Mathematical and Computer Modelling of Dynamical Systems
2004, Vol. 10, No. 1, pp. 41-57

A New Technique for Translating Discrete Hybrid Automata into Piecewise Affine Systems

BOŠTJAN POTOČNIK[*,1], GAŠPER MUŠIČ[*] AND BORUT ZUPANČIČ[*]

SUMMARY

The paper proposes a new translation algorithm that translates a hybrid system described as a *discrete hybrid automaton* (DHA) into an equivalent *piecewise affine* (PWA) system. The translation algorithm exploits, among others, a new technique for cell enumeration in hyperplane arrangement, all proposed in this paper. The new translation technique enables the transfer of several analysis and synthesis tools developed for PWA systems to a DHA class of hybrid systems.

Keywords: hybrid systems, modelling, piecewise affine systems, cell enumeration.

1. INTRODUCTION

Hybrid systems are dynamic systems that involve the interaction of continuous dynamics (modelled as differential or difference equations) and discrete dynamics (modelled by finite state machines). Hybrid systems have been the topic of intense research activity in recent years, primarily because of their potential importance in applications, e.g. the process industry. Hybrid models are important to a number of problems in system analysis, such as computation of trajectories, control, stability and safety analysis, etc.

Mathematical models represent the basis to any system analysis and design such as simulation, control, verification, etc. Several modelling formalisms for hybrid dynamical systems have been proposed in the literature [1, 10, 17] and each class of models is usually appropriate only for solving a certain problem. Among others, we would like to emphasise the Piecewise Affine (PWA) systems [21], Mixed Logical Dynamical (MLD) systems [7] and Discrete Hybrid Automata (DHA) [24], which are receiving increasing attention by the control community. The importance lies in the equivalent relations among them that allow the transfer of all analysis and synthesis tools developed for one particular class to any of the other equivalent subclasses of hybrid systems. In [14] the authors prove the equivalence between PWA, MLD, Linear Complementarity (LC) [15] and other classes of hybrid systems. Attention has

[*]Address correspondance to: Faculty of Electrical Engineering, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia.
[1]Corresponding author. Tel: +386 1 4768 764; Fax: +386 1 4264 631; e-mail: bostjan.potocnik@fe.uni-lj.si

to be given also to DHA systems, which can be translated into an MLD modelling framework using the tool HYSDEL (HYbrid System DEscription Language) [24].

In this paper, a new technique for translation of a hybrid dynamical system, modelled as a DHA system, into a PWA system will be presented. The translation algorithm takes into account a DHA model described by the HYSDEL modelling language. A DHA model is translated into a special structure, also containing an MLD model of a system, using the HYSDEL tool. Using the resulting structure, a PWA system is built. The paper is organised as follows: In Section 2, we introduce DHA, MLD and PWA systems. A DHA system represents the basis for transformation into MLD and PWA systems. In Section 3, we introduce several improvements and novelties to the technique for translating DHA into a PWA model. The efficiency of the proposed translation technique, tested on a car example adopted by several authors [4, 5, 13], is presented in Section 4. The conclusions are given in Section 5.

2. DHA, MLD AND PWA SYSTEMS

The emphasis will be given on the discrete-time representation of DHA, MLD and PWA systems. The algorithms to obtain a discrete-time PWA representation of an MLD system, and vice versa, are reported in [8, 4, 5]. The algorithm to obtain a discrete-time PWA representation of a DHA system is reported in [13]. The same topic will be addressed in this paper where we propose a new technique for translating DHA systems into PWA, which performs better than the one presented in [13]. Because of the close relations between all three modelling formalisms (DHA (HYSDEL)→ MLD, MLD → PWA, DHA (HYSDEL) → (MLD) → PWA)) and, due to better understanding on the topic, short descriptions to each of the three formalisms will be provided in the sequel.

2.1. Discrete Hybrid Automata (DHA)

According to [24] a *Discrete hybrid automaton* (DHA) is the interconnection of a *finite state machine* (FSM), which provides the discrete part of the hybrid system, with a *switched affine system* (SAS) providing the continuous part of the hybrid dynamics. The interaction between the two is based on two connecting elements: the *event generator* (EG), which extracts logic signals from the continuous part, and *mode selector*, which defines the mode (continuous dynamics) of the SAS based on logic variables (states, inputs and events). The DHA system is shown on Figure 1.

A switched affine system (SAS) represents a sampled continuous system that is described by the following set of linear affine equations:

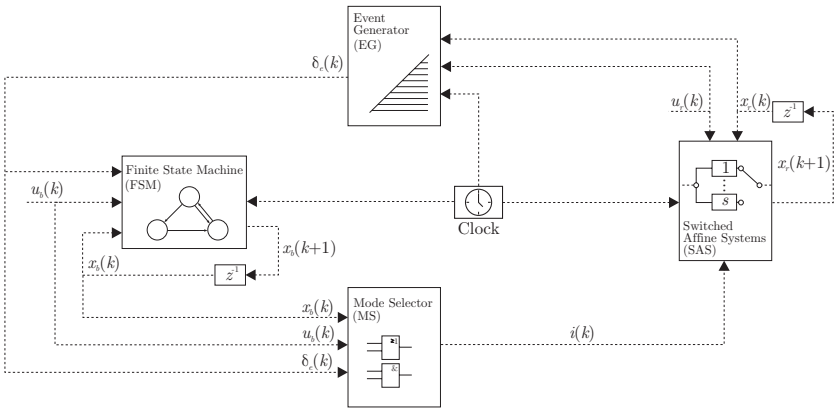


Fig. 1. A discrete hybrid automaton (DHA)

$$x_r(k+1) = A_{i(k)}x_r(k) + B_{i(k)}u_r(k) + f_{i(k)} \tag{1a}$$

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

where $k \in \mathbb{Z}_{\geq 0}$ represents the independent variable (time step) ($\mathbb{Z}_{\geq 0} \triangleq \{0, 1, ...\}$ is a set of nonnegative integers), $x_r \in \mathcal{X}_r \subseteq \mathbb{R}^{n_r}$ is the continuous state vector, $u_r \in \mathcal{U}_r \subseteq \mathbb{R}^{m_r}$ is the 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 set of matrices of suitable dimensions, and $i(k) \in \mathcal{I}$ is a variable that selects the linear state update dynamics. A SAS of the form (1) changes the state update equation when the switch occurs, i.e. $i(k) \in \mathcal{I}$ changes. An SAS can be also rewritten as the combination of linear terms and *if-then-else* rules. The state-update function (1a) can also be written as:

$$z_1(k) = \begin{cases} A_1x_r(k) + B_1u_r(k) + f_1 & \text{if } i(k) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2a}$$

$$\vdots$$

$$z_s(k) = \begin{cases} A_sx_r(k) + B_su_r(k) + f_s & \text{if } i(k) = s \\ 0 & \text{otherwise} \end{cases} \tag{2b}$$

$$x_r(k+1) = \sum_{i=1}^{s} z_i(k). \tag{2c}$$

An event generator (EG) generates a logic signal according to the satisfaction of linear affine constraints:

$$\delta_e(k) = f_H(x_r(k), u_r(k), k), \tag{3}$$

where $f_H : \mathbb{R}^{n_r} \times \mathbb{R}^{m_r} \times \mathbb{Z}_{\geq 0} \to \mathcal{D} \subseteq \{0,1\}^{n_e}$ is a vector of descriptive functions of a linear hyperplane. The relation f_H for time events is modelled as $[\delta_e^i(k) = 1] \leftrightarrow [kT_s \geq t_i]$, where T_s is the sampling time, while for threshold events is modelled as $[\delta_e^i(k) = 1] \leftrightarrow [a_i^T x_r(k) + b_i^T u_r(k) \leq c_i]$, and where a_i, b_i, c_i represent the parameters of a linear hyperplane. δ_e^i denotes the i-th component of a vector $\delta_e(k)$.

A finite state machine (FSM) is a discrete dynamic process that evolves according to a logic state update function:

$$x_b(k+1) = f_B(x_b(k), u_b(k), \delta_e(k)), \tag{4}$$

where $x_b \in \mathcal{X}_b \subseteq \{0,1\}^{n_b}$ is the binary state (we use the term binary instead of Boolean to emphasise the fact that $x_b \in \{0,1\}^{n_b}$), $u_b \in \mathcal{U}_b \subseteq \{0,1\}^{m_b}$ is the binary input, $\delta_e(k)$ is the input coming from the EG, and $f_B : \mathcal{X}_b \times \mathcal{U}_b \times \mathcal{D} \to \mathcal{X}_b$ is a deterministic logic function. An FSM may have also associated binary output

$$y_b(k) = g_B(x_b(k), u_b(k), \delta_e(k)), \tag{5}$$

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

A mode selector (MS) selects the dynamic mode $i(k)$ of the SAS according to the binary state $x_b(k)$, the binary inputs $u_b(k)$ and the events $\delta_e(k)$ using Boolean function $f_M : \mathcal{X}_b \times \mathcal{U}_b \times \mathcal{D} \to \mathcal{I}$. The output of this function

$$i(k) = f_M(x_b(k), u_b(k), \delta_e(k)) \tag{6}$$

is called the *active mode*.

HYbrid System DEscription Language (HYSDEL)

DHA models can be modelled by using modelling language called HYSDEL as shown in ([24]). The HYSDEL description of hybrid systems represents an abstract modelling step. Once the system is modelled as DHA, i.e. described by the HYSDEL language, the model can be translated into an MLD model using an associated HYSDEL compiler. At this point, we will give just a brief introduction into the structure of a HYSDEL list.

A HYSDEL list is composed of two parts: the INTERFACE, where all the variables and parameters are declared, and the IMPLEMENTATION, which consists of

specialised sections, where the relations between the variables are defined. Let us introduce the most significant sections. The AD section allows the definition of binary variables and is based on the semantics of the *event generator* (EG), i.e. in the AD section the δ_e variables are defined. The LOGIC section allows the specification of arbitrary functions of binary variables. Since the *mode selector* is defined as a Boolean function, it can be defined in this section. The DA section defines the switching of the continuous variables according to *if-then-else* rules depending on binary variables, i.e. part of *switched affine system* (SAS), namely z_i variables (see Equation (2)) are defined. The CONTINUOUS section defines the linear dynamics expressed as difference equations, i.e. defines the remaining Equation (2c) of the SAS. The LINEAR section defines continuous variables as an affine function of continuous variables, which in combination with the DA and the CONTINUOUS section enables more flexibility when modelling SAS. The AUTOMATA section specifies the state transition equations of the *finite state machine* (FMS) as a Boolean function (4), i.e. defines binary variables x_b. The MUST section specifies constraints on continuous and binary variables, i.e. defines the sets $\mathcal{X}_r, \mathcal{X}_b, \mathcal{U}_r$ and $\mathcal{U}_b$.

For more detailed description on the functionality of the modelling language HYSDEL and the associated compiler (tool HYSDEL), the reader is referred to [24, 25].

2.2. Mixed Logical Dynamical (MLD) Systems

In [7] a class of hybrid systems, called *Mixed Logical Dynamical* (MLD) systems, has been introduced in which logic, dynamics and constraints are integrated. An MLD system is described by the following relations:

$$x(k+1) = Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k) \tag{7a}$$

$$y(k) = Cx(k) + D_1u(k) + D_2\delta(k)D_3z(k) \tag{7b}$$

$$E_2\delta(k) + E_3z(k) \leq E_1u(k) + E_4x(k) + E_5, \tag{7c}$$

where $x(k) = [x_r^T(k)\; x_b^T(k)]^T$ ($x_r(k) \in \mathbb{R}^{n_r}$, $x_b(k) \in \{0,1\}^{n_b}$) are the states of a system, outputs $y(k)$ and inputs $u(k)$ have similar decomposition as $x(k)$, $z(k) \in \mathbb{R}^{r_r}$ are real and $\delta(k) \in \{0,1\}^{r_b}$ binary auxiliary variables. A, B_1, B_2, B_3, C, D_1, D_2, D_3, $E_1, ..., E_5$ are real constant matrices with suitable dimensions.

Using the current state $x(k)$ and input $u(k)$, the time evolution of (7) is determined by solving $\delta(k)$ and $z(k)$ from (7c), and then updating $x(k+1)$ and $y(k)$ from Equations (7a) and (7b), respectively. The MLD system (7) is assumed to be completely well-posed if for a given state $x(k)$ and input $u(k)$ the inequalities (7c) have a unique solution for $\delta(k)$ and $z(k)$. A simple algorithm to test well-posedness is given in [7].

2.3. Piecewise Affine (PWA) Systems

A discrete-time piecewise affine system is defined by the state-space equations

$$\begin{aligned} x(k+1) &= A_i x(k) + B_i u(k) + f_i \\ y(k) &= C_i x(k) + D_i u(k) + g_i \end{aligned} \qquad \text{for } \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \in \mathcal{X}_i, \tag{8}$$

where $x(k) \in \mathbb{R}^{n_r}$ is the state, $u(k) \in \mathbb{R}^{m_r}$ is the input, $y(k) \in \mathbb{R}^{p_r}$ is the output, at time instance k, $\{\mathcal{X}_i\}_{i=1,\dots,s}$ is a polyhedral partition of the state-input space defined by a system of inequalities $\{H_x^i x + H_u^i u \leq K^i\}$. $A_i, B_i, f_i, C_i, D_i, g_i, H_x^i, H_u^i$ and K^i are real matrices of suitable dimensions for all i. A PWA system (8) is well-posed if $x(k+1)$ and $y(k)$ have a unique solution for a given state $x(k)$ and input $u(k)$, i.e. $\mathcal{X}_i \cap \mathcal{X}_j = \varnothing \ \forall i \neq j$, $\cup_{i=1}^{s} \mathcal{X}_i = \mathbb{X}$.

For a more detailed description about PWA systems, the reader is referred to [21, 12, 16] and the references therein.

3. A NEW DHA (MLD) TO PWA TRANSLATION ALGORITHM

Two different translation approaches have already been proposed in the literature. These are the translation of an MLD to a PWA form [4, 5] and a translation of a DHA to a PWA form [13]. The motivation for these investigations lies in the fact that many analysis and synthesis tools were developed for PWA systems. For instance, stability criteria for PWA systems were proposed in [16, 18], observability and controllability was studied in [8], reachability analysis in [2] and synthesis of optimal control laws in [6].

The approach presented in this paper solves the problem similar to [4], where the author translates an MLD model into an equivalent PWA model using multi-parametric and mixed-integer programming, and deals with the same problem as in [13], where authors translate DHA into a PWA form. Due to the characteristics of the problem of translating DHA form of a hybrid system into PWA form the structure of the translation algorithm is similar to one in [13]. Compared to the technique presented in [13], we introduce several novelties and improvements to the translation algorithm to improve the performance. The performance of the translation algorithm is very important as the complexity of the translation grows with the hybrid system complexity, especially with the number of discrete states.

The first improvement presented in this paper is a new cell enumeration algorithm, which is presented in Section 3.2, with lower time complexity than the one used in [13]. The second improvement is the algorithm that defines (finds) feasible combinations of binary states and inputs, which represents very important part of the translation technique. The description of the problem and the solution algorithm is given

in Section 3.3. The third improvement (novelty) is a new post-processing operation, presented in Section 3.5, which reduces the number of polyhedral cells in the PWA system, i.e. merging the regions (cells) with the same affine dynamics and where the union remains convex.

3.1. DHA and PWA Systems

For better understanding, a summary of the equivalence between DHA and PWA systems, proven in [13], will be given. By considering fixed binary variables $\bar{\delta}_e, \bar{x}_b$ and $\bar{u}_b$, we obtain the following PWA system:

$$x_r(k+1) = A_{f_M(\bar{x}_b,\bar{u}_b,\bar{\delta}_e)}x_r(k) + B_{f_M(\bar{x}_b,\bar{u}_b,\bar{\delta}_e)}u(k) + f_{f_M(\bar{x}_b,\bar{u}_b,\bar{\delta}_e)} \tag{9a}$$

$$x_b(k+1) = f_B(\bar{x}_b, \bar{u}_b, \bar{\delta}_e) \tag{9b}$$

$$y_r(k) = C_{f_M(\bar{x}_b,\bar{u}_b,\bar{\delta}_e)}x_r(k) + D_{f_M(\bar{x}_b,\bar{u}_b,\bar{\delta}_e)}u(k) + g_{f_M(\bar{x}_b,\bar{u}_b,\bar{\delta}_e)} \tag{9c}$$

$$y_b(k) = g_B(\bar{x}_b, \bar{u}_b, \bar{\delta}_e) \tag{9d}$$

$$\text{if } x_b(k) = \bar{x}_b, \qquad u_b(k) = \bar{u}_b, \qquad [x_r^T(k)\, u_r^T(k)]^T \in \mathcal{X}_{\bar{\delta}_e}, \tag{9e}$$

where the functions f_M, f_B and g_B are defined by Equations (4),(5) and (6) respectively. By collecting $x = [\begin{smallmatrix} x_r \\ x_b \end{smallmatrix}]$ and $y = [\begin{smallmatrix} y_r \\ y_b \end{smallmatrix}]$ the system (9) is formally equivalent to PWA system (8). The question is how to define $\bar{\delta}_e$, $\bar{x}_b$ and $\bar{u}_b$? By considering the worst case, there exist $2^{(n_e+n_b+m_b)}$ possible combinations of the binary variables (δ_e, x_b, u_b), but in general most of them are infeasible due to the system operating limitations. The solution to the given problem will be given in the following sections.

3.2. Defining binary variable δ_e

In this section, we will address the problem of defining the binary variable δ_e that is defined by Equation (3) that represents the event generator (EG). The problem of defining the binary variable δ_e according to the fulfilment of the relation f_H in (3) can be described as a cell enumeration problem. To each cell described by one set of inequalities of the form $a^T x_r(k) + b^T u_r(k) \leq c$ exactly one combination of the binary variable δ_e is assigned.

In [13], the authors adopted an algorithm that enumerates all feasible modes for binary variable δ_e, i.e. defines a set $\mathcal{D}$, using an efficient algorithm for cell enumeration in hyperplane arrangement presented in [3, 11]. In the sequel, we will propose a new technique for cell enumeration, which is better performing than the existing one.

A new technique for cell enumeration in a hyperplane arrangement

Here, we will adopt the notation to the problem of enumerating the cells as presented in [3]. Let $\mathcal{A}$ be an arrangement of n distinct hyperplanes $\{H_i\}_{i=1,\ldots,n}$ in $\mathbb{R}^d$, where each hyperplane is given by a linear equality $H_i = \{x : a_i x = b_i\}$, where a_i and b_i represent the i-th row of A and b, respectively. For each $x \in \mathbb{R}^d$, the sign vector $SV(x)$ is the vector in $\{+, 0, -\}^n$ defined by

$$SV(x)_i = \begin{cases} + & \text{if } a_i x < b_i \\ 0 & \text{if } a_i x = b_i \\ - & \text{if } a_i x > b_i \end{cases} \qquad \text{for } i \in \{1, ..., n\}. \tag{10}$$

Figure 2 shows an example of $n = 4$ hyperplanes in $\mathbb{R}^2$ with appropriate sign vectors. Let $\mathcal{S}$ be the set of sign vectors s where $s = \{SV(x) : x \in \mathbb{R}^d\}$ and the set $C_s = \{x : SV(x) = s\}$ be a subspace of the state space for a given sign vector s. The set C_s is called a *cell* of the arrangement and is a polyhedron defined by equalities and inequalities. According to the way the time and threshold events are modelled (see Section 2.1- *Event Generator*), we reformulate the definition of $SV(x)$ (see (10)) without loss of generality to:

$$SV(x)_i = \begin{cases} + & \text{if } a_i x \leq b_i \\ - & \text{if } a_i x > b_i. \end{cases} \qquad \text{for } i \in \{1, ..., n\} \tag{11}$$

Actually this step is necessary to embrace all possible situations correctly, i.e. to define all possible cells. It is obvious that the definition (10) yields a larger set $\mathcal{S}$ than it is of our interest, i.e. many sign vectors with zero entries are irrelevant, but not all, e.g., a hyperplane itself may represent a *cell*.

We say that hyperplanes $\{H_i\}_{i=1,\ldots,n}$ are in a *general position* if there are no parallel hyperplanes and if any point in $\mathbb{R}^d$ belongs, to at most, d hyperplanes. The upper bound of the number of hyperplanes is defined by [9]:

$$\begin{aligned} &\sharp C_{(n,d)} \leq \sum_{i=0}^{d} \binom{n}{i} \text{ for } n \geq d \text{ and} \\ &\sharp C_{(n,d)} \leq 2^n \text{ for } n < d \end{aligned} \tag{12}$$

or recursively $\sharp C_{(n,d)} \leq \sharp C_{(n-1,d)} + \sharp C_{(n-1,d-1)}$, where $\sharp C_{(0,d)} = \sharp C_{(n,0)} = 1$. The equality holds when the hyperplanes are in the *general position*.

To summarise, the goal is to enumerate all the cells defined by a hyperplane arrangement. To this end, we propose the following cell enumeration algorithm:

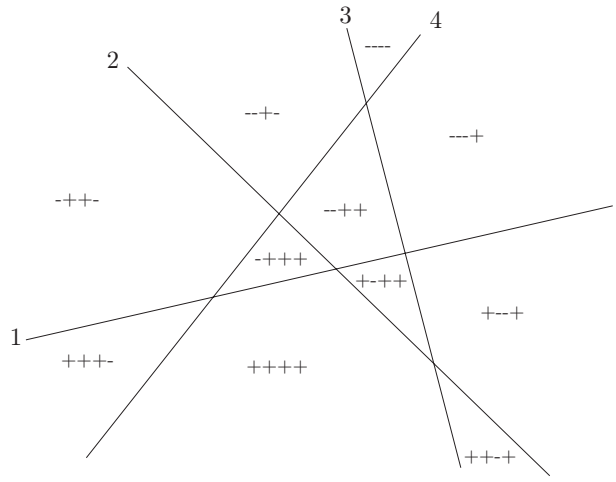


Fig. 2. An arrangement of four hyperplanes in $\mathbb{R}^2$ with the sign vectors

Algorithm 1:

Initialise
Get(A, b);
$\mathcal{S} = \begin{bmatrix} -1 & -1 & ... & -1 \end{bmatrix}$; (* initialise the set $\mathcal{S}$ *)
for $i = 1$ **to** n **do**
    $numS$=NumRows($\mathcal{S}$); (* current number of cells *)
    $\mathcal{S} = [\mathcal{S}; \mathcal{S}]$; (* add a hyperplane; double the number of cells *)
    for $j = 1$ **to** $numS$
        $\mathcal{S}(j, i) = 1$; (* adjust the signs to a half of the cells *)
    end
    for $k = 1$ **to** $2 \cdot numS$
        CheckFeasibility($\mathcal{S}(k, 1 : i), A(1 : i, :), b(1 : i, :)$);
        if Infeasible
            RemoveFromList($\mathcal{S}(k, :)$);
        end
    end
end

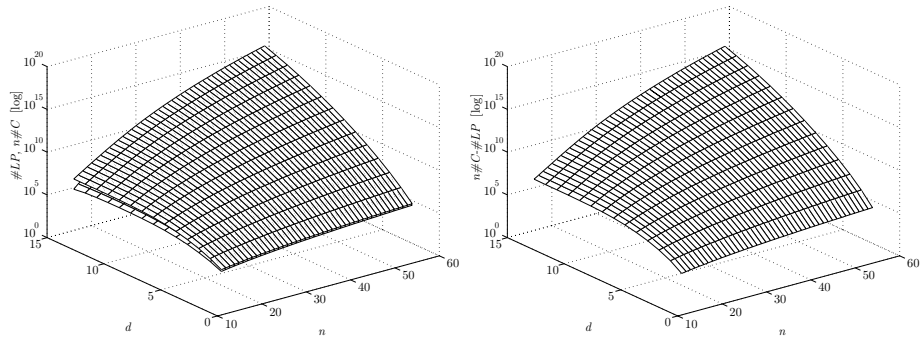
Algorithm 1 solves the cell enumeration problem by taking one hyperplane at a time. When a hyperplane is added, the number of cells is doubled due to splitting

the state space. This is handled by duplicating the set $\mathcal{S}$ and assigning the correct sign values related to the added hyperplane (1 corresponds to "$+$" and -1 to "$-$", see Equation (11)). To remove the infeasible sign vectors from the set $\mathcal{S}$, the feasibility test is run for all defined sign vectors. Only the part of $\mathcal{S}$, A and b related to the processed hyperplanes is considered. As soon as all hyperplanes are processed, the feasible set of signs $\mathcal{S}$ is defined and, consequently, the set of cell arrangements $\mathcal{C}_s$ is defined as well.

The worst case time complexity of Algorithm 1 is estimated to $O(\text{lp}(n,d) \cdot \sharp LP_{(n,d)})$, where $\text{lp}(n,d)$ is the time needed to run one feasibility test, using *linear program* (LP), for n hyperplanes in d dimensions. $\sharp LP_{(n,d)}$ is the number of feasibility tests (LP runs) for (n,d) problem and is calculated as:

$$\sharp LP_{(n,d)} \leq 2 \sum_{i=1}^{n-1} \sharp C_{(i,d)}. \tag{13}$$

The equality holds when the hyperplanes are in *general position*. The worst case time complexity of the proposed algorithm is lower than the one proposed in [3], improved in [11] and used in [13], where the time complexity is estimated to $O(n \cdot \text{lp}(n,d) \cdot \sharp C_{(n,d)})$. Considering Equations (12) and (13), we can write $n \cdot \sharp C_{(n,d)} - \sharp LP_{(n,d)} = \sharp C_{(n,d)} + \sum_{i=1}^{n-1} \left[\, i \cdot \sharp C_{(i,d-1)} - \sharp C_{(i,d)} \right]$, which implies $\sharp LP_{(n,d)} < n \cdot \sharp C_{(n,d)}$. The comparison of the worst case time complexities for both algorithms is shown in Figure 3.



(a) Comparison: $\sharp LP$, $n\sharp C$ (b) Difference: $n\sharp C - \sharp LP$

Fig. 3. The comparison (a) and the difference (b) of the time complexities (note that the z-axis is in logarithmic scale)

From Figure 3, it can be noticed that by increasing the number of hyperplanes n and the dimension d of the cell enumeration problem, the difference of the worst case

time complexities of the algorithms grows very fast. Of course, the calculated worst case complexity represents an estimation of the upper bound. For this reason, one may consider if there are cases where the hyperplanes are in general position and the algorithm applied in [13] is faster than the one proposed here. Considering the difference of the worst case time complexity (see Figure 3(b)), it is obvious that by increasing the number of hyperplanes n or dimension d of the enumeration problem the possibility of this happening is decreasing very rapidly. Besides, the difference in the performance of both approaches for small problems (low number of hyperplanes/dimension) is negligible. More interesting is the case where the number of cells in an arrangement is much lower compared to the maximum number of cells estimated using Equation (12). This happens when the hyperplanes are not in general position, e.g. some of the hyperplanes are parallel to each other. Sorting the hyperplanes in proper order, e.g. parallel hyperplanes first, which is easy to implement, can additionally improve the performance of the cell enumeration algorithm presented in this paper. The algorithm adopted in [13] starts the enumeration of cells from a cell that is randomly selected [3, 11] and, therefore, by rearranging the hyperplanes the performance of the algorithm cannot be improved.

3.3. Defining binary states x_b and inputs u_b

The cell enumeration algorithm presented in the previous section will be used to define binary variable δ_e in a DHA to PWA translation algorithm. We still have to define all feasible combinations of binary variables (x_b, u_b), i.e. to each binary variable $\delta_e \in \mathcal{D}$ a feasible combination of binary states and inputs $x_b \in \mathcal{X}_b, u_b \in \mathcal{U}_b$ has to be associated in order to define a PWA submodel (see Equation (9)). In the worst case, we have to run $\sharp C \cdot 2^{(n_b+m_b)}$ feasibility tests, i.e. we have to test all possible combinations of binary variables (x_b, u_b), for each feasible combination of the vector $\delta_e \in \mathcal{D}$. Of course, the number of feasible combinations of binary variables (x_b, u_b) can be much lower due to logic constraints, which can include also a combination with binary variable δ_e.

In [13], the authors define feasible combinations of binary variables (x_b, u_b) by applying binary search. By looking at the MATLAB implementation of the DHA to PWA translation algorithm, Piecewise Affine (PWA) Plugin: `hys2pwa 1.1.3`, which is available on the internet site `http://control.ee.ethz.ch/ hybrid/hysdel/`, the authors check for feasibility for each possible combination of (x_b, u_b), i.e. run worst case $2^{(n_b+m_b)}$ logic feasibility tests (without applying linear program) considering logic constraints on binary variables (x_b, u_b).

In this paper, we propose a different approach to defining all feasible combinations of binary variables (x_b, u_b). The proposed algorithm uses the same principle as it is used in the cell enumeration algorithm presented previously. One element of binary variables (x_b, u_b) is taken at once. Each next element of (x_b, u_b) is added to the set of possible combinations of already defined elements by extending the current set

with new possible combinations. The resulting set is checked for feasibility using appropriate logic constraints (without applying linear program), i.e. constraints for which all the elements are defined, and only the feasible combinations remain in the set. The procedure is repeated until all the feasible combinations of binary variables (x_b, u_b) are defined. To this end, we propose the following algorithm:

Algorithm 2:

Initialise
Get(A_{xu}, b_{xu}); (* logic constraints $A_{xu}[x_b^T, u_b^T]^T \leq b_{xu}$ *)
$\mathcal{XU} = [0\ 0\ ...\ 0]$; (* initialise the set $\mathcal{XU}$ *)
$n_{xu} = NumberOfElements(x_b, u_b)$;
for $i = 1$ **to** n_{xu} **do**
  $numXU$=NumRows($\mathcal{XU}$); (* number of feasible combinations *)
  (* add a binary variable x_b or u_b *)
  $\mathcal{XU} = [\mathcal{XU}; \mathcal{XU}]$; (* double the number of combinations *)
  for $j = 1$ **to** $numXU$
    $\mathcal{XU}(j, i) = 1$; (* set first half to 1 *)
  end
  for $k = 1$ **to** $2 \cdot numXU$
    rows=FindCorespondingConstraints(A_{xu}, b_{xu});
    if NotEmpty(rows)
      CheckLogicFeasibility($\mathcal{XU}, A_{xu}(rows, :), b_{xu}(rows, :)$);
      if Infeasible
        RemoveFromList($\mathcal{XU}(k, :)$);
      end
    end
  end
end,

where $\mathcal{XU}$ is a set of feasible combinations for (x_b, u_b).

Every time new element is defined Algorithm 2 removes infeasible combinations according to constraints valid for already defined elements. This enables the control of the growth of possible combinations of binary variables (x_b, u_b) constrained by logic constraints. In the worst case, if there are no logic constraints, the growth is exponential, but we don't have to run logic feasibility tests. By increasing the number of logic constraints the number of logic feasibility tests is increasing too, but the growth of possible combinations of binary variables (x_b, u_b) is decreasing and that is the most important. Thus, the complexity of Algorithm 2 depends on the composition and the

number of logic constraints and is therefore different from case to case. The performance can be even improved, by sorting the components of the binary vectors (x_b, u_b) according to logic constraints, i.e. we first sort the components, which appear in the rows of $[A_{xu}, b_{xu}]$ with the least elements. In comparison with the cell enumeration algorithm presented previously, the concept is the same but in this case, the infeasible combinations are not defined by hyperplanes but with logic constraints.

Considering that we have already defined a feasible set $\mathcal{D}$ from $\mathcal{S}$ applying Algorithm 1, we can extend Algorithm 2 such that it includes that available information, as usually the logic constraints include δ_e, x_b and u_b variables. This can be done by initialising Algorithm 2 with $\mathcal{XU} = [\mathcal{D}, \mathbf{0}]$ and by considering expanded logic constraints $A_{\delta_e xu}$ and $b_{\delta_e xu}$. This enables online removal of infeasible combinations of binary variables (δ_e, x_b, u_b) while searching for feasible combinations of (x_b, u_b). The number of logic feasibility tests is, therefore, much lower than $\sharp C \cdot 2^{n_b+m_b}$. The given extension is implemented in DHA to PWA translation algorithm given in the sequel.

3.4. DHA to PWA translation algorithm

Once a hybrid system is modelled using HYSDEL modelling language the resulting DHA model can be transformed into a special structure [25], which defines an MLD model of a system, but can also be used for a starting point for other algorithms translating DHA into other computational models. The structure contains, besides the MLD form, additional information about the relations between the variables (continuous and binary (Boolean)), the section in which they are defined (AD, LOGIC, ...) and *computable order*. The *computable order* defines the priority for computation. Variables with lower *computable order* have to be defined prior to the variables with higher order. The resulting structure is used in the translation algorithm from DHA to PWA.

For a better understanding of the translation algorithm, which will be given in the sequel, we introduce the following variables and sets: δ_{AD} represents binary variables defined in AD section and are equivalent to the δ_e defined previously; the set $\mathcal{AD} = \{\delta^i_{AD}\}_{i=1,\dots,\sharp C}$ contains all possible combinations of the variable δ_{AD}; δ_{LO} represents binary variables defined in LOGIC section and are used as auxiliary binary variables; the set $\mathcal{LO} = \{\delta^i_{LO}\}_{i=1,\dots,\sharp LO}$ contains all possible combinations of the variable δ_{LO}; z_{DA} represents continuous variables defined in DA section; the set $\mathcal{DA} = \{z^i_{DA}\}_{i=1,\dots,\sharp DA}$ contains all possible combinations of the variable z_{DA}, where each element z^i_{DA} is represented by the set of matrices $\{K^i_1, K^i_2, K^i_3\}$, subject to $z^i_{DA} = K^i_1 x + K^i_2 u + K^i_3$; z_{LI} represents continuous variables defined in LINEAR section; the set $\mathcal{LI} = \{z^i_{LI}\}_{i=1,\dots,\sharp LI}$ contains all possible combinations of the variable z_{LI}, which is defined analogically to z_{DA}, and sets $\mathcal{XB} = \{x^i_b\}_{i=1,\dots,\sharp XB}$ $\mathcal{UB} = \{u^i_b\}_{i=1,\dots,\sharp UB}$ contain all possible combinations of the variables x_b and u_b respectively. The meaning of the variables is strongly correlated with the sections in which they are defined, and consequently to the DHA form.

The translation algorithm from DHA to PWA is described by the following algorithm:

Algorithm 3:

Initialise
H=HysdelStructure(DHA); (* using HYSDEL *)
MaxOrder=GetMaxOrder(H); (* find maximal computable order *)
Initialise $\{\mathcal{DA}, \mathcal{LI}, \mathcal{LO}\}$;
Initialise $\{\mathcal{ADXU}\}$; (* (δ_{AD}, x_b, u_b) *)
for i=1 **to** MaxOrder **do**
  for all(H.AD.CompOrder = i) **do**
    define δ_{AD} variables using **Algorithm 1**;
    check logic constraints, update $\{\mathcal{ADXU}\}$;
  end
  for all(H.LO.CompOrder = i) **do**
    if not all elements in x_b, u_b required for δ_{LO} are defined,
      define required elements in x_b, u_b using **Algorithm 2**;
      check logic constraints, update $\{\mathcal{ADXU}\}$;
    end
    define δ_{LO}, check logic constraints, update $\{\mathcal{LO}\}$;
  end
  for all(H.DA.CompOrder = i) **do**
    if not all elements in x_b, u_b required for z_{DA} are defined,
      define required elements in x_b, u_b using **Algorithm 2**;
      check logic constraints, update $\{\mathcal{ADXU}\}$;
    end
    define z_{DA}, update $\{\mathcal{DA}\}$;
  end
  for all(H.LI.CompOrder = i) **do**
    define z_{LI}, update $\{\mathcal{LI}\}$;
  end
end
(* Set $\{\mathcal{ADXU}\}$ is defined *)
for all$((\delta_{AD}, x_b, u_b) \in \{\mathcal{ADXU}\})$ **do**
  define PWA system according to (9);
end
check for all the regions in PWA with the same affine

dynamics for possible merging into common convex regions;

Algorithm 3 enumerates all feasible modes for variables (δ_e, x_b, u_b), (note that δ_e is associated to δ_{AD}) and builds corresponding PWA model. Feasible modes for $\delta_e = f_H(x_r, u_r, z_{DA}, z_{LI}, k)$ are enumerated using Algorithm 1, while feasible modes for (x_b, u_b) are enumerated using Algorithm 2. As soon as all feasible combinations for variables (δ_e, x_b, u_b) are defined, the PWA form of a DHA system is defined. Finally, the merging of the regions of the PWA system with the same affine dynamics into common convex regions may be applied.

3.5. Merging the regions with the same affine dynamics

The number of polyhedral regions (cells) in the PWA system can be reduced by merging the regions with the same affine dynamics into common convex regions. The proposed procedure is based on the *Quine-McCluskey* [20, 19] minimisation algorithm of Boolean functions.

Considering Equation (9), we notice that the PWA dynamic is defined by three Boolean functions f_M, f_B and g_B that are defined by the variables x_b, u_b and δ_e (δ_{AD}). Let us group feasible combinations of variables (δ_e, x_b, u_b) into sets $M_{aff} = \{(\delta_e, x_b, u_b) : f_M(\delta_e, x_b, u_b) = f_M^{aff},\ f_B(\delta_e, x_b, u_b) = f_B^{aff},\ g_B(\delta_e, x_b, u_b) = g_B^{aff},\ D_{aff}[\delta_e^T, x_b^T, u_b^T]^T = d_{aff}\}$, for a given affine dynamics f_M^{aff}, f_B^{aff}, g_b^{aff} and fixed combination of some binary variables (δ_e, x_b, u_b) defined by $D_{aff}[\delta_e, x_b^T, u_b^T]^T = d_{aff}$. In general, certain components of binary vector δ_e can be indirectly dependent on some components of binary variables(δ_e, x_b, u_b) and by merging these cells we can not guarantee that the resulting cell will remain convex. The matrix D_{aff} and the vector d_{aff} are introduced to embrace those indirectly dependent components of binary vector δ_e.

By considering each set M_{aff} as a Boolean function represented by the truth table, the *Quine-McCluskey* minimisation algorithm of Boolean functions can be used. The result is the reduced representation $M_{aff}^{reduced}$ of the set M_{aff}, i.e. the reduced number of regions that define the same affine dynamics. By modifying the PWA system according to reduced representations for all $M_{aff}^{reduced}$ the reduced PWA system is obtained. If there is no dependence of some components of binary variable δ_e on some components of binary variables (δ_e, x_b, u_b), i.e. $D_{aff} = \mathbf{0}$ and $d_{aff} = \mathbf{0}$, the obtained reduced PWA system is in minimal form.

4. AN EXAMPLE

The efficiency of the proposed translation technique was tested on a hybrid model of a car with a robotised gear shift, presented in [23, 24]. The HYSDEL model of the car is reported in [24] and [4]. The car model has $n_r = 2$ continuous states (position and velocity of the car), no logic states, $m_r = 2$ continuous inputs (engine torque and braking force), $m_b = 6$ binary inputs (gears $\sharp 1, ..., \sharp 5$ + reverse gear).

Applying Algorithm 3 to the car example returns a PWA structure containing 30 regions, i.e. 30 PWA submodels, and is computed in 1.45 seconds using MATLAB 5.3 on a Pentium III 667 MHz machine. This is 50 times faster than reported in [4] and 5 times faster than reported in [13], considering that all three approaches run on a similar machine.

5. CONCLUSIONS

In this paper, a new technique was proposed for translating DHA systems described by the HYSDEL modelling language into equivalent PWA systems. The technique is based on the new cell enumeration in hyperplane arrangement algorithm presented in Section 3.2, on the new approach to defining all feasible combinations of binary variables (x_b, u_b) presented in Section 3.3 and on the merging algorithm to reduce the size of a PWA representation presented in Section 3.5. An efficient translation technique is important to increase the usability of several analysis and synthesis tools developed for PWA systems while using the tool HYSDEL in which relatively complex hybrid systems composed of linear dynamics, finite state machines, propositional logic, etc., can be described. The proposed algorithm can be used to translate all kinds of HYSDEL models into PWA form.

ACKNOWLEDGEMENTS

The authors thank to Prof. Alberto Bemporad for inspiring discussions.

REFERENCES

1. Antsaklis, P.J.: A Brief Introduction to the Theory and Applications of Hybrid Systems *Proceedings of the IEEE, Special Issue on Hybrid Systems: Theory and Applications*, 88(7) (2000), pp. 879–887.
2. Asarin, E., Bournez, O., Dang, T. and Maler, O.: Approximate Reachability Analysis of Piecewise-Linear Dynamical Systems. *in Hybrid Systems: Computation and Control*, B. Krogh and N. Lynch, Eds., Volume 1790 of Lecture Notes in Computer Science, pp. 20–31, Springer Verlag, (2000).
3. Avis, D. and Fukuda, K.: Reverse search for enumeration, *Discrete Applied Mathematics*, 65 (1996), pp. 21–46.

4. Bemporad, A.: An efficient technique for translating mixed logical dynamical systems into piecewise affine systems, *in Proc. 41th IEEE Conf. on Decision and Control*, (2002), pp. 1970–1975.
5. Bemporad, A.: Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form, *IEEE Trans. Automatic Control*, (2003), in press.
6. Bemporad, A., Borrelli, F. and Morari, M.: On the optimal control law for linear discrete time hybrid systems, *in Hybrid Systems: Computation and Control*, M. Greenstreet and C. Tomlin, Eds., Volume 2289 of Lecture Notes in Computer Science, pp. 105–119, Springer Verlag, (2002).
7. Bemporad, A. and Morari, M.: Control of systems integrating logic, dynamic, and constraints. *Automatica*, 35(3) (1999), pp. 407–427.
8. Bemporad, A., Ferrari-Trecate, G. and Morari, M.: Observability and controllability of piecewise affine and hybrid systems, *IEEE Trans. Automatic Control*, 45(10) (2000), pp. 1864–1876.
9. Buck, R.C.: Partition of space, *Amer. Math. Monthly*, 50 (1943), pp. 541–544.
10. Engell, S., Frehse, G. and Schnieder, E.: Modelling, Analysis and Design of Hybrid Systems, *Volume 279 of Lecture Notes in Control and Information Sciences*, Springer Verlag, (2002).
11. Ferrez, J.A., Fukuda, K. and Liebling, Th.M.: Cuts, Zonotopes and arrangements, Technical report, (2001), EPF, Lausanne, Switzerland.
12. Ferrari-Trecate, G., Cuzzola, F.A., Mignone, D. and Morari, M.: Analysis of discrete-time piecewise affine and hybrid systems, *Automatica*, 38(12) (2002), pp. 2139–2146.
13. Geyer T., Torrisi, F.D. and Morari, M.: Efficient Mode Enumeration of Compositional Hybrid Models, *in Hybrid Systems: Computation and Control*, O. Maler and A. Pnueli, Eds., Volume 2623 of Lecture Notes in Computer Science, pp. 216–232, Springer Verlag, (2003).
14. Heemels, W.P.M.H., De Schutter, B. and Bemporad, A.: Equivalence of hybrid dynamical models, *Automatica*, 37(7) (2001), pp. 1085–1091.
15. Heemels, W.P.M.H., Schumacher, J.M. and Weiland, S.M.: Linear complementarity systems, *SIAM Journal on Applied Mathematics*, 60(4) (2000), pp. 1234–1269.
16. Johansson, M. and Rantzer A.: Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Trans. on Automatic Control*, 43(4) (1998), pp. 555-559.
17. Labinaz G., Bayoumi, M.M. and Rudie, K.: A Survey of modelling and Control of Hybrid Systems, *Annual Reviews in Control*, 21 (1997), pp. 79–92.
18. Mignone, D., Ferrari-Trecate, G. and Morari, M.: Stability and Stabilization of Piecewise Affine and Hybrid Systems: An LMI Approach, *Proc. 39th Conference on Decision and Control*, (2000), Sydney, Australia.
19. McCIuskey, E.J.: Minimization of Boolean functions, *Bell System Technical Journal*, 35 (1956), pp. 1417–1444.
20. Quine, W.V.: The Problem of Simplifying Truth Functions, *American Mathematical Monthly*, 59 (1952), pp. 521–531.
21. Sontag, E.D.: Nonlinear regulation: The piecewise linear approach. *IEEE Trans. Automatic Control*, 26(2) (1981), pp. 346–358.
22. Sontag, E.D.: Interconnected automata and linear systems: A theoretical framework in discrete-time, *in Hybrid Systems III: Verification and Control*, R. Alur, T. Henzinger, and E.D. Sontag, Eds., Volume 1066 of Lecture Notes in Computer Science, pp. 436-448, Springer Verlag, (1996).
23. Torrisi, F.D. and Bemporad, A.: Discrete-time hybrid modelling and verification, *in Proc. 40th IEEE Conf. on Decision and Control*, (2001), pp. 2899–2904, Orlando, Florida.
24. Torrisi, F.D. and Bemporad, A.: HYSDEL - A Tool for Generating Computational Hybrid Models for Analysis and Synthesis Problems, submited to *IEEE TCST - Special Issue on Computer Automated Multi-Paradigm modelling*, (2002).
25. Torrisi, F.D., Bemporad, A., Bertini, G., Hertach, P., Jost, D. and Mignone, D.: HYSDEL - User Manual, Technical report AUT02-10, (2002), ETH Zrich, Switzerland.