SMT-based verification of Hybrid Systems

Alessandro Cimatti and Sergio Mover and Stefano Tonetta
FBK-irst, I38050, Trento, Italy

Abstract

Hybrid automata networks (HAN) are a powerful formalism to model complex embedded systems. In this paper, we survey the recent advances in the application of Satisfiability Modulo Theories (SMT) to the analysis of HAN. SMT can be seen as an extended form of Boolean satisfiability (SAT), where literals are interpreted with respect to a background theory (e.g. linear arithmetic). HAN can be symbolically represented by means of SMT formulae, and analyzed by generalizing to the case of SMT the traditional model checking algorithms based on SAT.

Introduction

Complex Embedded Systems (CES) consist of software and hardware components that operate autonomous devices interacting with the physical environment. CES are composed of many heterogeneous components, interacting with external environments, and deal with continuous and discrete dynamics. They are increasingly used in many industrial sectors (e.g. automotive, aerospace, consumer electronics, communications, medical and manufacturing), to carry out highly complex and often critical functions.

The lifecycle of CES is also very complex. On the one side, we have the off-line phase, which includes requirements analysis, functional verification, and safety assessment, and that is directed to ensuring that the system will operate correctly once deployed. For example, when designing the control layer for a mobile rover, we may want to carry out some safety assessment (e.g. guarantee that it will be able to operate even in presence of single or even multiple faults), or some diagnosability analysis (e.g. to guarantee that its sensors will be sufficient to detect all unexpected events).

On the other side, we have the operation phase, which may include low-level tasks such as closed-loop control of physical devices, but also higher level activities such as planning, execution monitoring, FDIR (fault detection, isolation, and recovery), and replanning. Each of them is a challenge on its own. For example, the activities of a mobile rover

*This paper was invited as a “What’s Hot” paper to the AAAI’12 Sub-Area Spotlights track.
Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
SMT solvers, such as incrementality, unsatisfiable core extraction, and interpolation. The SMT-based approach has been found particularly effective on networks of linear hybrid automata with large number of components and simple, and often non-deterministic, dynamics. Specialized techniques have been conceived for these cases, taking into account the structure of the network and the specific analyses at hand, e.g. scenario validation.

The paper is structured as follows. We first survey the field of SMT, and discuss SMT-based encoding and verification of generic infinite-state transition systems represented in the SMT framework. Then, we present networks of hybrid automata, their SMT-based encoding, and several SMT-based verification techniques, specialized to the analysis of HANs. Finally, we draw some conclusions and present important directions for future research.

**Satisfiability Modulo Theories**

*Satisfiability Modulo Theory* (SMT) (Barrett et al. 2009) is the problem of deciding the satisfiability of a first-order formula with respect to a decidable background theory $T$.

**Example 1** The formula $x < y \land (x + 3 = z \lor z > y)$ is satisfiable in the theory of Linear Arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$), since $x := 5$, $y := 6$, $z := 8$ is a model for the formula. $\mathcal{LA}(\mathbb{Q})$ interprets $x, y, z$ as rational variables, $3$ as a rational constant and $\lor, =, <, >$ as the corresponding operations and relations over the rationals.

There exist several theories of practical interests: *Equality and Uninterpreted Functions* ($\mathcal{EUF}$), Linear Arithmetic over the reals ($\mathcal{LA}(\mathbb{Q})$) and the integers ($\mathcal{LA}(\mathbb{Z})$), Real Closed Fields, Difference Logic ($\mathcal{DL}$), Bit Vectors ($\mathcal{BV}$) and Arrays. Moreover, under certain assumptions theories $T_1$ and $T_2$ may be combined in the theory $T_1 \cup T_2$. We refer to (Bozzano et al. 2006; de Moura and Bjørner 2008a) for details on theories and approaches on theory combination. In the context of Hybrid System verification, the most used theories are $\mathcal{LA}(\mathbb{Q})$ and Real Closed Fields, since in several cases they are expressive enough to model continuous variables and their evolution.

SMT solvers are tools which implement decision procedures for the SMT problem. The most efficient implementations of SMT solvers use the so-called “lazy approach”, where a SAT solver is tightly integrated with a $T$-solver. The role of the SAT solver is to enumerate the truth assignments to the Boolean abstraction of the first-order formula. The Boolean abstraction has the same Boolean structure of the first-order formula, but “replaces” the predicates which contain $T$ information with fresh Boolean variables. The Boolean abstraction of the Example 1 is $a \land (b \lor c)$, where $a, b, c$ are fresh Boolean variables. The $T$-solver is invoked when the SAT solver finds a model for the Boolean abstraction: the Boolean model maps directly to a conjunction of $T$ atoms, which the $T$-solver can handle. If the conjunction is satisfiable also the original formula is satisfiable. Otherwise the $T$-solver returns a conflict set which identifies a reason for the unsatisfiability. Then, the negation of the conflict set is learned by the SAT solver in order to prune the search. Examples of solvers based on the “lazy approach” are MATHSAT (Bruttomesso et al. 2008), Z3 (de Moura and Bjørner 2008b), YICES (Dutertre and de Moura 2006) and OPLEMSMT (Bruttomesso et al. 2010).

SMT-solvers often construct models in the case a formula is satisfiable and proofs if it is unsatisfiable. Proofs are used to generate additional informations, such as unsatisfiable cores and interpolants. An unsatisfiable core for an unsatisfiable set of clauses is a subset of the clauses which is still unsatisfiable. See (Cimatti, Griggio, and Sebastiani 2011) for a survey. A Craig Interpolant of two formulas $A$ and $B$, with $A \land B$ unsatisfiable, is a formula $I$ such that $A$ implies $I$, $I \land B$ is unsatisfiable and $I$ contains only variables common to both $A$ and $B$. Intuitively, an interpolant is an over-approximation of $A$ “guided” by $B$. See (Cimatti, Griggio, and Sebastiani 2010).

Finally, SMT solvers also feature an incremental interface: they are able to tackle sequences of satisfiability problems efficiently by reusing theory information discovered during the previous searches. Incrementality is exploited to improve the performances of several verification algorithms.

**SMT-based Verification of FOTS**

A first-order transition system (FOTS) symbolically represents an infinite state system using first-order logic formulas. Symbolic representation is well known and is also used in verification tools (Cimatti et al. 2002; Bensalem et al. 2000). A FOTS is a tuple $S = (V, W, I, Z, T)$. $V$ is the set of state variables of the system. A single state of the system is identified by an assignment to all the variables in $V$. $W$ is the “set of input variables” which define the labels of the transitions of the system. $I$ is a first-order formula over variables in $V$ which identifies the set of the initial states of $S$. $Z$ is a first-order formula over variables in $V$ which identifies the set of invariant states of $S$. Intuitively, states that are not in $Z$ are not part of the reachable states of $S$. $T$ is the transition relation of the system. It is a first-order formula over variables in $V$, $W$ and $V'$, which represents the current state of the system, $V$, with the variables $V'$, which represent the state of the system after the transition (i.e. $V' = \{v' | v \in V\}$). Also the input variables $W$ are taken into account in the transition relation. Input variables are assigned when performing a transition and models external inputs from the environment. Paths are concatenations of transitions starting from an initial state. A state $q$ is reachable in $S$ if there is a path in $S$ that ends in $q$.

**Example 2** Consider $S = (V, W, I, Z, T)$, where $V := \{B, x\}$, $W := \{R\}$, $I := B \land x = 0$, $Z := x \leq 10$, $T := (R \rightarrow I) \land (\neg R \rightarrow (B \leftrightarrow \neg B \land x' = x + 1))$. The initial state of the system is $q_0 = (B, x = 0)$. In every state the system non-deterministically performs a transition back to $q_0$, if $R$ is true, or a transition where the value of $B$ is negated and $x$ is incremented by 1. Note that, without the invariant $Z$, the transition system would have an infinite number of reachable states. Fig. 1 shows an explicit representation of the transition system.

**Bounded Model Checking** Bounded Model Checking (BMC) (Biere et al. 1999) is a technique that finds a violation of a property $\phi$ in a transition system $S$. The main
idea of BMC is to explore all the paths of the system $S$ up to a bounded number of steps $k$. Thus, BMC finds a violation for $\phi$ if there is a path which witnesses the violation in at most $k$ steps. Otherwise, BMC certifies that the property $\phi$ is not violated in all the paths of the systems of length $k$.

The set of all paths of $k$ steps and the violation condition of $\phi$ are encoded in a formula, $BMC(k)$. The formula $BMC(k)$ is satisfiable iff there exists a path of length $k$ from the initial state of $S$ to a state where $\phi$ does not hold. For finite state systems $BMC(k)$ is a propositional formula, thus it can be checked using a SAT solver. The BMC approach lifts to infinite state systems, using an SMT solver to check the satisfiability of the first-order formula $BMC(k)$.

Despite its incompleteness, BMC demonstrated its practical usefulness to find bugs for finite and infinite states systems (Audemard et al. 2002; 2005; Sorea 2002).

**K-induction**

K-induction (Sheeran, Singh, and Stälmarck 2000) is a technique that proves that if a set of states is not reachable in $k$ steps, then it is not reachable at all. On the lines of the induction principle, it consists of a base step, which solves the bounded reachability problem with a given bound $k$, and an inductive step, which concludes that $k$ is sufficient to solve the (unbounded) reachability problem. Both cases are reduced to checking the unsatisfiability of first-order formulas.

For finite-state systems there exists a bound $k$ which ensures the termination of k-induction. However, typically this is not the case for infinite state systems. A possible approach (de Moura, Rueß, and Sorea 2003) consists of strengthening the inductive condition. Another viable approach is to integrate abstraction techniques with k-induction (Tonetta 2009).

**Interpolation-based model checking**

The main idea of interpolation-based model checking (McMillan 2003) is to partition an unsatisfiable BMC encoding in two formulas, a prefix and a suffix which share only a single time frame. Since the encoding is unsatisfiable, the algorithm computes the interpolant of the prefix and the suffix: the interpolant is an overapproximation of the states reachable with the prefix and is inconsistent with the suffix. The operation is iterated to compute an overapproximation of the reachable states. A different algorithm (Vizel and Grumberg 2009) exploits the notion of interpolation sequence, which lead to the creation of different overapproximations. Interpolation-based model checking can be applied to infinite state systems (McMillan 2005). However, termination is not guaranteed.

**Abstraction**

Abstraction has been widely used in the analysis of infinite-state systems. Predicate abstraction (Graf and Säidi 1997) computes a finite-state abstraction of a system, which can be analyzed using finite state verification techniques. SMT solvers have been used for the computation of predicate abstraction (Lahiri, Nieuwenhuis, and Olivas 2006). An alternative approach integrates Binary Decision Diagrams (BDDs) with SMT techniques to compute the abstraction (Cimatti et al. 2010).

**Hybrid Automata Network**

**Hybrid Automata** (Henzinger 1996) (HA) are a well known framework used in formal verification to model the discrete and the continuous evolution of hybrid systems.

HA extends a finite state machine (FSM) with continuous components, modeled using real-valued variables. Each state of the FSM, called location, defines the so called flow conditions, which describe with differential equations the evolution of the continuous variables over time. For example, the flow condition $\dot{x} = v, \dot{v} = a, \dot{a} = 2$ describes an uniformly accelerated motion. Each location also defines invariant conditions over continuous variables, which must be satisfied whenever the system is inside the location. The discrete transitions of a HA defines how the system changes the current location. Each transitions is associated to a jump condition: it is a formula over the continuous variables at the current and at the next step, which describes the guards and the effects of the transition. Moreover, transitions are labeled with a symbol to enable synchronizations among automata. We call $\Sigma$ the set of events which label transitions.

A state of the HA is a tuple $(q_i, s_i)$, where $q_i$ is a location and $s_i$ is an assignment to the continuous variables. A run $\sigma = \langle q_0, s_0 \rangle \xrightarrow{a_1} \ldots \xrightarrow{a_n} \langle q_n, s_n \rangle$ of a HA is a sequence of states that the HA can visit. The first state satisfies the initial condition of the HA, while all the states satisfy the invariant condition of the corresponding location. Each state $\langle q_i, s_i \rangle$ moves to the next state $\langle q_{i+1}, s_{i+1} \rangle$ either with a discrete or a continuous transition, depending on the event $a_i$: if $a_i \in \Sigma$ then there is a discrete transition, otherwise $a_i \geq 0$ means that there is a continuous transition where the time elapsed is equal to $a_i$. A discrete transition is fired if its jump condition is satisfied. A continuous transition updates the continuous variables according to the flow condition in the current location and the amount of time elapsed, while it keeps the location unchanged.

**Example 3**

Figure 2 shows the HA for a rod component in a nuclear reactor. A possible run of the automaton is $\sigma = \langle \text{Ready}, x = 0 \rangle \xrightarrow{3} \langle \text{Ready}, x = 3 \rangle \xrightarrow{\text{Add}} \langle \text{In}, x = 0 \rangle \xrightarrow{2} \langle \text{In}, x = 3 \rangle \xrightarrow{\text{Remove}} \langle \text{Recover}, x = 0 \rangle \xrightarrow{16} \langle \text{Recover}, x = 16 \rangle \xrightarrow{22} \langle \text{Ready}, x = 16 \rangle$.

A Linear HA (LHA) is an HA where all the conditions are Boolean combinations of linear inequalities and the flow conditions contain variables in $X$ only.

Real hybrid systems are constituted by several components. A Network of Hybrid Automata $\mathcal{H} = H_1 \parallel \ldots \parallel H_n$ is a set of HA. Each automaton $H_i$ of the network moves asynchronously with transitions labeled with a local event.
(i.e. a symbol which is only in the alphabet \( \Sigma_i \)). Instead, automata synchronize on transitions labeled with a shared event (i.e. a symbol which is contained in the alphabet of more automata). Synchronization is used to model the communication between the automata of the network. There exist other formalisms to compose HAs such as Hybrid I/O Automata (Lynch, Segala, and Vaandrager 2003).

**SMT-based Verification of HAN**

**SMT-based Encoding** The SMT-based verification of a Hybrid Automata Network \( \mathcal{H} \) is enabled by encoding \( \mathcal{H} \) in a FOTS \( \mathcal{N} \). Each automaton \( H_i \) of \( \mathcal{H} \) is translated into an equivalent FOTS \( S_i \). Then, the network \( \mathcal{H} \) is encoded in \( \mathcal{S} \), which adds the synchronization constraints imposed by \( \mathcal{H} \). In the following, we describe the encoding of a network of linear hybrid automata with disjoint sets of continuous variables.

The current location of \( H_i \) is encoded with a set of Boolean variables while the continuous variables are encoded with real-valued variables. The initial states and the invariants of \( H_i \) are encoded into the first-order formulas \( I_i \) and \( Z_i \). An additional discrete variable \( \varepsilon_i \) encodes the transition events \( \Sigma \cup \{ T, S \} \), where \( T \) and \( S \) are two fresh events. The first determines when \( S_i \) performs a continuous evolution, while the latter determines when \( S_i \) stutters (i.e. when \( S_i \) does not move). The first-order formula \( T_i \) encodes that the value of \( \varepsilon_i \) is chosen non-deterministically and that \( \varepsilon_i = a \) implies the encoding of the discrete transition labelled with \( a \) in \( H_i \).

In a continuous transition, \( \text{loc} \) does not change, while the continuous variables change according to the flow conditions. A real variable \( \delta_i \) encodes the amount of time elapsed. A constraint \( \delta_i \geq 0 \) ensures that such amount is positive. Since the flow conditions are linear (i.e. of the form \( \sum a_j \cdot x_j \geq b \), where \( a_j, b \in \mathbb{R}, x_j \in X, b \geq 0 \)) the evolution of the continuous variables \( x_j \) is encoded with the constraint \( \varepsilon_i = T \rightarrow \sum a_j \cdot (x'_j - x_j) \geq b \cdot \delta_i \). Note that we are assuming that the invariants \( Z_i \) are convex so that they hold along the continuous transition.

The FOTS \( \mathcal{N} = (V, W, I, Z, T) \) which encodes \( \mathcal{H} \) is defined as \( V = \bigcup_i V_i \); \( W = \bigcup_i W_i \); \( I = \bigwedge_i I_i \); \( Z = \bigwedge_i Z_i \); \( T = \bigwedge_i T_i \wedge \text{SYNC} \), where \( \text{SYNC} \) is a synchronization constraint. For every couple of FOTS \( S_i, S_j \) and for every shared event \( a \) of \( S_i \) and \( S_j \), \( \text{SYNC} \) forces that every time \( S_i \) moves with a transition labelled with \( a \), also \( S_j \) moves with a transition labelled with \( a \). Moreover, note that the event \( T \) is common to all the FOTS.

The encoding presented so far follows the standard “global time” semantics of HAN (Henzinger 1996). Several verification algorithms are enabled by a different semantics, called “local time” semantics (Bengtsson et al. 1998). In the “local time” semantics the \( T \) event is a local event. This means that the real time in two FOTS evolves independently. Each FOTS \( S_i \) stores the amount of time elapsed from the beginning of the run in a continuous variable \( \text{TIME}_i \). The consistency of the network is ensured by a modified version of the \( \text{SYNC} \) constraints: when \( H_i \) and \( H_j \) synchronize on the same event \( a \), their local times, \( \text{TIME}_i \) and \( \text{TIME}_j \) must be the same. Moreover, \( \text{TIME} \) and \( \text{TIME} \) must be the same at the end of a run.

**Verification Techniques** BMC encodings for Timed Automata, a restricted class of hybrid automata useful to represent real-time systems, have been presented in (Audemard et al. 2002; Sorea 2002).

(Audemard et al. 2005) generalizes the encoding to Linear Hybrid Automata. The approach is similar to the one outlined in Section . (Abráham et al. 2005) focus on optimizations of the BMC encoding. The authors of (Bu et al. 2010) exploit the “local time” semantics in their BMC encoding: traces of the system are obtained by composing traces of the local automata, and superimposing compatibility constraints resulting from the synchronizations.

BMC has also been extended to classes of hybrid automata more expressive than LHA. The authors of (Franzle and Herde 2007) implements a SMT-solver based on interval constraint propagation, rather than on the “lazy” approach. The works in (Eggers et al. 2011; Ishii, Ueda, and Hosobe 2011) extend the SMT framework to deal natively with Ordinary Differential Equations (ODEs).

K-induction has been applied to the verification of safety properties for real-time and hybrid systems. Real-time systems are verified using k-induction with a manual strengthening in (Steiner and Duterre 2010). Both the automatic strengthening of k-induction (de Moura, Rueß, and Sorea 2003) and the abstract k-induction approach (Tonetta 2009) have been tested on LHA benchmarks.

The technique presented in (Cimatti et al. 2009) computes a predicate abstraction for a network of Hybrid Automata exploiting the structure of the HAN.

**Scenario Verification** In order to support user validation, it is very important to check whether a HAN may exhibit behaviors that satisfy a certain scenario, specifying some desired or undesired interactions among the components. The scenario-based verification problem consists of checking if a network of hybrid automata accepts some desired interactions among the components. In that case, we say that the scenario is feasible.

A basic language to express such scenarios of interaction is Message Sequence Charts (MSCs). MSCs are especially useful for the end users because of their clarity and graphical content. An MSC defines a single (partial-order) interaction of the components of a network \( \mathcal{H} = H_1 || \ldots || H_n \). For each hybrid automaton \( H_i \), the MSC defines a sequence

![Figure 2: Rod component of a nuclear reactor.](image-url)
of shared events \( a_1; \ldots; a_n \), called instance. An MSC is the parallel composition of instances \( \sigma_1, \ldots, \sigma_n \). Figure 3 shows an example of MSC for the HAN described in (Henzinger 1996). An MSC is feasible in a network \( \mathcal{H} \) if there exist a run of \( \mathcal{H} \) such that, for all \( \mathcal{H}_i \), the sequence of shared events obtained projecting the run only on the shared events of \( \mathcal{H}_i \) is equal to \( \sigma_i \). Roughly speaking, the components of \( \mathcal{H} \) must synchronize following the same sequence of events described in the MSC. MSCs have been extended in several ways. A particular variant is Constrained MSC (CMSC) (Cimatti, Mover, and Tonetta 2011b), which enriches a MSC with first-order formulas over the variables of \( \mathcal{H} \). All the variables used in the constraints refer to a specific occurrence of an event in the MSC. These constraints turn out to be very useful and easy to handle with the SMT-based approach.

The classical approach to scenario-verification is based on the construction of a monitor that, composed with the network \( \mathcal{N} \), forces \( \mathcal{N} \) to follow only paths that satisfy the MSC. It is in spirit similar to the automata-approach to LTL model checking (Vardi 1995). The SMT-based verification techniques are applied off the shelf on the resulting FOTS. The monitor can be an additional component in the network or consist of many components, one for each instance of the CMSC. Exploiting local-time semantics, the monitor can also be reduced to follow one interleaving of the partial-order reduction defined by the CMSC.

The automata-based approach turns out to be inefficient (Cimatti, Mover, and Tonetta 2011a), since BMC unravels the system for several steps before finding a run which witnesses the feasibility of the MSC. The approaches presented in (Cimatti, Mover, and Tonetta 2011a; 2011b) directly encode the feasibility problem exploiting the structure of the MSC, rather than using an observer.

The approach in (Cimatti, Mover, and Tonetta 2011a) is a specialized BMC encoding which focuses on finding if a MSC is feasible in \( \mathcal{H} \). For each automaton \( \mathcal{H}_i \), we encode the set of paths that are consistent with the instance \( \sigma_i \). In the encoding, the position of the transitions labeled with a shared event is fixed a-priori. For example, we encode a shared events every \( k \) steps in the encoding. All the steps between two shared events encode a local event. In this way, the formulas are much simpler, since there is less non-determinism in the choice of the events to be performed.

The local encoding is enabled by adopting the “local time” semantics. Recall that in the “local time” semantics the continuous transition is a local event of each transition system \( S_i \). The synchronization constraints are then imposed between the different local encodings, to ensure that synchronizations happen at the same real time. Note that also the position of the synchronization constraints is fixed, resulting in a simplified encoding. The solver is fed with a sequence of problems with an increasing number of local transitions between two shared events. Since the formula that encodes the shared events and the synchronization constraints does not change from one problem to the other, we exploit the incrementality of the SMT-solver.

In order to prove that a scenario is unfeasible, we extended this approach with k-induction (Cimatti, Mover, and Tonetta 2011b). The base case of k-induction proves that the scenario is unfeasible in a given number of steps. The inductive case proves, for every sequence of local transition, that the system cannot reach new states. The approach also exploits the integration of k-induction with predicate abstraction (Tonetta 2009). Finally, unsat core and interpolation are used to provide the user with explanations that help in identifying the reasons of the unfeasibility.

**Conclusions and Future Directions**

In this paper, we have presented an overview of the recent applications of SMT for the formal verification of Hybrid Automata Networks. Directions of future work include the verification of systems with complex, nonlinear dynamics, and increasing scalability by means of compositional and hierarchical reasoning. For lack of space, we have disregarded SMT-based approaches to other important functions, such as safety assessment, planning, monitoring, and diagnosis. Although some of these challenges have been addressed in the discrete case by means of SAT-based approaches (Rintanen 2011), a full generalization in the case of SMT is the subject of ongoing research (see for instance (Gregory et al. 2012)).

**Acknowledgements**

This work benefits from the work of and many discussions with Marco Roveri, Paolo Traverso, Andrea Micheli, Luca Bonetti, Gianni Zampedri, Roberto Cavada, Marco Bozzano, Alberto Griggio, Iman Narasamdy, Andrei Tchal'tsev.

**References**


Steiner, W., and Dutertre, B. 2010. SMT-Based Formal Verification of a TTEthernet Synchronization Function. In FMICS, 148–163.
