Verifying Industrial Hybrid Systems with MathSAT*

Gilles Audemard[1], Marco Bozzano[2], Alessandro Cimatti[2], and Roberto Sebastiani[2,3]

[1] Centre de Recherche en Informatique de Lens
IUT de Lens, Rue de l'université, SP16, F 62307 Lens Cedex
`audemard@iut-lens.univ-artois.fr`
[2] ITC-IRST, Via Sommarive 18, 38050 Povo, Trento, Italy
`{bozzano,cimatti}@itc.it`
[3] DIT, Università di Trento, Via Sommarive 14, 38050 Povo, Trento, Italy
`rseba@dit.unitn.it`

Abstract. Industrial systems of practical relevance can be often characterized in terms of discrete control variables and real-valued physical variables, and can therefore be modeled as hybrid automata. Unfortunately, continuity of the physical behaviour over time, or triangular constraints, must often be assumed, which yield an undecidable class of hybrid automata.
In this paper, we propose a technique for bounded reachability of linear hybrid automata, based on the reduction of a bounded reachability problem to a MathSAT problem, i.e. satisfiability of a boolean combination of propositional variables and mathematical constraints. The MathSAT solver can be used to check the existence (or absence) of paths of bounded length.
The approach is very similar in spirit to SAT-based bounded model checking; furthermore, the ability to reason directly about real variables gives computational leverage over discretization-based methods. Despite the undecidability of the general problem, the proposed method is able to provide valuable information on large designs of practical relevance.

1 Introduction

Many systems and plants of industrial relevance (e.g., engines, turbines) are defined in terms of discrete control variables and physical real-valued variables (e.g., speed, pressure), and can be naturally modeled as hybrid automata: depending on a discrete state (e.g., "nominal", "increasing"), different equations describe

* This work has been sponsored by the CALCULEMUS! IHP-RTN EC project, contract code HPRN-CT-2000-00102, and has thus benefited of the financial contribution of the Commission through the IHP programme. It has also been partly supported by ESACS, an European sponsored project, contract no. G4RD-CT-2000-00361. The fourth author is also sponsored by a MIUR COFIN02 project, code 2002097822_003.

the behaviour of the physical variable (e.g., speed). Frequently, the dynamics of physical variables is continuous: i.e., transitions from a discrete state to another should not necessarily yield a discontinuity in the physical dimension. For instance, in the transition from "increasing" to "decreasing", the velocity should not change its value (but only its derivative). Furthermore, the evolution can depend on the comparison between the values of physical variables. Unfortunately, either imposing continuity or allowing for comparisons between variables (also known as triangular constraints) result in a class of hybrid automata where even reachability is undecidable [HKPV98]. Yet, it is very important to be able to develop tools that allow to formally validate such designs, that often implement critical functionalities (e.g., control systems for avionics).

In this paper, we address the problem of verifying hybrid automata with continuous variables and triangular constraints. We propose a formal verification method for bounded reachability. The approach is based on the encoding of a bounded reachability problem into a MathSAT problem, i.e. the problem of checking the satisfiability of a boolean combination of propositional variables and mathematical constraints over real variables. The approach is made practical by the use of the efficient MathSAT solver [ABC+02a], that extends and integrates state-of-the-art techniques for propositional satisfiability (SAT) with a set of mathematical reasoners. The approach presented in this paper is largely similar to bounded model checking [BCCZ99], and enhances the method presented at [ACKS02], limited to timed systems, to dealing with real variables with arbitrary linear dynamics.

The proposed technique is clearly incomplete, and currently limited to the case of linear dynamics. Despite these facts, however, it allows us to represent and to analyze interesting systems from real-world applications [BVÅ+03,BCC+03], providing useful information, especially oriented to debugging and goal-directed simulation. An experimental analysis shows that our techniques is competitive with state of the art verification tools such as HyTech, and with methods based on the discretization of real variables.

Outline of the paper The rest of the paper is structured as follows. In Section 2 we illustrate a motivating example for our approach; in Section 3 we give a short and informal introduction to our model of hybrid systems; in Section 4 we give a brief overview of SAT-based bounded model checking and we discuss in more detail our encoding of hybrid systems into MathSAT; in Section 5 we discuss some experiments, and finally in Section 6 and 7 we discuss related work and draw some conclusions.

2 A Motivating Example: the Secondary Power System

Throughout the paper, we use a running example to motivate and illustrate the main concepts we present. Specifically, we discuss the modeling and analysis of a real-world safety-critical system, namely the Secondary Power System (SPS). It is an industrial case study which has been and is being investigated within ESACS (Enhanced Safety Analysis for Complex Systems), a

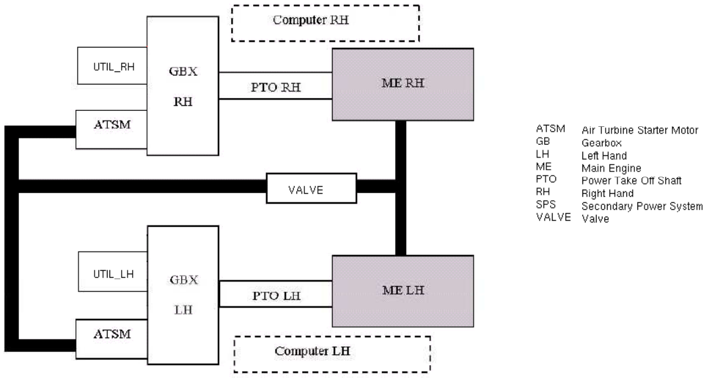

Fig. 1. SPS schematic view

European-Union-sponsored project in the avionics sector, whose goal is to define a methodology to improve the safety analysis practice for complex systems development [BVÅ+03,BCC+03].

The SPS drives the hydraulic and electrical utilities of an aircraft. It is an example of safety-critical system with embedded hardware and software components. The hardware subsystems comprise (electro)-mechanical components (e.g., control valves, relays, shafts, gearboxes, freewheels) and electronic transducers (e.g., speed and pressure sensors), whereas the software component is given by embedded controllers (SPS computers).

The SPS drives the utilities of both the left and right hand side of the aircraft. To ensure the basic safety requirement, i.e. *no single failures shall cause the total loss of the SPS utilities*, the architecture of the system includes two basic redundancies: there are two independent and perfectly symmetric lines, whose purpose is to drive the left and the right hand side utilities, respectively; for each side, the mechanical drive of the relevant utilities (normal mode) is redounded by a pneumatic drive (*cross-bleed mode*) in case of failure of one of the components in the mechanical line.

Figure 1 shows a simplified schematic view of the SPS. The SPS normal operation consists in transmitting the mechanical power from the engines to the relevant hydraulic and electrical generators. Specifically, the mechanical power of the main engine (ME) is transmitted via the Power Take Off Shaft (PTO) to a gearbox (GBX) which feeds the utilities. A component may fail due to abnormal operational conditions or ruptures. As an example, *flameout* and *grippage* are two possible failure modes of the main engine. To ensure safety of in-flight operation, in case of an engine failure the SPS computers automatically initiate a *cross-bleed* procedure consisting in driving the hydraulic and electrical generators by means of an air turbine motor (ATSM), using bled air from a valve (VALVE), which is in turn fed by the mechanical power coming from the opposite engine. Correct

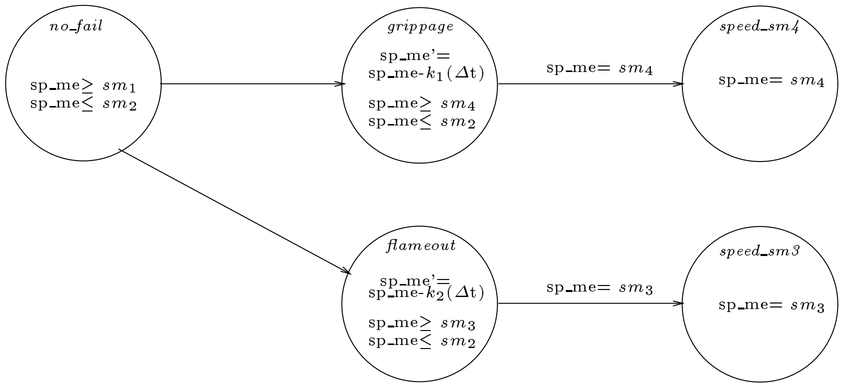

Fig. 2. SPS: main engine automaton (ME)

functioning of the cross-bleed procedure is an example of one safety requirement of the SPS. Some experimental results about this will be presented in Section 5.

3 Modeling Hybrid Automata

In this section we briefly present and exemplify our model of hybrid systems. The model is inspired by the *linear* and *rectangular hybrid automata* models presented in [Hen96,HHWT97]. Informally, a hybrid system can be seen as the parallel composition of a collection of hybrid automata, which can communicate either by explicit synchronization on some *channel*, or implicitly by means of *shared variables*. Each automaton models both *discrete* events (e.g., failure of a component) and continuous activities of analog variables (e.g., time, component speed). At any given instant of time, the state of a hybrid automaton is defined by a control location (discrete state) and the values of all the analog variables (continuous state). The state can change either because of an *instantaneous* discrete transition, which changes the control location and may also affect the values of the analog variables (e.g., re-initialization is possible) or because of a *time elapse* (continuous) transition, which changes only the values of the analog variables according to some specified law. Hybrid systems can be seen as an extension of the timed systems model of [ACKS02], in which the only analog variables are clocks. In the following, by *elementary linear expression* we mean an equality and/or (non-strict) inequality over *linear terms* (i.e., linear combinations of real-valued variables with rational coefficients).

Figure 2 and 3 depict two examples of hybrid automata, modeling the *main engine* (ME) and the valve (VALVE) components of the SPS. A hybrid automaton consists of the following components:

Locations A finite set of locations, encoding the discrete states of the hybrid automaton. The automaton in Figure 2 has five locations, drawn as circles,

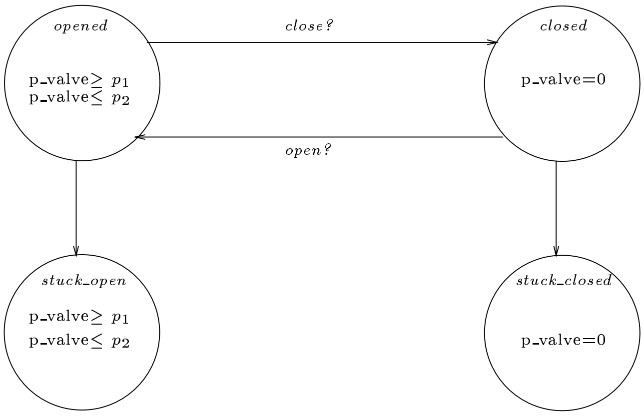

Fig. 3. SPS: valve automaton (VALVE)

which model the discrete state of the ME of the SPS. Location no_fail models the default behaviour of the engine; locations *grippage* and $flameout$ model two different faulty states; locations *speed_sm*4 and *speed_sm*3 model states in which the speed of the engine has the constant value sm_4 and sm_3.

Analog Variables A finite vector of real-valued data variables $(w_1, \ldots, w_n)$. The *sp_me* variable in Figure 2 encodes the speed of the ME. *Clock* variables of [ACKS02] may be seen as a particular case of real-valued variables. Primed variables (e.g., *sp_me*′) are used to denote the value of real-valued variables after execution of a transition.

Initial and Invariant Conditions Every location of a hybrid automaton may be declared as initial (meaning that it is a legal initial state of the system). Every location may be equipped with *invariants* on the real-valued variables, expressed by means of a set of elementary linear expressions $\{\psi_1, ..., \psi_h\}$ over the variables $w_1, \ldots, w_n$. Location no_fail is the initial location of the ME automaton (Figure 2), and is equipped with an invariant enforcing the *sp_me* variable to stay between the constant values sm_1 and sm_2. The invariant in location *speed_sm*3 forces *sp_me* to assume constantly the value sm_3.

Channels A finite set of channels is used for discrete communication between automata. A channel c may be used as an *input* (notation c?) or an *output* (notation c!) channel for synchronizing different automata. For instance, the pressure valve automaton of Figure 3 uses two different input channels called *open*? and *close*?. The intended semantics is that the pressure valve automaton awaits for incoming commands (requesting either opening of closing of the valve) coming from the relevant SPS computer controller.

Transitions A finite set of discrete transitions encodes the *discrete* evolution of the automaton. Each transition (also called *switch*) has a *source* and *target* location, and may be equipped with a set $\{\gamma_1, ..., \gamma_k\}$ of *guards* (pre-conditions) and a set $\{\theta_1, ..., \theta_m\}$ of *jump conditions* (post-conditions) on the real-valued variables. A guard is an elementary linear expression over $w_1, \ldots, w_n$; a jump condition is an elementary linear expression over $w_1, \ldots, w_n, w'_1, \ldots, w'_n$. In Figure 2, the transition from *flameout* to *speed_sm*3 has a guard $sp\_me = sm_3$ and no jump condition. By convention, the absence of jump conditions on a transition forces real-valued variables to preserve their value (e.g., in the previous example $sp\_me' = sp\_me$ implicitly holds). Transitions may be equipped with one or more optional labels denoting the channels on which the automaton must synchronize. For instance, two transitions in Figure 3 are labeled with the input channels *open*? and *close*?

Variable Dynamics Variable dynamics describe how the real-valued variables change in presence of a time elapse transition, and are expressed, for each location, as a set $\{\Psi_1, \ldots, \Psi_k\}$ of elementary linear expressions over $w_1, \ldots, w_n$, $w'_1, \ldots, w'_n$, As an example, in Figure 2 the *sp_me* variable in location *grippage* varies according to the law $sp\_me' = sp\_me - k_1(\Delta t)$ (where k_1 is a constant), that is, the speed decreases linearly (proportionally to the time delay) with first derivative equal to k_1. The expression Δt, encoding the time delay, will be explained in Section 4.2.

The hybrid automata presented here do not fall into the *rectangular* automata class described in [Hen96], since re-initialization of variables is not enforced, and triangular constraints are possible. As a consequence, even the problem of reachability for this class of automata is undecidable [HKPV98].

4 Bounded Model Checking for Hybrid Systems

In this section we give a very short overview of SAT based model checking, and we discuss the encoding of our model of hybrid systems, informally described in Section 3, into MathSAT.

4.1 SAT Based Bounded Model Checking

Bounded Model Checking (BMC) is a recent approach to symbolic model checking [BCCZ99]. Given a Kripke structure M, and an LTL formula f, the idea is to check whether f is true in M by looking for a counterexample (i.e., a witness to the violation of f) that can be presented within a bound of k steps. Given k, the problem is reduced to the satisfiability of a propositional formula $[[M, \neg f]]_k$. For instance, for a property of the form $f := \mathbf{G}p(\underline{s})$, where $p(\underline{s})$ is a boolean formula in the boolean variables $\underline{s}$, then

$$[[M, \neg f]]_k = \mathcal{I}(\underline{s}^{(0)}) \wedge \bigwedge_{i=0}^{k} \mathcal{C}(\underline{s}^{(i)}) \wedge \bigwedge_{i=0}^{k-1} \mathcal{R}(\underline{s}^{(i)}, \underline{s}^{(i+1)}) \wedge \bigvee_{i=0}^{k} \neg p(\underline{s}^{(i)}),$$

where $\mathcal{I}(\underline{s}^{(0)})$ is a representation of the initial conditions, $\mathcal{C}(\underline{s}^{(i)})$ is a representation of the invariant conditions at step i, and $\mathcal{R}(\underline{s}^{(i)}, \underline{s}^{(i+1)})$ is a representation of the transition relation from step i to step $i+1$. If $[[M, \neg f]]_k$ is satisfiable, the propositional model provides a counterexaple of k steps to f. If $[[M, \neg f]]_k$ is unsatisfiable, then nothing can be said about the existence of counterexamples to $M \models f$ with higher bound. The typical technique is to generate and solve $[[M, \neg f]]_k$ for increasing values of k, until either a counter-example is found, or a given time-out is reached.

BMC is being increasingly accepted as practical technique, effective in particular in the process of falsification, i.e. bug finding. The technique relies on the use of efficient SAT solvers (e.g., based on DPLL procedures [DLL62]) for checking the propositional satisfiability of $[[M, \neg f]]_k$. As shown in [CFG+01], BMC avoids the blow-up in memory that can occur with model checking based on Binary Decision Diagrams, and is therefore able to tackle much larger circuits.

4.2 The encoding

Our approach to the verification of hybrid automata is a generalization of BMC for timed systems, as proposed in [ACKS02]. The approach reduces a BMC problem for timed systems to the problem of deciding the satisfiability of math-formulas, i.e. boolean combinations of boolean variables and linear (in)equalities over real variables, representing absolute time and clocks. The resulting math-formulas are tackled with MathSAT [ABC+02b,ABC+02a], a solver combining an efficient DPLL procedure with mathematical constraint solvers of increasing deductive power.

In the encoding for timed automata, boolean variables are used to encode the discrete part of the system, while linear constraints on real variables encode the timed part. In particular, each location l is represented by a bitwise encoding $\underline{l}$, so that $\underline{l_i}$ holds if and only if the system is in the location l_i; each synchronization event (channel, shared variable) is represented by a corresponding boolean variables; each switch is represented by a single boolean variable (say, T) which holds if and only if the system executes the corresponding switch; a boolean variable T_δ, representing a continuous transition, holds if and only if time elapses by some $\delta > 0$; finally, for each process P_i, we introduce a boolean variable T^i_{null}, that holds if and only if process P_i does nothing. In order to deal with time, we introduce a real valued variable t representing absolute time, and, for each clock x, a real valued variable ox representing the difference with respect to absolute time. All mathematical constraints required in the encoding are in the form $v_1 - v_2 \bowtie c$, $\bowtie \in \{\leq, \geq, =, >, <\}$ v_1 and v_2 being real variables representing either absolute time or clock values. The reader can refer to [ACKS02] for details.

We tackle the case of hybrid automata by considering that it is an extension of the case of timed automata. The encoding for the timed case is extended by introducing a set of real variables ω_i's, representing physical entities. To simplify the notation, in the following we write: "Δt" for the difference $t' - t$ between absolute time in the next and in the current state; "$\Delta\omega$" for "$\omega' - \omega$", so that, e.g., we write "$c \cdot \Delta t ...$" for "$c \cdot t' - c \cdot t ...$"; "$\Delta\omega = 0$" for "$\omega' = \omega$", "$\Delta\omega \leq ...$" for "$\omega' \leq \omega + ...$". We also write "$(w \in [t_1, t_2])$" for "$(w \geq t_1) \wedge (w \leq t_2)$", where t_1 and t_2 are linear terms. If ψ is a formula, ψ' denotes the formula obtained by substituting in ψ each propositional variable p_j with p'_j, and each real variable v_i with v'_i.

Initial conditions $\mathcal{I}(\underline{s}^{(0)})$. At step 0, in an initial location l, ω can either:

- be set to a given initial value c_0. If so, we represent this fact by the axiom:

$$\underline{l}^{(0)} \rightarrow (\omega^{(0)} = c_0); \tag{1}$$

- be set nondeterministically to an initial value within a closed interval $[a_0, b_0]$, $a_0, b_0 \in [-\infty, \infty]$. [1] If so, we represent this fact by the axiom:

$$\underline{l}^{(0)} \rightarrow (\omega^{(0)} \in [a_0, b_0]). \tag{2}$$

Invariant conditions $\mathcal{C}(\underline{s})$. For each location l equipped with the set $\{\psi_1, ..., \psi_h\}$ of invariants on real valued variables, we include the axiom

$$\underline{l} \rightarrow \bigwedge_j \psi_j. \tag{3}$$

Transition relation $\mathcal{R}(\underline{s}, \underline{s}')$. For each switch T equipped with a set $\{\gamma_1, ..., \gamma_k\}$ of guards and with a set $\{\theta_1, ..., \theta_m\}$ of jump conditions on the real valued variables ω_i's and t, we include the axioms

$$T \rightarrow \bigwedge_j \gamma_j, \tag{4}$$

$$T \rightarrow \bigwedge_j \theta'_j \tag{5}$$

respectively. For each physical variable ω that is not interested by a jump condition of switch T, and must therefore keep its value, we add the axiom:

$$T \rightarrow (\Delta\omega = 0). \tag{6}$$

When process i does nothing, in correspondence of T^i_{null}, each physical variable ω maintains its value:

$$T_{null} \rightarrow (\Delta\omega = 0). \tag{7}$$

When time elapses in a location l, physical variables ω_i evolve according to the set of variable dynamics $\{\Psi_1, \ldots, \Psi_k\}$ associated with l. For each location, we add the axiom

$$(\underline{l} \wedge T_\delta) \rightarrow \bigwedge_i \Psi_i \tag{8}$$

Different forms of variable dynamics are possible:

[1] "$a_0 = -\infty$" and "$b_0 = -\infty$" mean that there is no lower bound and no upper bound for ω respectively.

– ω maintains its value under a dynamic of the form:

$$(\underline{l} \wedge T_\delta) \rightarrow (\Delta\omega = 0); \tag{9}$$

– ω may evolve deterministically according to a linear function:

$$(\underline{l} \wedge T_\delta) \rightarrow (\Delta\omega = c \cdot \Delta t) \tag{10}$$

c being a constant.

– ω may evolve nondeterministically within two linear functions:

$$\omega' \in [b_1\omega + c_1 \cdot \Delta t - a_1, b_2\omega + c_2 \cdot \Delta t + a_2], \tag{11}$$

$$a_1, a_2 \geq 0,\ b_1, b_2 \in \{0, 1\},\ c_1, c_2 \in (-\infty, \infty). \tag{12}$$

If $a_1 = a_2 = 0$, then (11) encodes a triangular constraint. If $b_1 = b_2 = 0$ and $c_1 = c_2 = 0$, then (11) encodes a rectangular constraint.

– in the general case, the evolution of the variables can be nondeterministic within the space described by the linear inequalities $\{\Psi_1, \ldots, \Psi_k\}$, as in equation 8.

The encoding of properties basically follows the encoding in [ACKS02]. Our approach is bounded complete, in the following sense: if there exists a trace of length k, then the encoding of length k is satisfiable, and can be found by running MathSAT on it. The undecidability of the class of hybrid automata we are dealing with tells us that it is in general impossible to decide if a counterexample might be found with bigger k, or if the problem is unsolvable.

5 Experimental Evaluation

We evaluated the potential of the approach by tackling an example of hybrid systems of industrial relevance, i.e. the model of the SPS. The bounded reachability method described in Section 4 can be used both for model debugging (i.e., bug hunting) and for simulation of hybrid systems. In the following we provide some hints about the use of our methodology by showing some experimental results. For illustration purposes, we will discuss a simplified *one-sided* model of the SPS case study, including one instance of the ME, GBX, VALVE, ATSM, PTO and SPS computer components of Figure 1. Under this abstraction, the analogous components of the opposite side of the system are assumed to be correctly working. An example of property to be checked is given by (the negation of) the following formula:

```
(!GBX.loc_broken & !GBX.loc_grippage & !VALVE.loc_stuck_closed &
 !ATSM.loc_broken & !PTO.loc_fused)  U  GBX.sp_gbx <= sg1          (P1)
```

This is a typical safety property expressed via the LTL *until* operator. The intended semantics is whether there exists a path such that no failures of the GBX, VALVE, ATSM and PTO components happen along the path, and finally the speed of the *gearbox* (GBX component) drops below the constant value sg_1.

Time T_0 :
Locations : no_fail, gbx_pto_driven, $atsm\_idle$, sps_ok, $closed$
Analog Variables : $sp\_me = sm_2$, $sp\_gbx = sg_2$, $sp\_atsm = 0$
Discrete Trans : $me\_grippage$
Synchronizations : none
Time T_1 :
Locations : $grippage$, gbx_pto_driven, $atsm\_idle$, sps_ok, $closed$
Analog Variables : $sp\_me = sm_5$, $sp\_gbx = sg_3$, $sp\_atsm = 0$
Discrete Trans : $atsm\_inc\_a$, sps_inc_a, $valve\_open$
Synchronizations : SPS and ATSM on inc_a, SPS and VALVE on $open$
Time T_2 :
Locations : $grippage$, gbx_pto_driven, $atsm\_inc\_a$, sps_inc_a, $open$
Analog Variables : $sp\_me = sm_6$, $sp\_gbx = sg_4$, $sp\_atsm = sa_2$
Discrete Trans : $atsm\_inc\_a\_inc\_b$, $sps\_inc\_a\_inc\_b$
Synchronizations : SPS and ATSM on inc_b
Time T_3 :
Locations : $grippage$, gbx_pto_driven, $atsm\_inc\_b$, sps_inc_b, $open$
Analog Variables : $sp\_me = sm_7$, $sp\_gbx = sg_1$, $sp\_atsm = sa_3$

Fig. 4. An example of MathSAT trace

The negation of the above property can be seen as a *safety* property to be verified by the system (i.e., *in presence of failures due only to the main engine, the gearbox speed cannot drop below* sg_1). The rationale behind this property is that the *cross-bleed* procedure initiated by the SPS computer (see Section 2) is able to recover from an engine failure by using power coming from the opposite engine (which is assumed to be working in this one-sided model).

The property may or may not hold depending on the value chosen for the constant sg_1. In particular, if the value chosen for sg_1 exceeds a given threshold, the property is falsified by MathSAT (this means that the cross-bleed procedure is not able to prevent the gearbox speed to drop below that particular value). In this case, MathSAT generates an output trace showing an execution of the system which leads to the violation. The trace includes information on the discrete transitions and the time elapse transitions taken by the automata, the exact time delays and time points at which the transitions take place, and the synchronization channels between different automata. If the value of the constant sg_1 is chosen below a suitable threshold, property (P1) holds, and therefore MathSAT correctly does not find any counterexample. Regarding the choice of the constant sg_1, see the discussion in Section 7.

The trace generated by MathSAT is schematically shown in Figure 4. For each time instant, the trace shows the current locations of the ME, GBX, ATSM, and VALVE automata, the current values of the sp_me, sp_gbx, sp_atsm analog variables, the discrete transitions which take place at that time instant, and the synchronizations channels. For a better understanding of the trace, in Figure 5 we show a simplified version of the SPS computer automaton (only the part relevant to the simulation is shown). Notice that this automaton shows as example of

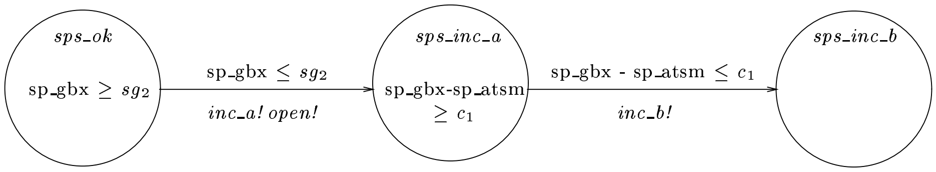

Fig. 5. SPS computer automaton (fragment)

triangular constraint, i.e. $sp\_gbx - sp\_atsm \leq c_1$ [$\geq c_1$], and of communication with shared variables (variables *sp_gbx* and *sp_atsm* model, respectively, the speed of the GBX and ATSM components).

The simulation begins at time T0, when an engine grippage takes place. Both the engine and the gearbox speeds begin to decrease. At time T1, the SPS computer detects a gearbox low speed condition, and therefore issues the opening of the valve (the VALVE and SPS computer automata synchronize on the *open* channel); as a result, the ATSM begins to increase its speed (SPS and ATSM synchronize on the *inc_a* channel). At time T2, the SPS computer issues a change in the ATSM dynamics (SPS and ATSM synchronize on *inc_b*). The simulation stops at time T3, when the gearbox speed reaches the value sg_1.

The same approach can be used for guided simulation. To give an example, we consider the following formula:

$$\begin{array}{l}(!\,\texttt{ME.loc\_eng\_flameout}\ \&\ !\,\texttt{GBX.loc\_broken}\ \&\ !\,\texttt{GBX.loc\_grippage}\ \&\\ !\,\texttt{VALVE.loc\_stuck\_closed}\ \&\ !\,\texttt{ATSM.loc\_broken}\ \&\ !\,\texttt{PTO.loc\_fused})\\ \quad\ \texttt{U}\ \ \texttt{GBX.sp\_atsm} >= \texttt{sa}_1\end{array} \tag{P2}$$

It is a variation of the previous reachability property, here we require that at the end of the path the speed of the ATSM component is greater than the constant value sa_1. Furthermore, by explicitly ruling out an engine *flameout*, we limit the possible failure modes of the main engine to *grippage*. As explained in Section 2, in presence of an engine failure, the ATSM component is responsible of carrying out the *cross-bleed* procedure, which consists in driving the gearbox with the pneumatic power coming from the valve. Correct functioning of the cross-bleed procedure requires the ATSM (which is initially idle) to start and bring up the gearbox speed. Using MathSAT, we are able to reconstruct a trace corresponding to the above property, which illustrates how the cross-bleed procedure is carried out. It is possible to tune the above simulation and perform further ones by adding further constraints on the trace to look for.

The performance of our method on the examples described above are reported in Table 1. For each problem length, we show computation time, total computation time up to that problem instance, and memory usage. Computation times include both parse and search time. The results have been obtained on a Pentium III machine 1.0 GHz, with 256 Mb memory, running Linux Redhat 7.1.

	Property P1				Property P2			
k	Time	Σ Time	Mem.	Result	Time	Σ Time	Mem.	Result
2	0.06	0.06	5.6	UNSAT	0.10	0.10	5.5	UNSAT
3	0.20	0.26	6.2	UNSAT	0.16	0.26	6.0	UNSAT
4	0.53	0.79	7.1	UNSAT	0.30	0.56	6.8	UNSAT
5	1.81	2.60	7.7	UNSAT	0.49	1.05	7.5	UNSAT
6	6.49	9.09	8.4	UNSAT	0.84	1.89	8.2	UNSAT
7	4.53	13.62	8.9	SAT	1.53	3.42	8.8	UNSAT
8					2.88	6.30	9.4	UNSAT
9					4.94	11.24	10.0	UNSAT
10					8.69	19.93	10.7	UNSAT
11					8.88	28.81	11.3	SAT

Table 1. Experimental results (Time in seconds, Memory in MB)

The minimal length trace generated by MathSat for P1 has length 7, whereas the one generated for P2 has length 11.

We also attempted a comparison with HyTech [HHWT97], a state-of-the-art tool for the analysis of hybrid systems. Differently from our approach, HyTech is based on the calculation of the reachable state space, and is therefore not limited to the bounded case. In principle, HyTech may not terminate when tackling an undecidable class of automata (as in the case of the SPS).

We encoded the models of the SPS, as closely as possible, into HyTech. Overflow errors in the underlying polyhedral libraries made it impossible for HyTech to compute the space of reachable configurations beyond the 5th iteration. We also attempted to use the `-o1` and `-o2` options, that are sometimes able to limit such problems, but in our case obtained no effect. From the point of view of performance, the time required by HyTech to reach the 5th iteration was 32 seconds, when run without options; the use of `-o1` and `-o2` required 50 and 86 seconds, respectively. The analysis is very preliminary, but seems to suggest that there is a clear potential in our techniques.

6 Related Work

The work presented in this paper builds upon our previous work on *timed systems* [ACKS02]. In [ACKS02], we showed how to reduce the problem of bounded model checking for timed systems to the satisfiability of a math-formula, which can then be checked by a SAT-solver. We also presented the MathSat solver [ABC+02b,ABC+02a], an efficient SAT-solver which is based on the integration of SAT techniques [BCCZ99] with some specialized decision procedures for linear mathematical constraints. A work related to ours, but still limited to timed systems, is [Sor02]. In the present work, as explained in Section 4.2, we have extended the encoding in order to deal with hybrid systems.

Our model for hybrid systems is closely related with the linear and rectangular hybrid automata models presented in [Hen96,HM00], the main difference being in the definition of the dynamics of the real-valued variables. In [Hen96], the dynamics (called *flow conditions*) of the real-valued variables are defined by means of linear constraints over the first derivatives of such variables, whereas in our model dynamics can be characterized by means of linear functions of the time delay, which directly constrain the behaviour of the variables. This approach is analogous to restricting the flows of the real-valued variables to stay inside a rectangular region, as in the rectangular automata model of [Hen96]. In fact, as noted in [HKPV98], under the hypothesis of working with *convex* linear constraints, requiring the flow to be inside a rectangular region amounts to requiring the existence of a smooth function inside the corresponding piecewise-linear envelope.

The model of hybrid I/O automata presented in [LSV03] is general enough to accommodate our model of hybrid automata. Discrete and continuous communication are achieved by means of, respectively, shared actions and shared variables. However, discrete events are not allowed to change the value of shared variables, as in our case.

As an alternative approach to the verification of hybrid systems, we cite [SRKC00], where the CheckMate tool is presented. CheckMate performs verification of hybrid systems using finite-state approximations called *quotient transition systems*. Although this approach is not restricted to linear hybrid automata, the verification analysis may be inconclusive, in which case a refinement of the current approximation may be attempted. An analysis of the current trends in model checking of hybrid systems can be found in [SSKE01].

This line of research has been carried on inside the ESACS [BVÅ+03] project (see `http://www.esacs.org`), an European-Union-sponsored project whose main goals are to define a methodology and a shared environment to improve the safety analysis practice for complex systems development. The Secondary Power System [BCC+03] is one of the case-studies investigated in ESACS. One of the main motivations for our research is the realization that the use of traditional finite-state model checking, based on the discretization of real variables, has a very hard time in dealing with the complexity of hybrid systems [BCC+03]. In fact, the results may depend on the step of discretization, and the state explosion problem makes such an approach infeasible in practice.

7 Conclusions and Future Work

In this paper, we have addressed the problem of verification of industrial systems that are naturally modeled as linear hybrid automata. The approach is an enhancement of the bounded model checking approach for timed systems proposed in [ACKS02] to the case of linear hybrid automata. Efficiency is gained by the use of the MathSAT solver. The main limitations are given by the undecidability of the analyzed class, and the constraints on the linearity of real variables dynamics. Despite this, however, the approach allows us to model and analyze systems of practical relevance, that HyTech is currently unable to deal with.

In the future, we will provide a more thorough experimental evaluation, by enlarging both the set of tools we compare with (some of them are cited in Section 6), and the set of case studies to analyze. Regarding the SPS example, we plan to experiment with more complex models, at different levels of granularity and abstraction (e.g., a two-sided model of the system). We will investigate how to optimize the MathSAT solver on these specific problems (e.g., by constraining the splitting variables in the style of [GMS98,Str00]), and will experiment with different encodings. As a first step towards bridging the gap between *bounded* model checking and *unbounded* verification, inductive reasoning techniques to prove invariant properties will be investigated. An important point we plan to address in the near future is concerned with *parametric* analysis, which is currently supported in HyTech. To exemplify, parametric analysis would allow us to replace the constant value sg_1 in property (P1) (see Section 5) with a parameter α in order to find out constraints on the parameter for which the property does or does not hold. Finally, in the future we plan to extend the framework to properties expressed in full LTL.

References

[ABC+02a] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In Andrei Voronkov, editor, *CADE-18: Conference on Automated Deduction*, volume 2392 of *LNAI*, pages 195–210, Copenhagen, Denmark, 2002. Springer.

[ABC+02b] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements. In Jacques Calmet, Bernard Benhamou, Olga Caprotti, Laurent Henocque, and Volker Sorge, editors, *CALCULEMUS-2002: Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, volume 2385 of *LNAI*, pages 231–245, Marseille, France, 2002. Springer.

[ACKS02] Gilles Audemard, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani. Bounded Model Checking for Timed Systems. In Doron A. Peled and Moshe Y. Vardi, editors, *FORTE 2002: Conference on Formal Techniques for Networked and Distributed Systems*, volume 2529 of *LNCS*, Houston, Texas, November 2002. Springer.

[BCC+03] M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorita. Improving Safety Assessment of Complex Systems: An industrial case study. In *Proc. Formal Methods Europe (FME 2003)*, 2003. To appear.

[BCCZ99] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In R. Cleaveland, editor, *Proc. 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.

[BVÅ+03] M. Bozzano, A. Villafiorita, O. Åkerlund, P. Bieber, C. Bougnol, E. Böde, M. Bretschneider, A. Cavallo, C. Castel, M. Cifaldi, A. Cimatti, A. Griffault, C. Kehren, B. Lawrence, A. Lüdtke, S. Metge, C. Papadopoulos, R. Passarello, T. Peikenkamp, P. Persson, C. Seguin, L. Trotta, L. Valacca, and G. Zacco. ESACS: An Integrated Methodology for Design and Safety Analysis of Complex Systems. In *European Safety and Reliability Conference (ESREL'03)*, pages 237–245. Balkema Publisher, 2003.

[CFG+01] F. Copty, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of Bounded Model Checking at an Industrial Setting. In *Proc. CAV'2001*, LNCS. Springer, 2001.

[DLL62] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[GMS98] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability. In *Proc. AAAI'98*, pages 948–953, 1998.

[Hen96] T.A. Henzinger. The Theory of Hybrid Automata. In *Proceedings 11th Annual International Symposium on Logic in Computer Science (LICS'96)*, pages 278–292, New Brunswick, New Jersey, 1996. IEEE Computer Society Press.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[HKPV98] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's Decidable About Hybrid Automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[HM00] T.A. Henzinger and R. Majumdar. Symbolic Model Checking for Rectangular Hybrid Systems. In S. Graf and M. I. Schwartzbach, editors, *Proceedings 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 142–156, Berlin, Germany, 2000. Springer-Verlag.

[LSV03] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O Automata. *Information and Computation*, 2003. To appear.

[Sor02] M. Sorea. Bounded Model Checking for Timed Automata. In *Proceedings 3rd Workshop on Models for Time-Critical Systems (MTCS'02)*, Brno, Czech Republic, 2002.

[SRKC00] B.I. Silva, K. Richeson, B.H. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using CheckMate. In *Proc. ADPM 2000, Automation of mixed processes: Hybrid Dynamic Systems*. Shaker Verlag, 2000.

[SSKE01] B.I. Silva, O. Stursberg, B.H. Krogh, and S. Engell. An assessment of the Current Status of Algorithmic Approaches to the Verification of Hybrid Systems. In *Proc. 40th Conference on Decision and Control*, 2001.

[Str00] Ofer Strichman. Tuning SAT checkers for bounded model checking. In *Proc. CAV00*, volume 1855 of *LNCS*, pages 480–494, Berlin, 2000. Springer.