HyComp: an SMT-based Model Checker for Hybrid Systems[⋆]

Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta

Fondazione Bruno Kessler
{cimatti,griggio,mover,tonettas}@fbk.eu

Abstract. HyComp is a model checker for hybrid systems based on Satisfiability Modulo Theories (SMT). HyComp takes as input networks of hybrid automata specified using the HyDI symbolic language. HyComp relies on the encoding of the network into an infinite-state transition system, which can be analyzed using SMT-based verification techniques (e.g. BMC, K-induction, IC3). The tool features specialized encodings of the automata network and can discretize various kinds of dynamics.
HyComp can verify invariant and LTL properties, and scenario specifications; it can also perform synthesis of parameters ensuring the satisfaction of a given (invariant) property. All these features are provided either through specialized algorithms, as in the case of scenario or LTL verification, or applying off-the-shelf algorithms based on SMT. We describe the tool in terms of functionalities, architecture, and implementation, and we present the results of an experimental evaluation.

1 Introduction

Embedded systems (e.g. control systems for railways, avionics, and space) feature the interaction of discrete systems with the environment by means of controlled and monitored variables that evolve continuously in time. The validation and verification of embedded systems designs must often take into account a model of the continuous evolution of such variables. Hybrid systems [26] are a clean modeling framework for embedded systems because they exhibit both continuous transitions ruled by flow conditions and discrete changes represented with logical formulas.

A fundamental step in the design of these systems is the validation and verification of the models, performed by checking specifications expressed e.g. as invariants, temporal-logic formulas, or scenarios. In spite of the undecidability of these problems, several verification techniques have been developed and have proved to be applicable in a wide number of cases. An emerging approach to the verification of hybrid systems is the application of techniques based on Satisfiability Modulo Theories (SMT). The hybrid system is encoded into a symbolic transition system and reachability problems are represented by means of first-order formulas, which can then be solved with SMT-based techniques. Thanks to the strong progress in the field of SMT, these approaches are increasingly applied in real settings.

⋆ This work was carried out within the D-MILS project, which is partially funded under the European Commission's Seventh Framework Programme (FP7).

In this paper we present HYCOMP, a symbolic model checker for hybrid systems. HYCOMP is built on top of the NUXMV model checker [9], and implements various verification techniques based on SMT. HYCOMP takes as input networks of hybrid automata specified using the HyDI symbolic language [15]. HYCOMP relies on the encoding of the network into an infinite-state transition system, which can then be analyzed using various SMT-based verification techniques provided by NUXMV (e.g. BMC, K-induction, IC3). The tool features specialized encodings of the automata network and can discretize various kinds of dynamics. HYCOMP can verify invariant and LTL properties [14], and scenario specifications [16]; it can also perform synthesis of parameters ensuring the satisfaction of a given (invariant) property [12]. The tool has been used as a research platform for developing novel verification techniques, both for hybrid systems [8, 16, 14, 33, 17] as well as for more general infinite-state systems [12, 13]. Moreover, it has been used in different projects, both industrial and research-oriented ones (such the ESA-funded projects IRONCAP and HASDEL, and the FP7 project MISSA). In these projects HYCOMP turned out to be really useful to support the analysis of asynchronous systems (also in the discrete case, as a front-end to NUXMV) and to solve expressive verification problems (e.g. to verify temporal properties of real-time systems). The tool is freely available for non-commercial use and can be downloaded at `http://hycomp.fbk.eu`. In this paper, we focus on the technical details about HYCOMP as a tool.

Related tools There exist several related tools and languages for the verification of hybrid systems. These tools are mainly focused on the verification of invariants and most of them compute an overapproximation of the set of the reachable states. HYTECH [24] is a model checker for linear hybrid automata, which represents the continuous part of the reachable states using polyhedra. PHAVER [21] and SPACEEX [22] model affine continuous dynamics with inputs. They check invariant properties computing an approximation of the set of the reachable states using different techniques (polyhedra and support functions). Other model checkers, HSOLVER [36], D/DT [3] and ARIADNE [6], FLOW* [10], verify invariants of non-linear hybrid systems.

KEYMAERA [35] is a theorem prover for hybrid systems. It can handle non-linear hybrid systems, with symbolic parameters and an unbounded number of components. Opposed to HYCOMP, it may require a manual user intervention during the proof process and it supports a subset of LTL properties.

HybridSAL [37] is very similar to HYCOMP. The tool encodes linear hybrid systems as infinite-state transition systems, which can be verified using the SAL [32] model checker. HybridSAL also implements other abstraction techniques (e.g. See [40]), but it does not implement the quantifier free encoding for polynomial hybrid systems. The tool cannot prove LTL properties, it does not provide verification algorithms that exploit the hybrid automata network, and is not integrated with the efficient invariant verification algorithms of NUXMV (e.g. *IC3*).

In the fragment of timed automata, the reference tool is UPPAAL [5]. It supports the model checking of a subset of TCTL (Timed Computation Tree Logic) properties. The reachability is explicit in the discrete states of the automata. The tool does not handle hybrid systems and LTL properties. Moreover, UPPAAL does not allow the user to model parametric designs.

ATMOC is an SMT-based model checker for invariant [30], LTL [28] and MITL [29] properties for symbolic timed automata.

MCMT [23] and PASSEL [27] are two other SMT-based tools for verifying parameterized systems composed by timed or linear hybrid automata. They differ since the focus is on systems with an infinite number of processes, which HYCOMP cannot handle. They cannot verify LTL and scenario specification, while only MCMT can synthesize parameters. Neither of them can analyze systems with complex dynamics.

Outline In §2, we give a brief overview of the HyDI modeling language. In §3 we describe the tool functionalities; we provide implementation details in §4, and in §5 we present results of an empirical evaluation of HYCOMP wrt. related state-of-the-art tools. We conclude the paper in §6.

2 Modeling Language

Overview The input language of HYCOMP is HYDI [15] (Hybrid automata with DIscrete interaction). A HYDI program describes a network of hybrid automata interacting with standard discrete synchronizations. HYDI extends the language of the NUXMV model checker (which in turn extends the language of the NUSMV model checker with infinite domain types) with specific constructs related to the hybrid semantics and to the synchronization of asynchronous processes. The network is defined in the *main* module, which declares a set of processes (defined by instantiations of modules) and a set of synchronizations. The modules contain the definition of the hybrid behavior. The discrete-time part is described with a set of discrete variables (e.g., Boolean, integer, real) and a set of formulas representing the initial states, the invariant conditions, and transition relation. The continuous-time part is described with continuous variables, flow and urgent conditions.

A simple example Figure 1 shows a small example of communicating tanks specified in HYDI. Each tank has an input and output flow of water. The input water flows only in one of the tanks and when this tank is full, a valve switches the water flow to the other tank. While one tank is being filled with new water, the other is being emptied since there is always a flow of water that goes out of each tank.

More specifically, *tank1* and *tank2* are two instances of the module *Tank*, which is instantiated with different values of the parameters. These are a flag *initial*, which chooses which tank initially takes the incoming water, the maximum input flow, and the minimum output flow. The synchronizations connect the event *noflowin* of *tank1*, which represents the stop of flow in *tank1*, with the event *flowin*, which starts the flow in *tank2*, and vice versa.

The discrete state space of each tank is described with two variables: *state* and *flow*. The *state* variable represents the condition of the tank to be *empty*, *full*, or *half*-empty/full (either filling or emptying). The *flow* variable is a Boolean that represents if there is or not an input flow of water. The continuous variables *q*, *inq*, *outq* represent the quantity of water that is present in the tank, the incoming quantity and the outgoing quantity, respectively.

```
MODULE main
VAR   tank1: Tank(TRUE,2,1);
      tank2: Tank(FALSE,2,1);
SYNC tank1,tank2
      EVENTS flowin,noflowin;
SYNC tank1,tank2
      EVENTS noflowin,flowin;

MODULE Tank(initial, maxin, minout)
EVENT flowin, noflowin, tau;
VAR state: {empty, half, full};
    flow: boolean; inq: continuous;
    q: continuous; outq: continuous;

INIT q=0 & (initial <-> flow)
INVAR   q>=0 & q<=100 &
        (state=empty -> q=0) &
        (state=full -> q=100)
```

```
TRANS
(EVENT=flowin -> (next(flow)=TRUE &
                  next(state)=state))&
(EVENT=noflowin -> (state=full &
            next(flow)=FALSE &
            next(state)=state))&
(EVENT=tau -> (next(flow)=flow)) &
 next(q)=q

FLOW
((state=empty & !flow) -> der(q)=0) &
(!(state=empty & !flow) ->
            der(q)=der(inq)-der(outq));
FLOW
(!flow -> (der(inq)=0)) &
(flow -> (der(inq)>0 &
            der(inq)<=maxin))&
der(outq)>=minout
```

Fig. 1. A small HyDI example.

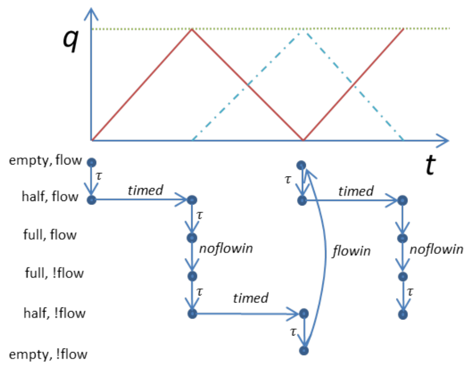


Fig. 2. A possible execution of the *tank1* process in the tank example. The lower part shows the sequence of transitions and discrete states. In the upper part, the quantity q is plotted against time (the dash line represents the quantity in the other tank).

Any transition satisfying the transition and invariant conditions is valid. Therefore, the *state* variable can change only with an internal *tau* event; when q is 0 then it can pass from *half* to *empty* and backwards, while when q is 100 the state can pass from *half* to *full* and backwards; when the tank receives the event *flowin* the *flow* variable becomes true; when the tank is full, it triggers the event *noflowin* switching the *flow* variable to false. Note how the symbolic representation allows a compact definition of discrete states (there are implicitly six discrete states in the example) and discrete transitions (six in the example).

The derivative of q is always given by the difference between the rate of water flowing in and the rate of water flowing out. The water flowing in the tank is zero if the *flow* variable is false, otherwise it is positive and not greater than a *maxin* value that is passed as parameter to the *tank* module. The rate of water flowing out is instead always greater than another parameter named *maxout*.

Intuitively, the system performs discrete and continuous transitions. In the former case, the variables evolve according to the invariant and transition conditions. In the latter case, the discrete variables do not change, while the continuous variables change according to the invariant and flow conditions (with an implicit elapsing of time). For example, Figure 2 shows a trace of *tank1* that starts from the state *empty* with *flow=TRUE* and *q=0*; then a *tau* transition changes the state into *half*; then a *timed* transition makes *q* reach the value *100* and another *tau* transition changes the state into *full*; in this state, *tank1* can synchronize with *tank2* switching *flow* into *FALSE*. Now a *tau* transition change the state to *half*, and another *timed* transition makes *q* reach the value *0*. The trace continues in this way oscillating the quantity *q* between *0* and *100*.

Supported continuous dynamics HYCOMP supports different types of flow conditions. Each type enables different kinds of verification. In particular, we distinguish among the following classes of hybrid systems:

– Hybrid systems with linear constraints (see [26]), also known as *linear hybrid automata*, where the flow condition is given by symbolic constraints over the derivatives of continuous variables.
– Hybrid systems with linear ODE (see [31, 22]), also known as *linear hybrid systems* where the flow condition is defined by a system of linear Ordinary Differential Equations (ODE).
– Hybrid systems with polynomial dynamics (see [20]): hybrid systems such that the continuous evolution is described with a function over time, thus without using derivatives.

In the first two cases, the flow condition is in the form $\phi(V_D) \rightarrow \psi(V_C, \dot{V}_C)$ where $\phi(V_D)$ is a formula over the discrete variables defining where the flow is valid, while $\psi(V_C, \dot{V}_C)$ is a formula over the continuous variables and their derivatives defining the actual dynamics. Both ϕ and ψ are restricted to linear arithmetic.

In the case of hybrid systems with linear constraints, ψ is a conjunction of equalities or inequalities over derivatives only (thus, without occurrences of continuous variables). The tank example falls in this class. In the case of hybrid systems with linear ODEs, ψ is a conjunction of equalities over both derivatives and continuous variables. The case of hybrid systems with polynomial dynamics, are supported with another keyword **`EXPLICIT_FLOW`**, which must be followed by an equality defining the next value of a continuous variable after a timed transition as a polynomial of the *delta* variable representing the elapsed time.

Supported synchronizations Synchronizations specify if two events of two processes must happen at the same time. If two events are not synchronized, they interleave. Such synchronization is quite standard in automata theory and process algebra. It has been generalized with guards to restrict when the synchronization can happen.

Processes can share variables through the passage of parameters in the instantiations. However, they are limited to read the variables of other processes. This permits an easy identification of when the variables do not change even if the transitions are described with a generic relation (compared to a more restrictive functional description).

In order to capture the semantics of some design languages, it is necessary to enrich the synchronization with further constraints that specify a particular policy scheduling the interaction of the processes. For this reason, it is possible to specify a *scheduler* in the main module of the HYDI program in terms of state variables, initial and transition conditions. These conditions may predicate over the events of the processes.

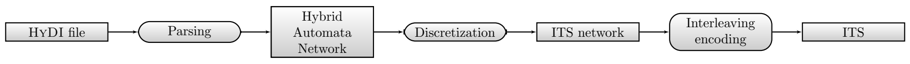


Fig. 3. Encoding process

3 Description of tool functionalities

3.1 Encodings

HyComp implements the encoding of a hybrid automata network into *Infinite-state Transition Systems* (*ITSs*). The encoding process, shown in Figure 3, is constituted of two main phases: the *discretization* and the *interleaving encoding*. The input of this process is a HyDI program, while the resulting transition system can be exported into the nuXmv format. If the input HyDI program is purely discrete, HyComp supports an alternative flow in the encoding process, which can parse a discrete HyDI file into a discrete asynchronous network of components, thus bypassing the discretization phase. The *discretization* phase encodes the continuous variables, the flow and urgent conditions of the hybrid automata network into a network of discrete *ITSs*. In the *interleaving encoding*, the tool translates the interleaving of the transition systems of the network and their synchronization constraints into a synchronous composition. We refer the reader to [15, 34] for the formalizaion with proofs of correctness of the encoding process.

Discretization of a process The discretization phase translates each HyDI process P_c into a discrete HyDI process P_d (a process with no continuous variables, no flow and urgent conditions). Continuous variables are converted into discrete real variables and an additional real variable *delta* is introduced to represent the amount of time elapsed in the continuous transition. Moreover, P_c defines an additional event value *timed*, which labels the discrete transition of P_c that encodes the continuous transition of P_d.

The definition of the timed transition ensures that all the discrete variables of P_c do not change, that the amount of time elapsed is non-negative, and that the continuous variables evolve according to the flow condition and the *delta* variable. The different types of supported dynamics described in Section 2 are handled in different ways.

In the linear hybrid automata case, the predicate of the flow condition are a linear combination of the first derivatives of the continuous variables (i.e. $\sum_{x \in X} \dot{x} + a \leq 0$). The discretization encodes a linear combination as a formula P_d that relates the change of values of the variables to the amount of time elapsed `delta`. For the tank example, we have the following discretization:

```
TRANS (EVENT = timed) -> (
(delta=0 -> (next(q)=q & next(inq)=inq & next(outq)=outq)) &
((state=empty & !flow) -> next(q)=q) &
(!(state=empty & !flow) -> (
          next(q)-q=next(inq)-inq-next(outq)+outq)) &
(!flow -> (next(inq)=inq)) &
(flow -> (next(inq)-inq > 0 & next(inq)-inq <= delta*maxin)) &
next(outq)-outq >= delta*minout)
```

In the linear hybrid automata we just encode the invariant condition of P_c as **`INVAR`** in P_d. The encoding is correct due to the convexity of invariant conditions (that is enforced in the HYDI syntax).

In the polynomial hybrid system case, the input model already defines an explicit solution in function of `delta`. HYCOMP can also compute a polynomial explicit solution in `delta` for some linear hybrid systems. The capabilities of the tool are limited to a very simple case, where the explicit solution can be obtained by substitution (e.g. given $\dot{x} = y$, $\dot{y} = 1$ we can easily compute $y(t) = t + y(0)$ and $x(t) = \frac{1}{2}t^2 + y(0)t + x(0)$). Due to the possible non-linearity of the solution the invariant may be violated for some value $0 < \epsilon <$ *delta*, even if the invariant is convex and it holds on the interval points (0 and *delta*). For this reason, HYCOMP implements a specialized encoding [17], which limits the duration of the timed transition in order to always observe the points where the invariant changes its truth value.

For linear hybrid systems, HYCOMP implements the time-aware relational abstraction encoding of [33]. The idea of relational abstraction is to obtain a formula $R(X, X')$ such that, if there is a trajectory from v to v' in the linear system, then v, v' is a model for $R(X, X')$. $R(X, X')$ over-approximates the original hybrid system and thus the resulting encoding can be used to prove safety properties.

The discretization process encodes the **`URGENT`** conditions that can be expressed in HYDI. An **`URGENT`** condition is a formula $U(V)$, where V are discrete variables, such that if $U(V)$ holds time cannot elapse. HYCOMP encodes the urgent condition as **`TRANS`** `U(V) -> delta = 0`.

The discretization process can be controlled by two additional options. The first option automatically adds a clock variable *time* that keeps track of the total amount of time elapsed in the system. The variable may complicate some verification algorithms (e.g. the *BMC* algorithm for LTL properties is completely unuseful when using this encoding, since in the transition system there are no more infinite paths where the value of the time diverges), but it may be necessary for other algorithms (e.g. the one based on local-time semantic and *K-zeno*). The second option removes from the encoding the possibility to have a path with two consecutive continuous transitions[1]. In this case the encoding adds an additional Boolean variable *b*, which records if the last transition was the time elapse (**`EVENT`** `= timed ->` **`next`**`(b)`) and forbids two consecutive time elapses (**`EVENT`** `= timed -> !b`).

Discretization of the network HYCOMP can perform two different encodings of hybrid automata networks, one based on *global-time semantics* and the other on *local-time semantics* [4]. The global-time semantic captures the standard semantic of a network of hybrid automata: time elapses in all the automata in the network and for the same duration. Instead, in the local-time semantic each automaton keeps the total amount of time elapsed in a local clock variable, which is incremented *independently* by each automaton. In this way, time may elapse in one automaton but not in the others. The encoding also forces that, when automata synchronize, they must also agree on the value of their local time clocks. The same condition on clocks is also required at the end of a run.

[1] The option is not sound for the encoding of polynomial hybrid systems

Global-time and local-time semantic are encoded using synchronization constraints. For the global-time semantic, HYCOMP adds a strong synchronization constraint between each pair of automata in the network. For the tank example, it would add the following **SYNC** constraint:

```
SYNC tank1,tank2  EVENTS timed, timed
  CONDITION tank1.delta = tank2.delta;
```

The **CONDITION** constraint must hold when there is the synchronization.

HYCOMP encodes the local-time semantic changing each synchronization condition and invariant property of the system. The encoding forces that the local time variable of the automata must have the same value when there is a synchronization. In the tank example, HYCOMP would create the following **SYNC** constraints:

```
SYNC tank1, tank2  EVENTS flowin, noflowin
  CONDITION tank1.time = tank2.time;
SYNC tank1, tank2 EVENTS noflowin, flowin
  CONDITION tank1.time = tank2.time;
```

The same condition about time has to be enforced also on **INVARSPEC** properties. HYCOMP encodes each property **INVARSPEC** `P` as **INVARSPEC** `S -> P`, where S encodes the equality of all the local time variables of the network processes.

Interleaving encoding In order to convert the asynchronous composition of the processes into a synchronous composition, HYCOMP adds to each process an additional event, *stutter*. This represents an additional transition where the process remains in the same state while the other processes move.

Then, HYCOMP encodes the synchronization constraints as an additional global **TRANS** constraints. The encoding of the first **SYNC** declaration of the tank example is:

```
TRANS tank1.EVENT = flowin <-> tank2.EVENT = noflowin
```

HYCOMP provides two additional options. The *step semantic* relaxes the interleaving encoding allowing to execute in parallel several independent transitions. The other option allows to generate an encoding partitioned by the values of the **EVENT** variable.

3.2 Verification

HYCOMP provides the possibility to verify different kinds of properties, namely invariants, LTL, and scenario specifications. These are based on different verification algorithms, which work either directly on the network of asynchronous ITSs (scenario verification, *BMC* using shallow synchronization) or on the synchronous transition system (*BMC*, *IC3*, *K-induction*).

Invariant properties HYCOMP implements several algorithms to verify invariant properties. The property is expressed as a first-order formula over the state variables of the hybrid automata network. The tool can either prove or falsify the property and, in the latter case, construct a finite path that witnesses the violation.

HyComp verifies invariant properties by using several SMT-based algorithms implemented in nuXmv: *IC3*, *K-induction*, their combination with implicit predicate abstraction [38, 13] and Bounded Model Checking (BMC). HyComp implements specialized BMC encodings for networks of hybrid automata: the tool implements a BMC encoding that alternates continuous and discrete transitions [1] and the *shallow synchronization* encoding [8], which exploits local-time semantic to obtain shorter counterexample paths.

We note that all the verification algorithms are enabled when the encoding is expressed in *Linear Real Arithmetic Theory*. This is the case if the hybrid automaton is linear or when using time-aware relational abstraction, but it is not the case for polynomial hybrid systems. The limitation is due to the integration of an SMT solver supporting the *Theory of Reals* (i.e. support for polynomials), since the tool only provides an experimental implementation of BMC that uses the z3 or iSAT [2] SMT-solvers[3].

LTL properties The tool allows the user to verify LTL properties interpreted over discrete sequences of states. It implements a specialized algorithm, *K-zeno* [14], which is based on a reduction of liveness to the reachability of an accepting condition and excludes Zeno paths (unrealistic paths where time does not diverge) from the analysis.

HyComp allows the user to call the nuXmv BMC algorithms for LTL verification to find a violation to the LTL property. However, in this case the Zeno paths of the hybrid automata are excluded in the encoding of the hybrid automata network using a fairness condition (i.e. a condition that holds infinitely often) that enforces the divergence of time. Note that the BMC algorithms will only find lasso-shaped paths.

Scenario specifications The last kind of specification verified by HyComp are scenarios: a scenario allows a user to specify the exchange of messages in a network of hybrid automata. The scenario specifications supported by HyComp are a variant of Message Sequence Charts (MSC). For all the automata in the network, an MSC defines a sequence of events (i.e. labels of the automata) and constraints evaluated when an event happens (e.g. the system must execute an event within a given amount of time). The MSC is *feasible* if there exists a path in the hybrid automata network that simulates it and that also satisfies the MSC constraints. Otherwise, the MSC is *unfeasible*.

HyComp implements two different approaches to verify scenario specifications. In one approach, the tool reduces the problem of scenario verification to a reachability problem, using an automaton to monitor the MSC feasibility. The other approach [16] exploits local-time semantic and consists of a specialized BMC encoding of the problem. The approach may either find a witness of feasibility or prove that a scenario is not feasible, using a variant of *K-induction*.

3.3 Parameter Synthesis

The tool allows the user to synthesize the set of parameter values of the system that guarantee its safe behavior. For example, the tool may be used to automatically syn-

[2] `http://z3.codeplex.com`, `http://projects.avacs.org/projects/isat`

[3] HyComp does not link or distribute z3 or iSAT, which should be installed by the end user

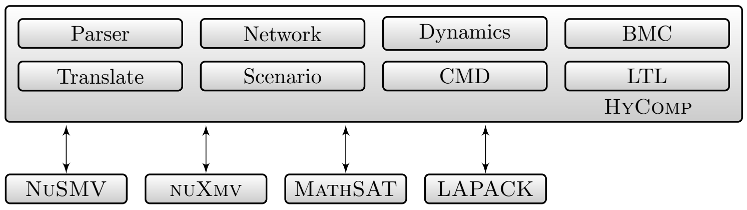


Fig. 4. HyComp architecture

thesize timeout values or deadlines that the system must guarantee (e.g. the maximum timeout to send a packet in a communication protocol).

In our framework, parameters are specified as **FROZENVAR** (a variable that never change its value during the system execution) and the safe behavior as an invariant property. The tool returns a formula of the parameters that represents the (possibly non-convex) feasible region of parameters.

HyComp uses the parameter synthesis algorithm implemented in nuXmv [12].

4 Tool architecture and implementation details

4.1 Architecture

In Figure 4 we show the architecture of the tool. HyComp uses as libraries the model checkers NuSMV and nuXmv [9] and the MathSAT [18] SMT solver.

HyComp uses several data structures and functions from NuSMV: its formula representation and manipulation package, its type system, its functions for flattening of hierarchical modules and its representation of transition systems.

HyComp uses the SMT-based algorithms implemented in nuXmv (e.g. *IC3*, *K-induction*, BMC, parameter synthesis) and also the nuXmv front-end to MathSAT. The front-end exposes the MathSAT functionalities (satisfiability check, incremental interface, extraction of unsat cores and interpolants), provides an automatic declaration of the variables in the solver and an automatic conversion from different formula representations (nuXmv and MathSAT representations). Finally HyComp also uses the linear algebra library LAPACKE[4] for the computation of relational abstractions.

The internal architecture of the tool is represented in the upper part of Figure 4. The tool is divided in packages that clearly separate different functionalities. The *parser* package is used to parse and type check a HyDI file. The results of this phase is a network of hybrid automata. The data structures that represent networks of hybrid automata and of transition systems are defined in the *network* package. All the encoding process is contained in the *translate* package, which also provides the functions to discretize continuous dynamics. Different representations of a continuous system and functions used to manipulate them are defined in the *dynamics* package. The verification algorithms for LTL verification is implemented in the *ltl* package, while the specialized BMC algorithms are implemented in the *bmc* package. Finally, the package *scenario*

[4] `http://www.netlib.org/lapack/`

implements the scenario verification algorithms and the *cmd* package provides the user commands that directly call the NUXMV algorithms (e.g. *IC3*, parameter synthesis).

4.2 Implementation Details

Network representation HYCOMP represents asynchronous network of processes, which can be either hybrid automata or transition systems. The data structure is agnostic of the process type and provides common functionalities to represent and manipulate synchronization constraints. One of these is the computation of the transitive closure of synchronizations (in HYDI, if there is a synchronization between the event a of p_1 and the event b of p_2, and another synchronization between the event b of p_2 and the event c of p_3, then there is an implicit synchronization between a of p_1 and c of p_3). HYCOMP represents the graph of synchronizations, where nodes are processes and undirected edges are synchronizations, and computes its transitive closure.

Mapback of results While the user is aware of the existence of the various encoding phases, the tool hides all the artifacts of the encoding. This is important to avoid misunderstanding and allows for modifying the encoding in the future. The encoding phases keep a map from a symbol in the source model to its correspondent symbol in the encoding (e.g. a continuous variable is mapped to the real variable used in the discrete encoding). Since we have several transformations (discretization and encoding of interleaving) we have several maps, which can be composed and inverted, to map the results obtained during verification (e.g. counterexample paths) to the original model.

Symbolic enumeration of discrete locations The discretization in the case of linear hybrid systems requires to reason on a system of ODEs. Since the input is symbolic, HYCOMP has to enumerate the set of discrete locations and, for each one of them, compute the correspondent system of ODEs. For example, consider the following `FLOW`:

```
FLOW der(x) = x & (b -> der(y) = 1) & (!b -> der(y) = 0);
```

If b is true, then the linear system is `der(x) = x & der(y) = 1`, otherwise we have `der(x) = x & der(y) = 0`. HYCOMP enumerates all the possible disjoint subsets of discrete locations using MathSAT. The idea is to use an additional Boolean variable for each discrete condition in the flow declarations (e.g. the variable `f0` for `TRUE`, `f1` for `b` and `f2` for `!b`), encoding that the variable is true if and only if the condition is true (e.g. `f0 <-> TRUE && f1 <-> b && f2 <-> !b`). Then, MathSAT enumerates all the possible satisfying partial models formed by the Boolean variables (in the example they are `f0 & f1 & !f2` and `f0 & !f1 & f2`). Each partial model identifies a symbolic discrete location where the `FLOW` is a system of ODEs.

5 Experimental evaluation

We show an experimental comparison on the verification of invariant properties on timed and linear hybrid automata. This comparison is novel and complements the comparisons for LTL, scenario verification, and parameter synthesis presented in previous papers [14, 16, 12].

	IC3-IA		*IC3-IA-ALT*		UPPAAL		UPPAAL-RED	
	#p	time	#p	time	#p	time	#p	time
Csma-cd	12	2608.94	**14**	258.22	6	18.50	6	251.96
Fischer	8	1466	**14**	476.88	11	312.48	11	401.67
FischerSAL	6	258.15	5	463.92	**11**	356.49	11	451.35
HDDI	14	220.77	14	224.55	**14**	2.21	14	3.07
Lynch-Mahata	8	1710.81	6	494.12	**11**	416.69	11	534.05
All instances	48	6265	53	1918	**53**	**1106**	53	1642

Fig. 5. Results on mutual exclusion properties. **#p** is the total number of instances solved and **time** the time in seconds took to solve them.

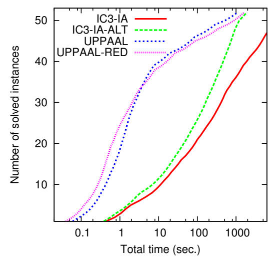


Fig. 6. Cumulative plot on mutex properties

The main goal of the experimental evaluation is to position the tool with respect to the existing state of the art and not to evaluate the algorithms. For the latter goal, one would need more benchmarks and properties.

All the experiments have been performed on a cluster of 64-bit Linux machines with a 2.7 Ghz Intel Xeon X5650 CPU, with a memory limit of 4Gb and a time limit of 900 seconds. The HYCOMP tool and the benchmarks used in the experiments are available at `https://es.fbk.eu/people/mover/tests/tacas15hycomp.tar.bz2`.

5.1 Timed automata

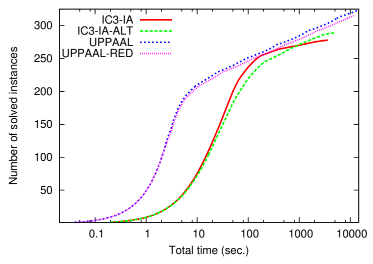
We compared HYCOMP with UPPAAL [5] on timed automata benchmarks obtained either from the UPPAAL or the MCMT [23] distributions, converting the benchmark in the HYDI language. We selected the following benchmarks: the *Fischer* protocol, one of its variant, *FischerSAL*, the *Csma-cd* protocol, the *HDDI* protocol and the *Lynch-Mahata* protocol. For each benchmark we checked the mutual exclusion property and we generated several invariant properties, which specify that a specific configuration of locations in the network is not reachable. We generated several instances of the benchmarks increasing the number of processes.

For HYCOMP, we run *IC3* with implicit predicate abstraction (*IC3-IA*), the BMC implementation that alternates timed and discrete transitions (*BMC*) and *IC3* on the encoding that avoids two consecutive timed transition (*IC3-IA-ALT*). In all the cases, we used the global-time semantic. For UPPAAL, we used two different configurations[5]: in the first one (UPPAAL) we used Different Bounded Matrices representation, while in the second one (UPPAAL-RED) we used the minimal constraint systems representation.

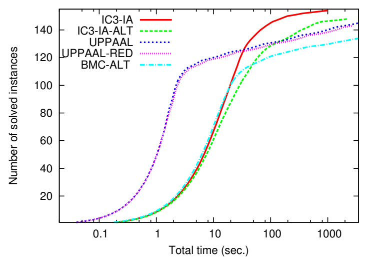
In Figure 6 and Table 5 we show the comparison on the mutual exclusion properties. We see that UPPAAL is generally faster than *IC3-IA-ALT* and *IC3-IA*. In detail, *IC3-IA* and *IC3-IA-ALT* outperform UPPAAL on two benchmarks, while they are worse on the other three: there are several instances that can be solved by UPPAAL and not by HYCOMP and vice-versa.

In Figure 7 we show the results verifying the automatically generated properties. UPPAAL solves more instances (325 in 14581 sec.) than *IC3-IA-ALT* (290 in 4776 sec).

[5] In both cases we used the version 4.0.14 of UPPAAL with the options "*-n 0*, *-o 0*, *-s 1*"



(b) All the properties



(b) Unsafe properties

Fig. 7. Cumulative plot on the automatically generated properties

	IC3-IA		*IC3-IA-ALT*		SpaceEx	
	#p	time	#p	time	#p	time
Distributed Controller	**14**	402.56	14	451.08	1	0.69
Fischer	5	905.82	**5**	558.49	3	74.73
Nuclear Reactor	14	783.02,	**14**	96.67	1	26.99
Navigation safe	28	1823.25	28	1768.78	**28**	43.76
Navigation-double safe	**17**	3213.59	16	3198.29	13	1599.16
Navigation unsafe	28	3280.17	28	4525.07	**28**	43.74
Navigation-double unsafe	**17**	4722.68	13	2438.36	14	2496.69
All instances	**123**	15131	118	13037	46	1745

Fig. 8. Results on LHA benchmarks. **#p** is the total number of instances solved and **time** the time in seconds took to solve them.

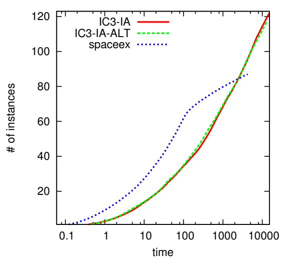


Fig. 9. Cumulative plot for LHA benchmarks

If we focus on unsafe properties, we see that *IC3-IA* (155 in 983 sec.) and *IC3-IA-ALT* (149 in 2274 sec.) are more effective than UPPAAL (146 in 3426 sec.).

5.2 Linear Hybrid Automata

We compared HyComp and SpaceEx [22] on the verification of invariant properties of the following linear hybrid automata benchmarks: an LHA version of the *Fischer* protocol [2], the control of nuclear reactor of [39] (*Nuclear Reactor*), the model of a robot controller [25] (*Distributed Controller*) and two LHA variants (*Navigation*, *Navigation-double*) of the navigation benchmark [19]. *Navigation* models describe the movement of an object in an nxn grid of square cells, which will eventually reach a stable region. *Navigation-double* is a variant with two grids and two objects.

For all the benchmarks, except the navigation ones, we checked a mutual exclusion property and we generated several instances increasing the number of components in the network. For *Navigation* and *Navigation-double*, we increased the number of cells in the grid and considered a safe and an unsafe property (the object is in the stability region after or before a given time).

For HyComp, we run *IC3-IA* and *IC3-IA-ALT*, while for SpaceEx we used the *phaver* scenario. We show the results of the comparison in Figure 9 and Table 8.

6 Conclusion

We presented HyComp, an SMT-based model checker for hybrid systems. The tool features an expressive input language and a rich set of functionalities, such as verification of invariant and LTL properties, verification of scenario specifications and parameter synthesis. We demonstrated the potential of the tool, showing its competitiveness with the state of the art.

We plan to develop HyComp in several directions, adding algorithms for abstraction-refinement in presence of complex dynamics, integrating more expressive specifications such as HRELTL and improving the underlying SMT-based verification algorithms. We also have plans to integrate HyComp in analysis tools for safety assessment (XSAP [7]) and contract-based design (OCRA [11]).

References

1. Ábrahám, E., Becker, B., Klaedtke, F., Steffen, M.: Optimizing bounded model checking for linear hybrid systems. In: VMCAI. pp. 396–412 (2005)
2. Alur, R., Dang, T., Ivancic, F.: Counterexample-guided predicate abstraction of hybrid systems. Theor. Comput. Sci. 354(2), 250–271 (2006)
3. Asarin, E., Dang, T., Maler, O.: The d/dt Tool for Verification of Hybrid Systems. In: CAV. pp. 365–370 (2002)
4. Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: CONCUR. pp. 485–500 (1998)
5. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Uppaal - a tool suite for automatic verification of real-time systems. In: Hybrid Systems. pp. 232–243 (1995)
6. Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume guarantee verification of nonlinear hybrid systems withariadne. International Journal of Robust and Nonlinear Control 24(4), 699–724 (2014)
7. Bozzano, M., Villafiorita, A.: The FSAP/NuSMV-SA Safety Analysis Platform. STTT 9(1), 5–24 (2007)
8. Bu, L., Cimatti, A., Li, X., Mover, S., Tonetta, S.: Model checking of hybrid systems using shallow synchronization. In: FMOODS/FORTE. pp. 155–169 (2010)
9. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: CAV. pp. 334–342 (2014)
10. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. pp. 258–263 (2013)
11. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A tool for checking the refinement of temporal contracts. In: ASE. pp. 702–705 (2013)
12. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Parameter synthesis with ic3. In: FMCAD. pp. 165–168 (2013)
13. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Ic3 modulo theories via implicit predicate abstraction. In: TACAS (2014)
14. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Verifying LTL properties of hybrid systems with k-liveness. In: CAV. pp. 424–440 (2014)
15. Cimatti, A., Mover, S., Tonetta, S.: Hydi: A language for symbolic hybrid systems with discrete interaction. In: EUROMICRO-SEAA. pp. 275–278 (2011)

16. Cimatti, A., Mover, S., Tonetta, S.: Smt-based scenario verification for hybrid systems. Formal Methods in System Design 42(1), 46–66 (2013)
17. Cimatti, A., Mover, S., Tonetta, S.: Quantifier-free encoding of invariants for hybrid systems. Formal Methods in System Design 45(2), 165–188 (2014)
18. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013)
19. Fehnker, A., F.Ivancic: Benchmarks for hybrid systems verification. In: HSCC. pp. 326–341 (2004)
20. Fränzle, M.: What Will Be Eventually True of Polynomial Hybrid Automata? In: TACS. pp. 340–359 (2001)
21. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. STTT 10(3), 263–279 (2008)
22. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: CAV. pp. 379–395 (2011)
23. Ghilardi, S., Ranise, S.: MCMT: A Model Checker Modulo Theories. In: IJCAR. pp. 22–29 (2010)
24. Henzinger, T.A., Ho, P., Wong-Toi, H.: HYTECH: A Model Checker for Hybrid Systems. STTT 1(1-2), 110–122 (1997)
25. Henzinger, T.A., Ho, P.H.: Hytech: The cornell hybrid technology tool. In: Hybrid Systems. pp. 265–293 (1994)
26. Henzinger, T.A.: The theory of hybrid automata. In: LICS. pp. 278–292 (1996)
27. Johnson, T.T., Mitra, S.: A small model theorem for rectangular hybrid automata networks. In: FMOODS/FORTE. pp. 18–34 (2012)
28. Kindermann, R., Junttila, T.A., Niemelä, I.: Beyond Lassos: Complete SMT-Based Bounded Model Checking for Timed Automata. In: FMOODS/FORTE. pp. 84–100 (2012)
29. Kindermann, R., Junttila, T.A., Niemelä, I.: Bounded Model Checking of an MITL Fragment for Timed Automata. In: ACSD. pp. 216–225 (2013)
30. Kindermann, R., Junttila, T.A., Niemelä, I.: Smt-based induction methods for timed systems. In: Formal Modeling and Analysis of Timed Systems - 10th International Conference, FORMATS 2012, London, UK, September 18-20, 2012. Proceedings. pp. 171–187 (2012)
31. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic Reachability Computation for Families of Linear Vector Fields. J. Symb. Comput. 32(3), 231–253 (2001)
32. de Moura, L.M., Owre, S., Rueß, H., Rushby, J.M., Shankar, N., Sorea, M., Tiwari, A.: SAL 2. In: CAV. pp. 496–500 (2004)
33. Mover, S., Cimatti, A., Tiwari, A., Tonetta, S.: Time-aware relational abstractions for hybrid systems. In: EMSOFT. pp. 1–10 (2013)
34. Mover, S.: Verification of Hybrid Systems using Satisfiability Modulo Theories. Ph.D. thesis, University of Trento (2014)
35. Platzer, A., Quesel, J.: KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). In: IJCAR. pp. 171–178 (2008)
36. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Trans. Embedded Comput. Syst. 6(1) (2007)
37. Tiwari, A.: HybridSAL Relational Abstracter. In: CAV. pp. 725–731 (2012)
38. Tonetta, S.: Abstract model checking without computing the abstraction. In: FM. pp. 89–105 (2009)
39. Wang, F.: Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures. IEEE Trans. Software Eng. 31(1), 38–51 (2005)
40. Zutshi, A., Sankaranarayanan, S., Tiwari, A.: Timed Relational Abstractions for Sampled Data Control Systems. In: CAV. pp. 343–361 (2012)