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 must be planned according to strict timing and resource requirements, and the information conveyed by the sensors can be used to diagnose whether any (or which) fault has occurred.

The off-line and operation phases are tightly intertwined: for instance, the description of the rover used to analyzed the control algorithms during design could be used for planning or for diagnosis in operation. In order to avoid gaps between the two phases, it is important to ensure consistency between the models of the system used off-line and in operation, and to be able to interpret the actual executions on the models.

Model checking (Clarke, Grumberg, and Peled 2001) provides a comprehensive formal framework (and also a technological basis) for such tasks (Cimatti, Pecheur, and Cavada 2003; Bozzano et al. 2011). Symbolic model checking (McMillan 1993) is limited to a Boolean representation case, and leverages on Boolean inference engines such as Binary Decision Diagrams (Bryant 1986) and, more recently, SAT solvers (Biere et al. 1999). Unfortunately, a Boolean representation is often insufficient to represent many important features of CES, such as activities with duration, and resource constraints and consumption.

Networks of communicating *Hybrid Automata* (HAs) (Henzinger 1996) are increasingly used as a formal framework to model discrete and continuous components and their interaction: local activities of each component amount to transitions local to each hybrid automaton; communications and other events that are shared between/visible for various components are modeled as synchronizing transitions of the automata in the network; the elapse of time is modeled as shared, delay transition.

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).

In terms of expressiveness, the SMT language provides a natural symbolic representation for the HAN: in addition to the Boolean part, used to represent the discrete parts of the automata, the theory description provides for a representation of the evolution of timed and continuous variables. In terms of the verification techniques, the approach allows for a direct extension of the SAT-based model checking algorithms, fully leveraging the advanced features of modern

*This paper was invited as a "What's Hot" paper to the AAAI'12 Sub-Area Spotlights track.

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 $\mathcal{T}$.

Example 1 *The formula* $x < y \wedge (x + 3 = z \vee z > y)$ *is satisfiable in the theory of Linear Arithmetic on 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* $+, =, <, >$ *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 $\mathcal{T}_1$ and $\mathcal{T}_2$ may be combined in the theory $\mathcal{T}_1 \cup \mathcal{T}_2$. 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 $\mathcal{T}$-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 $\mathcal{T}$ information with fresh Boolean variables. The Boolean abstraction of the Example 1 is $a \wedge (b \vee c)$, where a, b, c are fresh Boolean variables. The $\mathcal{T}$-solver is invoked when the SAT solver finds a model for the Boolean abstraction: the Boolean model maps directly to a conjunction of $\mathcal{T}$ atoms, which the $\mathcal{T}$-solver can handle. If the conjunction is satisfiable also the original formula is satisfiable. Otherwise the $\mathcal{T}$-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 OPEMSMT (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 $A \wedge B$ unsatisfiable, is a formula I such that A implies I, $I \wedge B$ 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 $S = \langle V, \mathcal{W}, I, Z, T \rangle$. 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. $\mathcal{W}$ 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, $\mathcal{W}$ and V'. T relates the variables which represent 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 $\mathcal{W}$ 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* $S = \langle V, \mathcal{W}, I, Z, T \rangle$*, where* $V := \{B, x\}$*,* $\mathcal{W} := \{R\}$*,* $I := B \wedge x = 0$*,* $Z := x \leq 10$*,* $T := (R \rightarrow I) \wedge (\neg R \rightarrow (B \leftrightarrow \neg B \wedge x' = x + 1))$*. The initial state of the system is* $q_0 = \langle B, x = 0 \rangle$*. 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 ϕ in a transition system S. The main

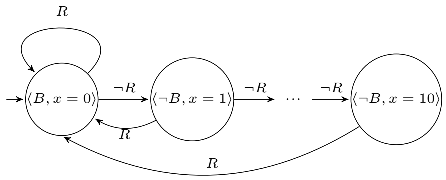

Figure 1: Explicit state representation of S from Example 2

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 ϕ if there is a path which witnesses the violation in at most k steps. Otherwise, BMC certifies that the property ϕ 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 ϕ 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 ϕ 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 Stålmarck 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 Saïdi 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 Oliveras 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 Σ the set of events which label transitions.

A state of the HA is a tuple $\langle q_i, s_i\rangle$, where q_i is a location and s_i is an assignment to the continuous variables. A run $\sigma = \langle q_0, s_0\rangle \xrightarrow{a_1} \dots \xrightarrow{a_n} \langle q_n, s_n\rangle$ 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* $\sigma = \langle Ready, x=0\rangle \xrightarrow{3} \langle Ready, x=3\rangle \xrightarrow{Add} \langle In, x=0\rangle \xrightarrow{3} \langle In, x=3\rangle \xrightarrow{Remove} \langle Recover, x=0\rangle \xrightarrow{16} \langle Recover, x=16\rangle \xrightarrow{\tau} \langle Ready, x=16\rangle$.

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

Real hybrid systems are constituted by several components. A *Network of Hybrid Automata* $\mathcal{H} = H_1 \parallel \dots \parallel H_n$ is a set of HA. Each automaton H_i of the network moves asynchronously with transitions labeled with a local event

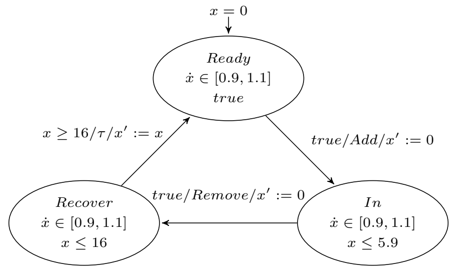

Figure 2: Rod component of a nuclear reactor.

(i.e. a symbol which is only in the alphabet Σ_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 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 ε_i encodes the transition events $\Sigma \cup \{\text{T}, \text{S}\}$, 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 ε_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, loc_i does not change, while the continuous variables change according to the *flow conditions*. A real variable δ_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 $\sum a_j \cdot \dot{x_j} \bowtie b$, where $a_j, b \in \mathbb{R}, x_j \in X, \bowtie \in \{<, \leq, =\}$) the evolution of the continuous variables x_j is encoded with the constraint $\varepsilon_i = \text{T} \rightarrow \sum a_j \cdot (x'_j - x_j) \bowtie b \cdot \delta_i$. Note that we are assuming that the invariants Z_i are convex so that they hold along the continuous transition.

The FOTS $\mathcal{N} = \langle V, W, I, Z, T \rangle$ 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 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, $SYNC$ 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 TIME_i. The consistency of the network is ensured by a modified version of the SYNC constraints: when H_i and H_j synchronize on the same event a, their local times, TIME_i and TIME_j must be the same. Moreover, TIME_i and TIME_j 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 . (Ábrahám 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 (Fränzle 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 Dutertre 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

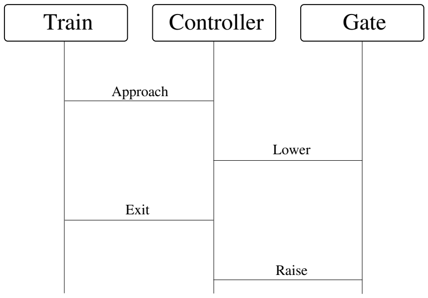

Figure 3: An MSC for the Train-Gate-Controller.

of shared events $a_1; \ldots ; a_l$, called *instance*. An MSC is the parallel composition of instances $\sigma_1 \parallel \ldots \parallel \sigma_n$. Figure 3 shows an example of MSC for the HAN described in (Henzinger 1996). An MSC is feasible in a network $\mathcal{H}$ iff there exist a run of $\mathcal{H}$ such that, for all H_i, the sequence of shared events obtained projecting the run only on the shared events of H_i is equal to σ_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 unrolls 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 H_i, we encode the set of paths that are consistent with the instance σ_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 Narasamdya, Andrei Tchaltsev.

References

Ábrahám, E.; Becker, B.; Klaedtke, F.; and Steffen, M. 2005. Optimizing bounded model checking for linear Hybrid Systems. In *VMCAI*, 396–412.

Audemard, G.; Cimatti, A.; Kornilowicz, A.; and Sebastiani, R. 2002. Bounded Model Checking for Timed Systems. In *FORTE*, 243–259.

Audemard, G.; Bozzano, M.; Cimatti, A.; and Sebastiani, R. 2005. Verifying industrial hybrid systems with MathSAT. *ENTCS* 119(2):17–32.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*. 825–885.

Bengtsson, J.; Jonsson, B.; Lilius, J.; and Yi, W. 1998. Partial order reductions for timed systems. In *CONCUR*, 485–500.

Bensalem, S.; Ganesh, V.; Lakhnech, Y.; Muoz, C.; Owre, S.; Rue, H.; Rushby, J.; Rusu, V.; Sadi, H.; Shankar, N.; Singerman, E.; and Tiwari, A. 2000. An Overview of SAL. In *LFM*, 187–196.

Biere, A.; Cimatti, A.; Clarke, E. M.; and Zhu, Y. 1999. Symbolic Model Checking without BDDs. In *TACAS*, 193–207.

Bozzano, M.; Bruttomesso, R.; Cimatti, A.; Junttila, T. A.; Ranise, S.; van Rossum, P.; and Sebastiani, R. 2006. Efficient theory combination via Boolean search. *Inf. Comput.* 204(10):1493–1525.

Bozzano, M.; Cimatti, A.; Roveri, M.; and Tchaltsev, A. 2011. A comprehensive approach to on-board autonomy verification and validation. In Walsh, T., ed., *Proceedings of IJCAI-11*, 2398–2403.

Bruttomesso, R.; Cimatti, A.; Franzén, A.; and Sebastiani, A. G. R. 2008. The MathSAT4 SMT solver. In *CAV*, 299–303. Springer.

Bruttomesso, R.; Pek, E.; Sharygina, N.; and Tsitovich, A. 2010. The OpenSMT solver. In *TACAS*, 150–153.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comp.* 35(8):677–691.

Bu, L.; Cimatti, A.; Li, X.; Mover, S.; and Tonetta, S. 2010. Model checking of Hybrid Systems using shallow synchronization. In *FORTE*, 155–169.

Cimatti, A.; Clarke, E.; Giunchiglia, E.; Giunchiglia, F.; Pistore, M.; Roveri, M.; Sebastiani, R.; and Tacchella, A. 2002. NuSMV 2: an opensource tool for symbolic model checking. In *CAV*, 359–364.

Cimatti, A.; Dubrovin, J.; Junttila, T. A.; and Roveri, M. 2009. Structure-aware computation of Predicate Abstraction. In *FMCAD*, 9–16.

Cimatti, A.; Franzén, A.; Griggio, A.; Kalyanasundaram, K.; and Roveri, M. 2010. Tighter integration of BDDs and SMT for Predicate Abstraction. In *DATE*, 1707–1712.

Cimatti, A.; Griggio, A.; and Sebastiani, R. 2010. Efficient generation of craig interpolants in Satisfiability Modulo Theories. *ACM Trans. Comput. Log.* 12(1):7.

Cimatti, A.; Griggio, A.; and Sebastiani, R. 2011. Computing small unsatisfiable cores in Satisfiability Modulo Theories. *J. Artif. Intell. Res. (JAIR)* 40:701–728.

Cimatti, A.; Mover, S.; and Tonetta, S. 2011a. Efficient scenario verification for Hybrid Automata. In *CAV*, 317–332.

Cimatti, A.; Mover, S.; and Tonetta, S. 2011b. Proving and explaining the unfeasibility of Message Sequence Charts for Hybrid Systems. In *FMCAD*.

Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal Verification of Diagnosability via Symbolic Model Checking. In *IJCAI*, 363–369. Morgan Kaufmann.

Clarke, E. M.; Grumberg, O.; and Peled, D. 2001. *Model checking*. MIT Press.

de Moura, L. M., and Bjørner, N. 2008a. Model-based Theory Combination. *Electr. Notes Theor. Comput. Sci.* 198(2):37–49.

de Moura, L. M., and Bjørner, N. 2008b. Z3: An efficient SMT solver. In *TACAS*, 337–340.

de Moura, L.; Rueß, H.; and Sorea, M. 2003. Bounded Model Checking and induction: from refutation to verification. In *CAV*, 14–26.

Dutertre, B., and de Moura, L. M. 2006. A fast Linear-Arithmetic solver for DPLL(T). In *CAV*, 81–94.

Eggers, A.; Ramdani, N.; Nedialkov, N.; and Fränzle, M. 2011. Improving SAT Modulo ODE for Hybrid Systems analysis by combining different enclosure methods. In *SEFM*, 172–187.

Fränzle, M., and Herde, C. 2007. HySAT: An efficient proof engine for Bounded Model Checking of Hybrid Systems. *Formal Methods in System Design* 30(3):179–198.

Graf, S., and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In *CAV*, 72–83.

Gregory, P.; Fox, M.; Long, D.; and Beck, J. 2012. Planning Modulo Theories: Extending the planning paradigm. In *Proceedings of ICAPS*. AAAI.

Henzinger, T. A. 1996. The theory of Hybrid Automata. In *LICS*, 278–292. IEEE CS.

Ishii, D.; Ueda, K.; and Hosobe, H. 2011. An interval-based SAT modulo ODE solver for model checking nonlinear Hybrid Systems. *STTT* 13(5):449–461.

Lahiri, S. K.; Nieuwenhuis, R.; and Oliveras, A. 2006. SMT techniques for fast Predicate Abstraction. In *CAV*, 424–437.

Lynch, N. A.; Segala, R.; and Vaandrager, F. W. 2003. Hybrid I/O Automata. *Inf. Comput.* 185(1):105–157.

McMillan, K. L. 1993. *Symbolic model checking*. Kluwer.

McMillan, K. L. 2003. Interpolation and SAT-Based Model Checking. In *CAV*, 1–13.

McMillan, K. L. 2005. Applications of craig interpolants in model checking. In *TACAS*, 1–12.

Rintanen, J. 2011. Planning with Specialized SAT Solvers. In *AAAI*. AAAI Press.

Sheeran, M.; Singh, S.; and Stålmarck, G. 2000. Checking safety properties using induction and a SAT-solver. In *FMCAD*, 108–125.

Sorea, M. 2002. Bounded Model Checking for Timed Automata. *Electr. Notes Theor. Comput. Sci.* 68(5):116–134.

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

Tonetta, S. 2009. Abstract model checking without computing the abstraction. In *FM*, 89–105.

Vardi, M. 1995. An automata-theoretic approach to Linear Temporal Logic. In *Banff Higher Order Works.*, 238–266.

Vizel, Y., and Grumberg, O. 2009. Interpolation-sequence based model checking. In *FMCAD*, 1–8.