," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sc., O. Maler and A. Pnueli, Eds. New York: Springer-Verlag, 2003, vol. 2623, pp. 216–232.

[32] A. Bemporad, "Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form," *IEEE Trans. Automat. Contr.*, 2004, to be published.

[33] F. Torrisi, A. Bemporad, G. Bertini, P. Hertach, D. Jost, and D. Mignone. (2002) HYSDEL 2.0.5—User Manual. Automatic Control Laboratory, ETH, Zurich, Switzerland. [Online]. Available: http://control.ee.ethz.ch/

[34] *Automotive Handbook*, 5th ed., Society of Automotive Engineers, Warrendale, PA, Dec. 2000.

[35] P. Julián, "A high level canonical piecewise linear representation: Theory and applications," Ph.D. dissertation, Univerisidad Nacional del Sur, Argentina, May 1999. MATLAB toolbox available online at.

[36] *CPLEX 8.0 User Manual*, ILOG, Inc., Gentilly, France, 2002.

[37] A. Bemporad, P. Borodani, and M. Mannelli, "Hybrid control of an automotive robotized gearbox for reduction of consumptions and emissions," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sc., O. Maler and A. Pnueli, Eds. New York: Springer-Verlag, 2003, pp. 81–96.

[38] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat, "A hybrid approach to traction control," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sci., M. Di Benedetto and A. S. Vincentelli, Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 162–174.

[39] A. Bemporad, N. Giorgetti, I. Kolmanovsky, and D. Hrovat, "A hybrid systems approach to modeling and optimal control of DISC engines," presented at the 41st IEEE Conf. Decision Control, Las Vegas, NV, Dec. 2002.

[40] 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*. ser. Lecture Notes in Comp. Sci., C. J. Tomlin and M. R. Greenstreet, Eds. New York: Springer-Verlag, 2002, vol. 2289, pp. 209–224.

Fabio Danilo Torrisi received the master degree in computer engineering from the University of Florence, Florence, Italy, in 1999, and the Ph.D. degree from the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, in 2003.

His research interests include hybrid systems, computational geometry, model predictive control, software engineering, and embedded systems.

Alberto Bemporad (S'93–M'99) received the M.S. degree in electrical engineering and the Ph.D. degree in control engineering from the University of Florence, Florence, Italy, in 1993 and 1997, respectively.

He spent the academic year 1996–1997 at the Center for Robotics and Automation, Department of Systems Science and Mathematics, Washington University, St. Louis, MO, as a Visiting Researcher. In 1997–1999, he held a postdoctoral position at the Automatic Control Laboratory, ETH, Zurich, Switzerland, where he collaborated as a Senior Researcher in 2000–2002. Since 1999, he has been an Assistant Professor at the University of Siena, Siena, Italy. He has published papers in the area of hybrid systems, model predictive control, computational geometry, and robotics. He is coauthoring the new version of the Model Predictive Control Toolbox (The Mathworks, Inc., Natick, MA).

Dr. Bemporad is an Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and Chair of the IEEE Control Systems Society Technical Committee on Hybrid Systems.