Verification of Hybrid Systems via Mathematical Programming

Alberto Bemporad and Manfred Morari

Institut für Automatik, Swiss Federal Institute of Technology
ETH Zentrum - ETL I29, CH 8092 Zürich, Switzerland
tel. +41-1-632 7626, fax +41-1-632 1211
{bemporad,morari}@aut.ee.ethz.ch
http://control.ethz.ch

Abstract. This paper proposes a novel approach to the verification of hybrid systems based on linear and mixed-integer linear programming. Models are described using the Mixed Logical Dynamical (MLD) formalism introduced in [5]. The proposed technique is demonstrated on a verification case study for an automotive suspension system.

1 Introduction

Hybrid models describe processes evolving according to dynamics and logic rules. The adjective "hybrid" stems from the fact that both continuous and discrete quantities are needed to describe the behavior of the process at hand. Hybrid systems have recently grown in interest not only for being theoretically challenging, but also for their impact on applications. Although many physical phenomena are hybrid in nature, the main interest is directed to real-time systems, where physical processes are controlled by embedded controllers. For this reason, it is important to have available tools to guarantee that this combination behaves as desired. Verification algorithms for hybrid systems are aimed at providing such a certification.

Most of the literature about verification of hybrid systems originates from the artificial intelligence realm, and solvers rely on symbolic computation. In this paper, we propose an approach stemming from system science and propose a solver based on mathematical programming. As an example application, we report a case study on an automotive suspension system.

2 Mixed Logic Dynamic (MLD) Systems

The mixed logic dynamic (MLD) form has been introduced in [5]. It is a modeling framework that allows to describe various classes of systems, like finite state machines interacting with dynamic systems, piecewise linear systems, systems with mixed discrete/continuous inputs and states, systems with qualitative outputs, and so on. Physical constraints, constraint prioritization, and heuristics

can also be included in the description of the system. For details, we defer to [5]. Here we only give the general MLD form

$$x(t+1) = Ax(t) + B_1 w(t) + B_2 \delta(t) + B_3 z(t) \tag{1a}$$

$$y(t) = Cx(t) + D_1 w(t) + D_2 \delta(t) + D_3 z(t) \tag{1b}$$

$$E_2 \delta(t) + E_3 z(t) \leq E_1 w(t) + E_4 x(t) + E_5 \tag{1c}$$

where $x = \left[\begin{smallmatrix} x_c \\ x_\ell \end{smallmatrix}\right]$, is the state of the system, whose components are distinguished between continuous $x_c \in \mathbb{R}^{n_c}$ and logical $x_\ell \in \{0,1\}^{n_\ell}$; $y = \left[\begin{smallmatrix} y_c \\ y_\ell \end{smallmatrix}\right]$, $y_c \in \mathbb{R}^{p_c}$ $y_\ell \in \{0,1\}^{p_\ell}$, is the output vector collecting quantities of interest, $w = \left[\begin{smallmatrix} w_c \\ w_\ell \end{smallmatrix}\right]$ is a vector of disturbances entering the system, collecting both continuous disturbances $w_c \in \mathbb{R}^{m_c}$, and binary disturbances $w_\ell \in \{0,1\}^{m_\ell}$ (e.g. faults [4]); $\delta \in \{0,1\}^{r_\ell}$ and $z \in \mathbb{R}^{r_c}$ represent auxiliary logical and continuous variables respectively. The auxiliary variables are introduced whenever logic propositions are translated into linear inequalities. The key idea of the approach is in fact to transform logic statements into mixed-integer linear inequalities. For instance: "$X_1 \wedge X_2$ FALSE" becomes "$\delta_1 + \delta_2 \leq 1$", or "if X_1 TRUE then Pressure $P_1 \leq P_0$" becomes "$P_1 - P_0 \leq M(1 - \delta_1)$", where M is a large number. All these constraints are summarized in the inequality (1c). Note that the description (1) is only apparently linear because of the integrality constraints. Also, the form (1) involves linear discrete-time dynamics. One might formulate a continuous time version by replacing $x(t+1)$ by $\dot{x}(t)$ in (1a), or a nonlinear version by changing the linear equations and inequalities in (1) to more general nonlinear functions. We restrict the dynamics to be linear and discrete-time in order to obtain computationally tractable schemes. Nevertheless, we believe that this framework permits the description of a very broad class of systems.

3 Automatic Verification of MLD Systems

Consider a linear discrete-time hybrid system of the form (1). Given a set of initial states $\mathcal{X}(0)$ and a set of disturbances $\mathcal{W}$, consider the following *Verification Problems*:

VP1 Verify that $\forall w \in \mathcal{W}$ and $\forall x(0) \in \mathcal{X}(0)$ the state $x(t) \in \mathcal{X}_s$, where $\mathcal{X}_s$ is an assigned set of *safe* states

VP2 Find the maximum range for $y(t)$

$$\max_{t \geq 0, w(t) \in W, x(0) \in \mathcal{X}(0)} \{C_c x_c(t) + C_\ell x_\ell(t) + D_1 w(t) + D_2 \delta(t) + D_3 z(t)\}$$

3.1 A Simple but Numerically Impractical Solution

In principle, problem **VP1** can be addressed by solving $\forall T \geq 0$ the following Mixed-Integer Feasibility Test (MIFT)

$$\begin{cases} x(0) \in \mathcal{X}(0) \\ x(T) \notin \mathcal{X}_s \\ w(t) \in \mathcal{W} \\ x(t+1) = Ax(t) + B_1 w(t) + B_2 \delta(t) + B_3 z(t) \\ E_2 \delta(t) + E_3 z(t) \leq E_1 w(t) + E_4 x(t) + E_5 \\ 0 \leq t \leq T \end{cases} \tag{2}$$

and **VP2** through the Mixed-Integer Program (MIP)

$$\max_{x(0), \{w(t), \delta(t), z(t)\}_{t=0}^{T}} Cx(T) + D_1 w(T) + D_2 \delta(T) + D_3 z(T)$$
$$\text{subj. to} \begin{cases} x(0) \in \mathcal{X}(0) \\ w(t) \in \mathcal{W},\ 0 \leq t \leq T \\ x(t+1) = Ax(t) + B_1 w(t) + B_2 \delta(t) + B_3 z(t) \\ E_2 \delta(t) + E_3 z(t) \leq E_1 w(t) + E_4 x(t) + E_5 \end{cases} \tag{3}$$

Even in the case of polyhedral sets $\mathcal{W}$, $\mathcal{X}_s$, $\mathcal{X}(0)$, solving the MIFT (2) and the Mixed Integer Linear Program (MILP)–(3) for large T becomes prohibitive. In fact, each problem (2) is $\mathcal{NP}$-complete because of the presence of integer variables [5], which means that in the worst case the required computation time grows exponentially with T [12].

3.2 A General Procedure for Verification of MLD Systems

The numerical complexities discussed above are due to the presence of free integer variables in the optimization problems. Note, however, that the binary variables δ are related to conditions on the continuous states x_c. Therefore, a trajectory of the hybrid system (1) can be partitioned in subtrajectories with $\delta(t) \equiv$ const, and analogously with $x_\ell(t) \equiv$ const. With this idea in mind, we consider a hypothetic partition of the continuous state space $\mathbb{R}^{n_c}$ in subregions $\mathcal{C}_i$ where system (1) evolves with $\delta(t) \equiv \delta_i$, $x_\ell(t) \equiv x_{\ell i}$, namely

$$\mathcal{C}_i = \{x_c \in \mathcal{X}_c :\ \exists z \in \mathbb{R}^{r_c}, w \in \mathbb{R}^m, \text{ such that} \\ E_3 z \leq E_1 w + E_4 \left[\begin{smallmatrix} x_c \\ x_{\ell i} \end{smallmatrix}\right] + E_5 - E_2 \delta_i\}$$

The number of these subregions is at most $2^{r_\ell + n_\ell}$, corresponding to all 0,1 combinations of each component of vector $\left[\begin{smallmatrix} \delta \\ x_\ell \end{smallmatrix}\right]$. However, in general the number of nonempty sets $\mathcal{C}_i$ is much smaller, as most combinations will not fulfill the constraints stemming from translations of logical propositions (for instance, the logic proposition "$[\delta_1 = 1] \wedge [\delta_2 = 1]$ FALSE" becomes $\delta_1 + \delta_2 \leq 1$, which rules out the combination $(\delta_1, \delta_2) = (1, 1)$). Without loss of generality, we assume that the logical components x_ℓ of the state are of the form

$$x_\ell(t) = e_J \delta(t)$$

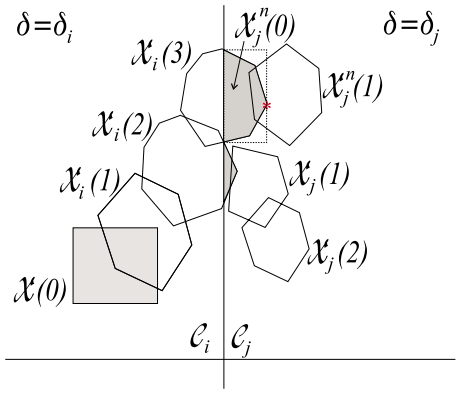


Fig. 1. Algorithm for verification.

where $e_J = [0, \dots, 0, 1, 0, \dots 0]$ is the J-th row of the $r_\ell \times r_\ell$ identity matrix. In fact, the state transition of logical states derives in general from a logic predicate involving literals associated with components of $\delta(t)$ and $x_\ell(t)$, and the latter can be expressed again as additional auxiliary variables $\delta_\ell(t)$, by simply adding the constraints $\delta_\ell(t) \le x_\ell(t)$, $-\delta_\ell(t) \ge -x_\ell(t)$ in (1c).

The main idea underlying the algorithm is sketched in Fig. 1 and can be summarized as follows. Assume that $\mathcal{X}(0) \subseteq \mathcal{C}_i$ for some $i \in \{0, \dots, 2^{r_\ell - 1}\}$. Then, consider the set $\mathcal{X}_i(t)$ of all possible evolutions from $\mathcal{X}(0)$ driven by $w(t) \in \mathcal{W}$ and such that $\delta(t) \equiv \delta_i$,

$$\mathcal{X}_i(t) = \{x : \ \exists x_c(0) \in \mathcal{X}(0), \{w(k)\}_{k=0}^{t} \text{ such that } x(k) \in \mathcal{C}_i, \ \forall k \le t\}.$$

Note that $\mathcal{X}_i(t) = \text{Reach}(t, \mathcal{X}(0)) \bigcap C_i$, where $\text{Reach}(t, \mathcal{X}(0))$ denotes the *reach set* from $\mathcal{X}(0)$. When at a certain time $\bar{t}$ there exists some state $x_c(\bar{t}) \in \text{Reach}(t, \mathcal{X}(0))$ such that $x_c(\bar{t}) \notin \mathcal{C}_i$, say $x_c(\bar{t}) \in \mathcal{C}_j$ (i.e. $\delta(\bar{t}) = \delta_j \neq \delta_i$ satisfies the constraints in (1) for some $w(\bar{t}) \in \mathcal{W}$, $z(\bar{t}) \in \mathbb{R}^{r_c}$), then $\mathcal{X}_j(0) \triangleq \text{Reach}(t, \mathcal{X}(0)) \bigcap \mathcal{C}_j$ is used to start the exploration of a new region with $\delta(t) \equiv \delta_j$. A certain evolution $\mathcal{X}_i(t)$ is considered explored or *fathomed* at time T_i when $\mathcal{X}_i(T_i) \subseteq \mathcal{X}_i(T_i - 1)$, i.e. the set $\mathcal{X}_i(T_i)$ has shrunk or is invariant, or is empty (i.e. all trajectories escape from $\mathcal{X}(T_i - 1)$). It may happen that during the exploration inside $\mathcal{C}_j$ at time t some states $x \in \mathcal{X}^* \triangleq \text{Reach}(t, \mathcal{X}_j(0)) \bigcap \mathcal{C}_i$ enter again a region $\mathcal{C}_i$ where an exploration has already been performed[1]. In this case, if $\mathcal{X}^* \not\subseteq \bigcup_{t=0}^{T_i} \mathcal{X}_i(t)$, $\forall t \le T_i$, a new exploration is performed from $\mathcal{X}_i^1(0) = \mathcal{X}^*$; otherwise no action is taken. The procedure stops only when all evolutions $\mathcal{X}_i^h(\cdot)$ have been explored. Problem **VP2** is solved by maximizing $C_c x_c(t) + D_1 w(t) + D_3 z(t)$ over $\mathcal{X}_i^h(t) \times \mathcal{W} \times \mathbb{R}^{r_c}$. When the aim is to solve **VP1**, the procedure stops when $\mathcal{X}_i^h(t) \times \{e_J \delta_i\} \bigcap \mathcal{X}_u \neq \emptyset$, where $\mathcal{X}_u$ is the complementary set of $\mathcal{X}_s$, i.e. the set of "unsafe" states. This algorithm is formulated in Table 1. Note that there is no

[1] If the system where in continuous time, then $\mathcal{X}^*$ would be a point or, at most, a polyhedron of dimension $n_c - 1$.

need to investigate a priori all possible regions $\mathcal{C}_i$, corresponding to all possible combinations of δ. The algorithm will explore only those combinations which are reachable from the assigned initial state $\mathcal{X}(0)$.

Algorithm 1.

0. Push problem $P_0^0 = \{i = 0,\ h = 0,\ \delta_i = 0,\ T_i^h = 0,\ \mathcal{X}_i^h(0) = \mathcal{X}(0) \subseteq \mathcal{C}_0\}$ on STACK. $Y_{\max} = [-\infty, \dots, -\infty]'$.
1. While STACK nonempty,
 1.1. Pop problem P_i^h from STACK.
 1.2. $t \leftarrow 0$.
 1.3. If $\mathcal{X}_i^h(0) \subseteq \bigcup_{\tau=1}^{T_j^k} \mathcal{X}_j^k(\tau)$, for some fathomed problem P_j^k, go to 1.
 1.4. $Y_{\max} \leftarrow \max\{Y_{\max}, \max_{x_c \in \mathcal{X}_i^h(t), w \in \mathcal{W}, z \in \mathbb{R}^{r_c}} C_c x_c + C_\ell x_{\ell i} + D_1 w + D_2 \delta_i + D_3 z\}$.
 1.5. If $\mathcal{X}_i^h(t) \times \{e_J \delta_i\} \bigcap \mathcal{X}_u \neq \emptyset$, system is unsafe. Stop.
 1.6. $t \leftarrow t + 1$.
 1.7. For all $\mathcal{C}_j \neq \mathcal{C}_i$ such that $\mathrm{Reach}(t, \mathcal{X}_i^h(0)) \bigcap \mathcal{C}_j \neq \emptyset$:
  1.7.1. $n \leftarrow \max\{k :\ P_j^k \text{ is on STACK}\} + 1$.
  1.7.2. $P_j^n \leftarrow \{j,\ n,\ \delta_j,\ T_j^n = 0,\ \mathcal{X}_j^n(0) = \mathrm{Reach}(t, \mathcal{X}_i^h(0)) \bigcap \mathcal{C}_j\}$.
  1.7.3. Push P_j^n on STACK.
 1.8. Compute $\mathcal{X}_i^h(t)$.
 1.9. If $\mathcal{X}_i^h(t) \subseteq \mathcal{X}_i^h(t-1)$ or $\mathcal{X}_i^h(t) = \emptyset$, fathom P_i^h and go to 1.
 1.10. Go to 1.4.
2. Stop.

Table 1. Basic algorithm for verification of hybrid systems.

As the problem of verification of hybrid systems is undecidable [1, 11], there is no guarantee that Algorithm 1 will terminate. However from a practical point of view decidability would not be enough, as there is no difference between a non-terminating procedure and a procedure which runs out of time or memory [2]. Nevertheless, when the proposed algorithm terminates, it provides an answer to **VP1**and **VP2**respectively.

4 Verification of MLD Systems Based on Mathematical Programming

Below we describe how Algorithm 1 can be implemented using Linear Programming (LP) and Mixed Integer Linear Programming (MILP). We assume that the set of disturbances $\mathcal{W}$ and the set of initial states $\mathcal{X}(0)$, as well as the safe set $\mathcal{X}_s \triangleq \{x :\ K_1 x \leq K_2\}$, are polyhedra, and that relations between continuous and logic variables have the form $[\delta = 1] \leftrightarrow [x_c \leq 0]$. The latter assumption, which is rather general and satisfied in many applications, allows one to search

for new subregions by simply minimizing and maximizing the components of the continuous part of the state [2].

Because of the linear form of MLD system (1), the evolution $x(t)$ can be expressed as

$$x(t) = A^t x_0 + \sum_{i=0}^{t-1} A^i \left[B_1 w(t-1-i) + B_2 \delta(t-1-i) + B_3 z(t-1-i)\right] \quad (4)$$

subject to (1c). Therefore, optimization of linear functions $f(x(t), w(t), \delta(t), z(t))$ results in an MILP if the variables $\delta(t)$ are free, or LP if $\delta(t)$ are fixed (for instance by setting $\delta(t) = \delta_i$ to enforce $x(t) \in \mathcal{C}_i$). The same holds for feasibility problems of the form $x(t) \in \mathcal{X}$, where $\mathcal{X}$ is a polyhedron. These considerations allow one to rewrite Algorithm 1 as follows.

Algorithm 2.

0. Push problem $P_0^0 = \{i = 0,\ h = 0,\ \delta_i = 0,\ T_i^h = 0,\ \mathcal{X}_i^h(0) = \mathcal{X}(0) \subseteq \mathcal{C}_0\}$ on STACK. $Y_{\max} = [-\infty, \dots, -\infty]'$.
1. While STACK nonempty,
  1.1. Pop problem P_i^h from STACK.
  1.2. $t \leftarrow 0$,

$$M_j(0) \leftarrow \max_{x \in \mathcal{X}_i^h(0)} \left[\begin{smallmatrix} x(t) \\ -x(t) \end{smallmatrix}\right]$$

  1.3. For all fathomed problems P_i^k:
    1.3.1. For $\tau = 0, \dots, T_i^k$
      1.3.1.1. For $v \in \{v_1, \dots, v_n\}$=vertices of $\mathcal{X}_i^h(0)$ solve the feasibility problem

$$\begin{cases} x_c(\tau) = v \\ (4)+(1c),\ \ t = 0, \dots, \tau \\ w(t) \in W \\ \delta(t) = \delta_i \\ x_\ell(0) = x_{\ell i} \\ x_c(0) \in \mathcal{X}_i^k(0) \end{cases}$$

      1.3.1.2. If feasible $\forall v$, fathom P_i^h and go to 1.
  1.4. Solve

$$\bar{Y} \leftarrow \begin{cases} \max_{\{w(k), z(k)\}_{k=0}^t, x_c(0)} C_c x_c(t) + D_1 w(t) + D_3 z(t) + [D_2 \delta_i + C_\ell x_{\ell i}] \\ \text{subj. to} \begin{cases} (4)+(1c),\ \ k = 0, \dots, t \\ w(k) \in W,\ \ k = 0, \dots, t \\ x_c(0) \in \mathcal{X}_i^h(0) \\ x_\ell(0) = x_{\ell i} \\ \delta(k) = \delta_i,\ \ k = 0, \dots, t \end{cases} \end{cases}$$

  and set $Y_{\max} \leftarrow \max\{Y_{\max}, \bar{Y}\}$

[2] The algorithm can be extended for $[\delta = 1] \leftrightarrow [Cx_c \le 0]$ by minimizing/maximizing $C^{[j]}x$, where $C^{[j]}$ denotes the j-th row of C. Equivalently, the algorithm described below can be used by defining new state components $x_{\text{aug}} = Cx_c$.

1.5. Solve the feasibility problem defined by the constraints in 1.4. and $K_1^{[j]} x(t) \geq K_2^{[j]}$, $j = 1, \ldots,$ number of rows of K_1. If any is feasible, system is unsafe. Stop.
1.6. $t \leftarrow t+1$
1.7. For $j = 1, \ldots, n_c$:
1.7.1. Solve

$$M_j(t) \leftarrow \begin{cases} \max_{\{w(k),z(k)\}_{k=0}^{t}, x_c(0), \delta(t)\}} \left[\begin{smallmatrix} x(t) \\ -x(t) \end{smallmatrix} \right] \\ \text{subj. to} \begin{cases} (4)+(1c), \ \ k = 0, \ldots, t \\ w(k) \in W, \ \ k = 0, \ldots, t \\ \delta(k) = \delta_i, \ \ k = 0, \ldots, t-1 \\ \delta(t) \in \{0,1\}^{r_\ell} \\ x_c(0) \in \mathcal{X}_i^h(0) \\ x_\ell(0) = x_{\ell i} \end{cases} \end{cases}$$

1.7.2. For each optimization in 1.7.1., if $\delta(t) = \delta_j \neq \delta_i$,
1.7.2.1. $n \leftarrow \max\{k : \ P_j^k$ is on STACK$\} + 1$.
1.7.2.2. If $\mathcal{X}_j^n(0) \not\subseteq \mathcal{X}_j^k(0)$, $\forall P_j(k)$ on STACK, push $P_j^n = \{j, \ n, \ \delta_j, \ T_j^n = 0, \ \mathcal{X}_j^n(0) \subseteq \mathcal{C}_j\}$ on STACK
1.7.2.3. Recompute 1.7.1. with the additional constraint $\delta(t) = \delta_i$
1.8. If $M_j(t) \leq M_j(t-1)$ or the problems in 1.7.1. or 1.7.2.3. are infeasible, fathom P_i^h and go to 1.1.
1.9. Goto 1.4.
2. Stop.

At step (1.7.2.2.), one must define the new set $\mathcal{X}_j^n(0)$. According to Algorithm 1, one should define $\mathcal{X}_j^n(0) = \text{Reach}(t, \mathcal{X}_i^h(0)) \bigcap \mathcal{C}_j$, where the reach set at time t $\text{Reach}(t, \mathcal{X}_i^h(0))$ is implicitly defined by Eqs. (4)+(1c). This definition of $\mathcal{X}_j^n(0)$, although exact, has the disadvantage that the number of constraints defining $\mathcal{X}_j^n(0)$ might keep growing during the execution of Algorithm 2. In this paper we propose two alternatives, leading to inner and outer approximations of $\mathcal{X}_j^n(0)$ respectively. The inner approximation $\mathcal{X}_j^n(0) = \{x_c(t)\}$ ($x_c(t)$ corresponds to the point marked as '$\star$' in Fig. 1). The second consists of approximating $\mathcal{X}_j^n(0)$ with a hyper-rectangle (dashed rectangle in Fig. 1), which is computed by covering the points obtained as in step (1.7.2.3.) by letting $\delta(t) = \delta_j$. Other approximations are possible, and will be investigated in the future. For instance, a better inner approximation consists of taking the convex hull of these points. Ellipsoidal approximations seem to be not appropriate, as they would result in quadratic constraints in the optimization problems. Parallelotopic or higher order polyhedra can be better approximations. Another technique can consist in approximating recursively the reach sets during the exploration, so that $\text{Reach}(t, \mathcal{X}(0))$ always has a number of faces which is less than a specified limit. Finally, inner and outer approximations can be run in parallel, by increasing their complexity until lower and upper bounds to the verification problem converge within a desired threshold. These ideas will be investigated in future research.

At step (1.3.1.1.) the algorithm checks the stronger condition $\mathcal{X}_i^h(0) \subseteq \mathcal{X}_i^k(t)$ for some $t \leq T_i^k$, instead of $\mathcal{X}_i^h(0) \subseteq \bigcup_{i=0}^{T_i^k} \mathcal{X}_i^k(t)$. Because of convexity of the sets $\mathcal{X}_i^k(t)$, the first condition is easier to test, as only inclusion of the vertices

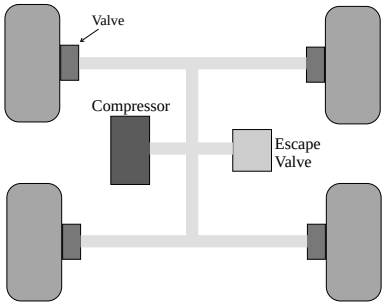


Fig. 2. Suspension system.

of $\mathcal{X}_i^h(0)$ must be checked. Note that although the results of the verification algorithm are not affected, the number of explored regions might increase.

4.1 Complexity of Algorithm 2

The optimization problems have the following complexity: (1.2) solves $2n_c$ LPs; (1.3.1.1.) one feasibility problem over linear constraints (LP); (1.4.) p_c LPs; (1.5.) m feasibility tests over linear constraints (LP), where m=number of rows in matrix K_1, i.e. number of inequalities defining the safe set $\mathcal{X}_s$; (1.7.1.) $2n_c$ MILPs with r_ℓ integer variables; (1.7.2.2) depends on the complexity of the sets $\mathcal{X}_j^k(0)$, for instance requires very little computation when $\mathcal{X}_j^k(0)$ are hyper-rectangles; (1.7.2.3.) $\leq 2n_c$ LPs. Note that compared to the simple approach described in Sect 3.1 where the number of integer variables involved in the optimization grows with time, in each optimization at step (1.7.1.) the number of integer variables remains equal to r_ℓ.

Although the number of continuous variables adds only minor computational complexity when compared to the number of integers, Algorithm 2 solves linear problems with a number of variables proportional to T_i^k. Our experience is that the problems are usually fathomed after a small number of time steps, i.e. T_i^k do not increase much. However, one might get large T_i^k when fast sampling is applied to slow dynamics, for instance, because of the presence of different time constants in the systems.

5 Verification of an Automotive Electronic Height Control System

In this section we describe an application of Algorithm 2 to the case study proposed first in [13], and reconsidered in [7] and [8]. The aim is to verify that an automotive control system satisfies certain driving comfort requirements.

5.1 Description of the System

The chassis level of the car is controlled by a pneumatic suspension system. The level is raised by pumping air into the system, and lowered by opening an escape valve. The configuration can be seen in Fig. 2. For the sake of simplicity, as in [13, 7], we consider an abstract model including only one wheel. The suspension system is commanded by a logic controller, whose behavior is represented in Fig. 3. In short, the controller switches the compressor on when the level of the chassis is below a certain outer tolerance OTl, off when it reaches again an inner tolerance ITl, while it opens the valve when the level is above OTh, and closes it again when the level decreases below ITh. Because of high-frequency disturbances due to irregularities of the surface of the road, the controller switches based on a filtered version $f(t) = \frac{1}{1+as}h(t)$ of the measured level h of the chassis. The filter is reset to $f = 0$ each time f returns within the inner range [ITl, ITh]. The compressor can lift the chassis at a rate $cp(t) \in [cp_{\min}\ cp_{\max}]$, and the escape valve can lower it at a rate $ev(t) \in [ev_{\min}\ ev_{\max}]$. These parameters are reported in Table 2. We model this uncertainty as unmeasured disturbances, by letting $cp(t) = \overline{cp} + \Delta cp(t)$, $ev(t) = \overline{ev} + \Delta ev(t)$, where $\overline{cp} = \frac{cp_{\max}+cp_{\min}}{2}$, $\overline{ev} = \frac{ev_{\max}+ev_{\min}}{2}$, and $\Delta cp(t)$, $\Delta ev(t)$ range within $[\frac{cp_{\min}-cp_{\max}}{2}, \frac{cp_{\max}-cp_{\min}}{2}]$ and $[\frac{ev_{\min}-ev_{\max}}{2}, \frac{ev_{\max}-ev_{\min}}{2}]$ respectively.

Symbol	OTh	OTl	ITh	ITl	$1/a$	T_s	$cp_{\min}$	$cp_{\max}$	$ev_{\min}$	$ev_{\max}$	$d_{\min}$	$d_{\max}$	f_{sp}
Value	20	-40	16	-6	2	1	1	2	-2	-1	-1	1	0
Unit	mm	mm	mm	mm	s	s	mm s^{-1}	mm s^{-1}	mm s^{-1}	mm s^{-1}	mm s^{-1}	mm s^{-1}	mm

Table 2. Model parameters

6 Modeling the Automotive Hybrid System in MLD Form

The Electronic Height Controller is represented by the automaton depicted in Fig. 3. We introduce auxiliary binary variables in order to translate the automaton into the MLD form (1). We will use a shortened notation by writing δ instead of $[\delta = 1]$, and $\bar{\delta}$ instead of $[\delta = 0]$. Define

$$\delta_1 \leftrightarrow [f(t) \leq \text{ITh}]\,,\ \ \delta_2 \leftrightarrow [f(t) \leq \text{ITl}]$$
$$\delta_3 \leftrightarrow [f(t) \geq \text{OTh}]\,, \delta_4 \leftrightarrow [f(t) \geq \text{OTl}]$$

As OTl<ITl<ITh<OTh, it is easy to see that

$$\delta_1 \rightarrow \bar{\delta}_3; \quad \delta_2 \rightarrow \delta_1,\ \bar{\delta}_3; \quad \bar{\delta}_4 \rightarrow \bar{\delta}_3,\ \delta_2,\ \delta_1$$

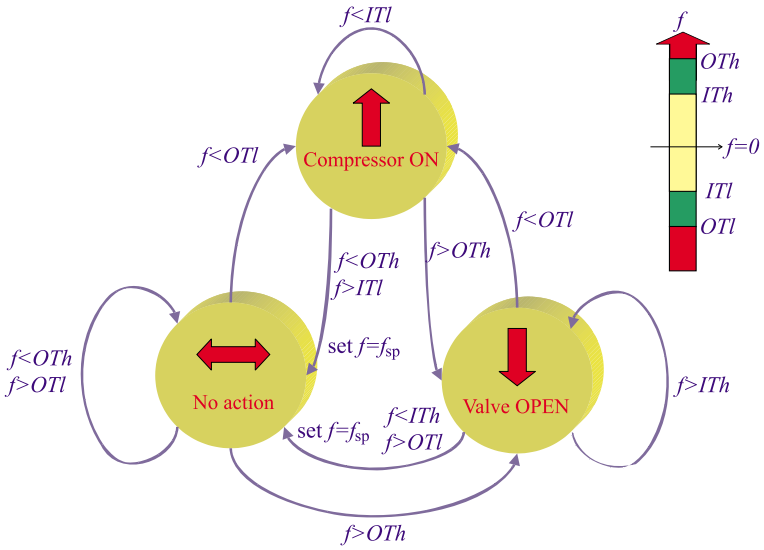


Fig. 3. Hybrid automaton for the controller.

The three states of the automaton in Fig. 3 are represented by a two-dimensional logic state

$$x_\ell = \begin{cases} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \text{if compressor OFF, valve CLOSED} \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \text{if compressor ON, valve CLOSED} \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \text{if compressor OFF, valve OPEN} \end{cases} \tag{5}$$

By using Karnough map techniques, (5) can be rewritten as

$$x_{\ell 1}(t+1) = \bar{x}_{\ell 1}(t)\delta_3(t) + x_{\ell 1}(t)\bar{\delta}_1(t) \tag{6}$$

$$x_{\ell 2}(t+1) = x_{\ell 2}(t)\delta_2(t) + \bar{x}_{\ell 1}(t)\bar{\delta}_4(t) \tag{7}$$

(note that typically (6)–(7) are available from the team which designed the logic controller). By defining $\delta_5 = \bar{x}_{\ell 1}\delta_3$, $\delta_6 = x_{\ell 1}\bar{\delta}_1$, $\delta_7 = x_{\ell 2}\delta_2$, $\delta_8 = \bar{x}_{\ell 2}\bar{\delta}_4$, $\delta_9 = \delta_5 \vee \delta_6$, $\delta_{10} = \delta_7 \vee \delta_8$, Eqs. (6)–(7) are equivalent to $x_{\ell 1}(t+1) = \delta_9(t)$, $x_{\ell 2}(t+1) = \delta_{10}(t)$. As the state $\left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$ does not exist, the additional constraints $x_{\ell 1} + x_{\ell 2} \leq 1$, $\delta_9 + \delta_{10} \leq 1$ are included[3].

The input $w(t)$ of the system is redefined as

$$w(t) = \begin{cases} \overline{cp} + \Delta cp(t) \text{ if } x_\ell(t) = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right] \\ \overline{ev} + \Delta ev(t) \text{ if } x_\ell(t) = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] \\ 0 \text{ if } x_\ell(t) = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right] \end{cases} \tag{8}$$

By letting $\delta_{11} = \bar{x}_{\ell 1}x_{\ell 2}$, $\delta_{12} = x_{\ell 1}\bar{x}_{\ell 2}$, with $\delta_{11} + \delta_{12} \leq 1$, and $z_1 = cp\ \delta_{11}$, $z_2 = ev\ \delta_{12}$, one gets $w(t) = z_1(t) + z_2(t)$. The continuous dynamics of the car

[3] Contrary to optimization over continuous variables, in mixed-integer programming constraints involving integer variables can help the solver significantly.

and the filter are sampled by exact discretization (by introduction of zero-order holders), namely

$$f(t+1) = e^{-aT_s} f(t) + (1 - e^{-aT_s}) h(t) \tag{9}$$
$$h(t+1) = h(t) + T_s[d(t) + w(t)] \tag{10}$$

As the filter is reset to a set point value $f_{\rm sp}$ during the transitions $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right] \to \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ and $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \to \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$, i.e.

$$\left[[x_\ell(t) = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \wedge x_\ell(t+1) = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]] \vee [x_\ell(t) = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \wedge x_\ell(t+1) = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]]\right] \to [f(t) = f_{\rm sp}]$$

we introduce the variables $\delta_{13} = \overline{(\bar{\delta}_9 \bar{\delta}_{10} \delta_{14})}$, where $\delta_{14} = x_{\ell 1} \vee x_{\ell 2}$, and modify (9) in the form

$$f(t) = z_3(t),\ z_3(t) = [e^{-aT_s} f(t) + (1 - e^{-aT_s}) h(t)] \delta_{13}(t) + f_{\rm sp}[1 - \delta_{13}(t)]$$

In summary, the system has $x_c = [f, h]'$, $x_\ell = [x_{\ell 1}, x_{\ell 2}]'$, $w = [\Delta cp, \Delta ev, d]'$, $\delta = [\delta_1, \ldots, \delta_{14}]'$, and $z = [z_1, z_2, z_3]'$.

Matrix	A	B_1	B_2	B_3	E_1	E_2	E_3	E_4	E_5
Sparsity (%)	93.75	92.67	96.43	75.00	97.74	89.46	93.22	85.59	47.45

Table 3. Sparsity of MLD matrices

6.1 MLD Translation & HYSDEL List

The system described above is translated into the MLD form (1) by using the language HYSDEL (HYbrid System DEscription Language) currently developed at the Automatic Control Lab, ETH Zürich. The description of the system in HYSDEL is reported in Table 4. The HYSDEL compiler automatically generates the matrices of the system. The sparsity is reported in Table 3. The number of constraints in (1c) is 59.

6.2 Numerical Results

Algorithm 2 has been implemented in Matlab using rectangular approximations of new regions to explore, and provides a maximum range $-44.54149 \leq h(t) \leq 25.00000$. The rectangular approximations $\mathcal{X}_i^h(0)$ in the (f, h) plane are shown in Fig. 4. The algorithm uses interpreted m-code and terminates in 761 s on a PC Pentium II 300 MHz with 96 Mb RAM. In [7], where this verification problem is solved analytically, the authors report the exact range $-43 \leq h(t) \leq 23$. In [13], the authors use HyTech [2] for symbolic verification, and obtain the range $-47 \leq h(t) \leq 27$. These bounds are slightly conservative because, in order

```
% Description of variables and constants

state f,h,xl1,xl2;
input d,dc,dev;

const OTh, OTl, ITh, ITl;
const M1,M2,M3,M4,m1,m2,m3,m4;
const e;
const Ts,cbar,evbar,eats,cmax,cmin,evmax,evmin;

% Variable types

real f,h,z1,z2,z3,d,dc,dev;
logic d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14;


% Relations

d1 = {f-ITh <= 0, M1, m1, e};
d2 = {f-ITl <= 0, M2, m2, e};
d3 = {f-OTh >= 0, M3, m3, e};
d4 = {f-OTl >= 0, M4, m4, e};

d5 = ~xl1 & d3;   % Should be accepted also: d5=(1-xl1)&d3, d5=(1-xl1)*d3
d6 = xl1 & ~d1;
d7 = xl2 & d2;
d8 = ~xl2 & ~d4;
d9 = d5 | d6;
d10 = d7 | d8;
d11 = ~xl1 & xl2;
d12 = xl1 & ~xl2;
d13 = ~(~d9 & ~d10 & d14);
d14 = xl1 | xl2;

z1 = d11 * (cbar + dc) {cmax,cmin,e};
z2 = d12 * (evbar + dev) {evmax,evmin,e};
z3 = (eats * f + (1 - eats) * h)*d13 {10*OTh,10*OTl,e};

% Other constraints

must xl1 + xl2 <= 1;
must ~(d9 & d10);
must ~(d11 & d12);

% Update

update f = z3;
update h = h + Ts * (d + z1 + z2);
update xl1 = d9;
update xl2 = d10;
```

Table 4. HYSDEL description of the automotive active leveler.

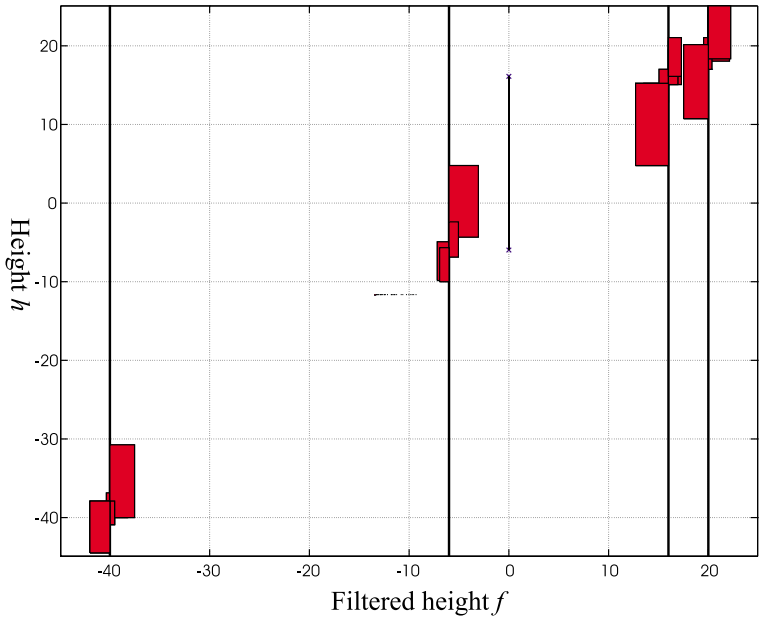
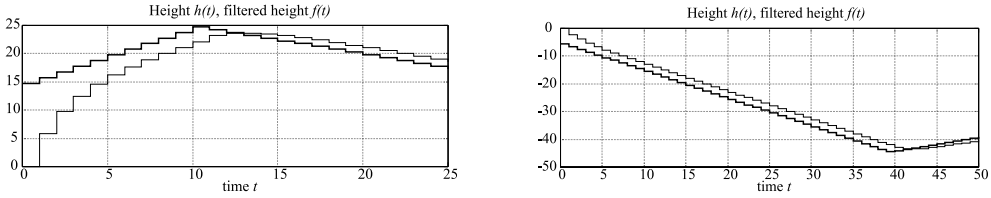


Fig. 4. Evolution of Algorithm 2 in the (f, h) plane using rectangular approximations of the new regions. The vertical thick lines represent the guard lines, where the logic conditions switch.

to fit the model used in HyTech, the dynamics is described as $k_1 \leq \dot{x} \leq k_2$. The authors report a computation time of 62 m on a Sun SparcStation 20 with 128 Mb RAM. In [8], the author reports computation times up to one day.

Although some conservativeness arises from the rectangular approximations, the algorithm presented in this paper provides a larger range mainly because of the discrete-time filter dynamics. In fact, one can easily check that the sequence of disturbances $d(t) \equiv 1$, $\Delta cp(t) \equiv 0$, $\Delta ev(t) \equiv 0$ leads the initial state $f(0) = 0$, $h(0) = 14.76537$ to $h(16) = 24.76537$ (as $f(8) = 19.99999$, the transition from the logic state $(0, 0)$ to $(1, 0)$ happens between $t = 9$ and $t = 10$. See Fig. 5(a)). Note that by using a point-wise inner approximation, one gets $h(t) \leq 24.35318$. Concerning the lower bound, by applying $d(t) \equiv -1$, $\Delta cp(t) \equiv 0$, $\Delta ev(t) \equiv 0$ from the initial state $f(0) = 0$, $h(0) = -5.54149$, one reaches exactly the computed lower bound $h(39) = -44.54149$ (in fact for $f(37) = -39.99999$, the transition from the logic state $(0, 0)$ to $(0, 1)$ happens between $t = 38$ and $t = 39$. See Fig. 5(b)).

These different approaches to the solution of the verification problem have their benefits and disadvantages. The method proposed in [2] uses symbolic computation, can handle a wide class of verification problems, but requires approximation of the dynamics and is computationally expensive. In [8], there is no approximation of the dynamics, but the author uses parallelotopic approximations of the reach sets in order to compute the solution of the verification



(a) $d(t) \equiv 1$. (b) $d(t) \equiv -1$

Fig. 5. Worst-case simulation with step disturbances $d(t)$ (thin line: $f(t)$; thick line: $h(t)$).

problem. The approach suggested in [7] is exact, but seems to be tailored to the particular example. The algorithm proposed in this paper promises to be computationally affordable, can handle a wider class of hybrid models, and allows the solution of verification problems that can be recast in an optimization framework.

7 Conclusions

In this paper we have presented a novel approach to the verification of hybrid systems. It is based on linear/mixed-integer linear optimization and relies on the modeling formalism introduced in [5]. Computational feasibility of the approach has been shown in a non-trivial case study. Future research will be devoted to examine different inner/outer approximation techniques, iterative approximation of the reach sets, and improving the efficiency of the computer codes used to test the proposed algorithms.

Acknowledgments

The authors thank Prof. Sanjoy Mitter, Dr. Nicola Elia, and Fabio Torrisi for fruitful discussions.

References

1. R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In A.P. Ravn R.L. Grossman, A. Nerode and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture notes in computer Science*, pages 209–229. Springer Verlag, 1993.
2. R. Alur, T.A. Henzinger, and P.-H. Ho: Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22:181–201, 1996.

3. A. Asarin, O. Maler, A. Pnueli: On the Analysis of Dynamical Systems having Piecewise-Constant Derivatives. *Theoretical Computer Science*, 138:35–65, 1995.
4. A. Bemporad, D. Mignone, and M. Morari: Moving horizon estimation for hybrid systems and fault detection. *Submitted American Control Conference*, 1999.
5. A. Bemporad and M. Morari: Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3), March 1999.
6. T. M. Cavalier, P. M. Pardalos, and A. L. Soyster: Modeling and integer programming techniques applied to propositional calculus. *Computers Opns Res.*, 17(6):561–570, 1990.
7. N. Elia and B. Brandin: Verification of an automotive active leveler. Technical report, Dept. of Electrical Engineering and Computer Science, M.I.T., 1999.
8. A. Fehnker: Automotive control revised - linear inequalities as approximation of reachable sets. In *Hybrid Systems: Computation and Control*, volume 1386 of *Lecture notes in Computer Science*, pages 110–125. Springer Verlag, 1998.
9. R. Fletcher and S. Leyffer: A mixed integer quadratic programming package. Technical report, University of Dundee, Dept. of Mathematics, Scotland, U.K., 1994.
10. R. Fletcher and S. Leyffer: Numerical experience with lower bounds for miqp branch-and-bound. Technical report, University of Dundee, Dept. of Mathematics, Scotland, U.K., 1995. submitted to SIAM Journal on Optimization, `http://www.mcs.dundee.ac.uk:8080/ sleyffer/miqp_art.ps.Z`.
11. Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine: Integration graphs: a class of decidable hybrid systems. In A.P. Ravn R.L. Grossman, A. Nerode and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture notes in computer Science*, pages 179–208. Springer Verlag, 1993.
12. R. Raman and I. E. Grossmann: Relation between milp modeling and logical inference for chemical process synthesis. *Computers Chem. Engng.*, 15(2):73–84, 1991.
13. T. Stauner, Olaf Müller, and M. Fuchs: Using HYTECH to verify an automotive control system. In O. Maler, editor, *Hybrid and Real-Time Systems*, volume 1201 of *Lecture notes in Computer Science*, pages 139–153. Springer Verlag, 1997.
14. M.L. Tyler and M. Morari: Propositional logic in control and monitoring problems. *Automatica*, in print, 1999.
15. H.P. Williams: *Model Building in Mathematical Programming.* John Wiley & Sons, Third Edition, 1993.