Quantified Bounded Model Checking for Rectangular Hybrid Automata

Luan Viet Nguyen, University of Texas at Arlington
Djordje Maksimovic, University of Toronto
Taylor T. Johnson, University of Texas at Arlington
Andreas Veneris, University of Toronto

***Abstract*—Satisfiability Modulo Theories (SMT) solvers have been successfully applied to solve many problems in formal verification such as bounded model checking (BMC) for many classes of systems from integrated circuits to cyber-physical systems (CPS). Typically, BMC is performed by checking satisfiability of a possibly long, but quantifier-free formula. However, BMC problems can naturally be encoded as quantified formulas over the number of BMC steps. In this approach, we then use decision procedures supporting quantifiers to check satisfiability of these quantified formulas. This approach has previously been applied to perform BMC using a Quantified Boolean Formula (QBF) encoding for purely discrete systems, and then discharges the QBF checks using QBF solvers. In this paper, we present a new quantified encoding of BMC for rectangular hybrid automata (RHA), which requires using more general logics due to the real (dense) time and real-valued state variables modeling continuous states. We have implemented a preliminary experimental prototype of the method using the HyST model transformation tool to generate the quantified BMC (QBMC) queries for the Z3 SMT solver. We describe experimental results on several timed and hybrid automata benchmarks, such as the Fischer and Lynch-Shavit mutual exclusion algorithms. We compare our approach to quantifier-free BMC approaches, such as those in the dReach tool that uses the dReal SMT solver, and the HyComp tool built on top of nuXmv that uses the MathSAT SMT solver. Based on our promising experimental results, QBMC may in the future be an effective analysis approach for RHA as further improvements are made in quantifier handling in SMT solvers such as Z3.**

***Index Terms*—bounded model checking, hybrid automata, timed automata, satisfiability modulo theories**

I. Introduction

Boolean Satisfiability (SAT) is the canonical NP-complete problem and is to determine if a given Boolean formula is satisfiable, i.e., check if there exists an assignment of values to variables where the formula is true. A Boolean formula is given in Conjunctive Normal Form (CNF), that is, a conjunction of clauses, each of which is a disjunction of literals. Satisfiability modulo theories (SMT) is a generalization of SAT, where literals are interpreted with respect to a background theory (e.g., linear real arithmetic, nonlinear integer arithmetic, bit-vectors, etc.).

Recently, SMT-based techniques have been developed to formally verify hybrid systems [1]–[6]. Typically, these SMT-based methods are used in bounded model checking (BMC), which is to check for a transition system A and a specification P whether $I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge (\bigvee_{i=0}^{k} P(s_i))$ is satisfiable. Here, $I(s_0)$ encodes an initial set of states over a set of variables s_0, $T(s_i, s_{i+1})$ represents the transition relation from iteration i to $i+1$ over sets of variables s_i and s_{i+1}, and $P(s_i)$ encodes the specification at step i.

Hybrid automata are a modeling formalism used to verify dynamical systems including both continuous states and dynamics as well as discrete states and transitions. Examples of systems naturally modeled by hybrid automata arise in the interaction of physical plants and software controllers in real-time systems and cyber-physical systems (CPS). In essence, hybrid automata augment finite state machines with a set of real-valued variables that evolve continuously over intervals of real time. In hybrid automata, a transition relation $T = D \cup \mathcal{T}$ encodes both discrete transitions D and continuous trajectories $\mathcal{T}$ over intervals of real-time. Rectangular hybrid automata (RHA) are a special class of hybrid automata with continuous dynamics described by rectangular differential inclusions and where all other quantities (guard conditions, invariants, resets, etc.) of the automata are linear inequalities over constants [2], [7]. Sets of states, as well as discrete transitions and continuous trajectories of RHA, can be symbolically represented by SMT formulas over real and Boolean variables.

Depending on the underlying logics supported, SMT solvers may or may not support quantifiers. While quantifiers make the language more expressive, they increase the complexity of computations like checking satisfiability and may also lead to undecidability. Techniques allowing quantifiers, such as in quantified Boolean formula (QBF) solvers, have been developed for BMC of purely discrete systems, such as finite state machines [8], [9]. However, to the best of our knowledge, there has been no effort to develop quantified BMC (QBMC) methods for timed or hybrid automata, which we develop in this paper. Of course, this is partially because the underlying SMT solver requires support for complex combination theories and efficient algorithms to check quantified formulas, which until recently, were either not available or not scalable.

The logic used requires some finite sort for the discrete states (such as a enumerated type or bitvectors) and reals for the continuous states and trajectories. In this paper, we use LRABV (linear real arithmetic with bit-vectors) for encoding QBMC for timed automata and RHA, and we note that general hybrid automata would need NRABV (nonlinear real arithmetic with bit-vectors) or beyond, such as those whose solutions involve special (transcendental) functions like sin, cos, exp, etc. While none of these logics are officially supported in the SMT-LIB2 standard (nor the 2.5 draft) as of the time of this writing [10], several solvers do have unofficial support for this combination theory, such as the latest versions of Z3, which is the SMT solver used in this paper [11].

Related Work: When defining the semantics of hybrid automata, first-order or higher logic is typically used and quantifiers typically show up in several places. Existential quantifiers over reals are used to specify that some amount of real time may elapse in a given location of the hybrid automaton. Universal quantifiers over reals representing real time are used to construct invariants that are enforced at all times, while in a given location of the hybrid automaton; otherwise real time is not allowed to advance, and a discrete transition must be taken, if any are enabled based on the current state and guards of the transitions. Alternative approaches to the one described in this paper have previously been developed, where the universal quantifiers used to define invariants' semantics are explicitly removed from the SMT expressions to create quantifier-free formulas. This allows the use of existing SMT-based procedures and avoids quantifier-elimination and other quantifier-handling procedures [2], [3], [12]. We note that this approach does not use quantifiers on the number of steps $k \geq 0$ in the BMC computation. which we do in this paper. Specifically, we suggest that effectively encoding the BMC problem in a quantified form over the number of steps k may provide a more scalable approach in the future as quantifier handling procedures are improved in the underlying solvers. We accomplish this by extending existing results for BMC of discrete systems with QBF solvers [9] to timed and hybrid automata, specifically RHA.

Typical approaches to analyze timed and hybrid automata use symbolic representations of states such as difference bound matrices (DBMs) to represent clock regions in Uppaal [13] or polyhedra in HyTech [14]. Several other formal verification tools for hybrid automata focus on performing reachability computations, and overapproximate the set of reachable states using various data structures to symbolically represent geometric sets of states, such as Taylor models in Flow* [15] and support functions in SpaceEx [16]. Reachability analysis tools like Flow* and SpaceEx focus on computing reachable states, although there is a direct equivalence between time-bounded reachability computations and BMC.

Several SMT-based approaches can verify properties of timed and hybrid automata. dReal is an SMT-solver for first-order logic formulas over the reals, and uses a δ-complete decision procedure [17]. dReach is a BMC tool that queries dReal to check satisfiability of SMT formulas encoding the transitions and trajectories for hybrid automata [4]. HyComp is a verification tool for networks (parallel compositions) of hybrid automata with polynomial and other dynamics [6] and is built on top of nuXmv [18]. For k-induction and IC3, HyComp may perform unbounded model checking, but in the BMC mode, it also allows a limit on the number of steps, and also encodes the semantics of the network of hybrid automata's transition relation and trajectories. A very closely related approach to this paper also encodes BMC problems for timed automata using quantified formulas, but this quantification is to encode unknown or incomplete components, and is not a quantification over the BMC length [19]. Passel is a parameterized verification tool for networks of RHA that may prove properties regardless of the number N of automata in the network [2]. Passel implements an extension to hybrid automata of the invisible invariants approach for parameterized verification, and consists of an invariant synthesis procedure [20] that relies on reachability computations [5]. Passel encodes the semantics of networks of hybrid automata as SMT formulas and checks satisfiability and validity using the Z3 SMT solver. Additionally, when performing reachability computations, Passel makes use of quantifier elimination procedures over the reals and bit-vectors [5].

Contributions: In this paper, we present a new SMT-based verification technique that encodes the BMC problem for RHA in a quantified form, which we call quantified BMC (QBMC). We take hybrid automata in the SpaceEx format [16], which are then translated to the QBMC encoding proposed in this paper using the HyST model transformation tool [21]. We then perform QBMC by querying the Z3 SMT solver via its Python API and use its quantifier-handling procedures [11]. We present preliminary experimental results where the QBMC approach and Z3 perform competitively, when compared to (a) the dReach tool that performs BMC using an SMT check by querying the dReal δ-decidable SMT solver [4], [17], and (b) the HyComp tool built on top of nuXmv that uses the MathSAT SMT solver [22]. The examples include standard ones such as Fischer and Lynch-Shavit mutual exclusion, as well as an illustrative example to describe the encoding. The main contribution of this paper is the first encoding of BMC as a quantified problem for RHA. Our results subsume the case for timed automata, as RHA are more expressive than timed automata, and we note this is also the first QBMC approach for timed automata.

II. Hybrid Automata Syntax and Semantics

A hybrid automaton is essentially a finite state machine extended with a set of real-valued variables that evolve continuously over intervals of real-time.

Syntax: The syntactic structure of a hybrid automaton is formally defined as follows.

Definition 1: A hybrid automaton $\mathcal{H}$ is a tuple, $\mathcal{H} \triangleq \langle Loc, Var, Inv, Flow, Trans, Init \rangle$, with the components as follows. (a) Loc is a finite set of discrete locations. (b) Var is a finite set of n continuous, real-valued variables, and $\mathcal{Q} \triangleq Loc \times \mathbb{R}^n$ is the state-space. (c) Inv is a finite set of invariants, one for each discrete location, and for each location $\ell \in Loc$, $Inv(\ell) \subseteq \mathbb{R}^n$. (d) $Flow$ is a finite set of ordinary differential inclusions, one for each continuous variable $x \in Var$, and $Flow(\ell, x) \subseteq \mathbb{R}^n$ describes the continuous dynamics in each location $\ell \in Loc$. (e) $Trans$ is a finite set of transitions between locations. Each transition is a tuple $\tau \triangleq \langle \ell, \ell', g, u \rangle$, where ℓ is a source location and ℓ' is a target location that *may* be taken when a guard condition g is satisfied, and the post-state is updated by an update map u. (f) $Init$ is an initial condition, which consists of a set of locations in Loc and a formula over Var, so that $Init \subseteq \mathcal{Q}$.

For RHA, all the expressions appearing in invariants, guards, and updates must be boolean combinations of constant inequalities, and the flows are rectangular differential inclusions ($\dot{x} \in [a, b]$ for $a \leq b$) [7]. We use the dot (.) notation to refer to different components of tuples e.g., $\mathcal{H}.Inv$ refers to the invariants of automaton $\mathcal{H}$ and $\tau.g$ refers to the guard of a transition τ. If clear from context, we drop $\mathcal{H}$ and τ and refer to the individual components of the tuple.

Semantics: The semantics of a hybrid automaton $\mathcal{H}$ are defined in terms of executions, which are sequences of states. A *state* q of $\mathcal{H}$ is a tuple $q \triangleq \langle \ell, v \rangle$, where $\ell \in Loc$ is a location, and $v \in \mathbb{R}^n$ is a valuation of all variables in Var. Formally, for a set of variables Var, a *valuation* is a function mapping each $x \in Var$ to a point in its type—here, $\mathbb{R}$. The state-space $\mathcal{Q}$ is the set of all states of $\mathcal{H}$. Updates of states are described by a transition relation $T \subseteq \mathcal{Q} \times \mathcal{Q}$. For a transition $\langle q, q' \rangle \in T$ where $q \triangleq \langle \ell, v \rangle$ and $q' \triangleq \langle \ell', v' \rangle$, we denote $q \rightarrow q' \in T$ as the transition between the current state q and the next state q'. The transition relation T is partitioned into disjoint sets of discrete transitions and continuous trajectories that respectively describe the discrete and continuous behaviors of the automaton. Thus, $T \triangleq D \cup \mathcal{T}$, where: (a) $D \subseteq \mathcal{Q} \times \mathcal{Q}$ is the set of discrete transitions that describe instantaneous updates of state, (b) $\mathcal{T} \subseteq \mathcal{Q} \times \mathcal{Q}$ is the set of continuous trajectories that describe updates of state over real time intervals.

Discrete transitions. A discrete transition $q \rightarrow q' \in D$ models an instantaneous update from the current state q to the next state q'. There is a discrete transition $q \rightarrow q' \in D$ if and only if (iff): $\exists \tau \in Trans : q.v \models \tau.g \wedge q'.v' \models \tau.u$, where $\tau.g$, and $\tau.u$ are the guard condition and the update map of the discrete transition τ, respectively.

Continuous trajectories. A continuous trajectory $q \rightarrow q' \in \mathcal{T}$ models the update of state q to q' over an interval of real time. The set-valued function Δ returns a set of states and is defined as: $\Delta(q.\ell, q.v, x, t) \in q.v.x + \int_{\delta=t_0}^{t} f(q.\ell, x) d\delta$, where $f \in Flow$ is a flow rate. A formula over $Var \cup \dot{Var}$ that describes the evolution of a real variables $x \in Var$ over a real time interval $J = [t_0, t]$, and $q.v.x$ is the value of continuous variable x of the state q at $t = t_0$. Then, there is a trajectory $q \rightarrow q' \in \mathcal{T}$ iff: $\exists t_\alpha \in \mathbb{R}_{\geq 0}\ \forall t_\beta \in \mathbb{R}_{\geq 0}\ \exists \ell \in Loc : t_\beta \leq t_\alpha \ \wedge\ \Delta(q.\ell, q.v, Var, t_\beta) \models Inv(\ell) \ \wedge\ q'.v'.Var \in \Delta(q.\ell, q.v, Var, t_\alpha)$. For each real variable x, $q.v.x$ must evolve to the valuation $q'.v'.x$ at precisely time t_α and corresponding to the flow rate of x in location ℓ. Additionally, all states along the trajectory must satisfy the invariant $Inv(\ell)$ i.e., at every point in the interval of real time $t_\beta \leq t_\alpha$.

Executions. An *execution* of $\mathcal{H}$ is a sequence $\pi \triangleq q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots$, such that: (a) $q_0 \in Init$ is an initial state, and (b) either $q_i \rightarrow q_{i+1} \in D$ is a discrete transition or $q_i \rightarrow q_{i+1} \in \mathcal{T}$ is a continuous trajectory for each consecutive pair of states in the sequence π. A state $q_k \triangleq \langle \ell_k, v_k \rangle$ is *reachable* from initial state $q_0 \triangleq \langle \ell_0, v_0 \rangle \in Init$ iff there exists a finite execution $\pi \triangleq q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_k$.

Safety specifications. In this paper, we develop the QBMC procedure to check whether safety properties of hybrid automata are satisfied up-to iteration k. A *safety specification* ϕ is a formula over Loc and Var that describes a set of states $[\![\phi]\!] \subseteq \mathcal{Q}$, where $[\![\cdot]\!]$ is the set of states satisfying ϕ. For an automaton $\mathcal{H}$ and a safety specification ϕ, the automaton satisfies the specification, denoted $\mathcal{H} \models \phi$, iff for every execution π, for every state $q_0, q_1, \ldots, q_k$ in the execution π, we have $\pi.q_k \in [\![\phi]\!]$. If $\mathcal{H} \models \phi$ for every $i \in \{0, \ldots, k\}$, then the system is safe up-to iteration k. If $\mathcal{H} \models \phi$ for any k, then the system is safe. For a safety specification ϕ, a *counterexample* is an execution π where some state $q \in \pi$ violates ϕ, i.e., $q \not\models \phi$, or equivalently, $q \notin [\![\phi]\!]$.

III. Quantified BMC for Hybrid Automata

Bounded model checking (BMC) has been used widely in verification and falsification of safety and liveness properties of various classes of systems, from finite state machines to hybrid automata. The key idea is to search for a counterexample execution whose length is bounded by a number of steps k. In other words, BMC will explore all executions from any initial state of the system Ψ to detect whether there is a way to reach a bad state that violates a given property (or to find a loop in the case of liveness). Then this path is considered as a counterexample to the property that may help the user to debug the system. For finite state systems, BMC can be encoded as a propositional formula to be checked as satisfiable or unsatisfiable using a Boolean SAT solver. For hybrid automata, BMC can be encoded as a formula over reals and finite sorts (such as Booleans, bitvectors, or enumerated types). In this paper, we focus only on hybrid automata with rectangular differential inclusion dynamics ($\dot{x} \in [a, b]$ for real constants $a \leq b$), and for this class of automata, the formulas are within linear real arithmetic (LRA). We first illustrate BMC for hybrid automata using the traditional quantifier-free encoding, and then describe the quantified BMC (QBMC), which is the main contribution of this paper.

Quantifier-Free BMC for Hybrid Automata: Let P be a set of given specifications of the hybrid automata, the BMC problem will determine whether a specification $P(q_k) \in P$ is safe after k steps, and it is:

$$\Phi(k) \triangleq I(V_0) \wedge \bigwedge_{i=0}^{k-1} T_i(V, V') \wedge (\bigvee_{i=0}^{k} P(V_i)), \tag{1}$$

where V_i corresponds to the set of variables Var of the automaton $\mathcal{H}$ appropriately renamed. For example, V_i contains of every variable $v \in Var$ syntactically renamed to v_i, etc., and V' consists of primed variables, e.g., v' for each $v \in Var$. In Equation 1, $I(V_0)$ encodes the initial set of states, $T_i(V, V')$ encodes the transition between consecutive pairs of sets of states, and $P(V_i)$ is a safety specification at iteration i. We note that the sets of variables V_i for each iteration i are implicitly existentially quantified and e.g., we could equivalently prefix $\exists V_0, V_1, \ldots, V_k$. We drop the sets of variables for a shorter notation, e.g., Equation 1 is equivalent to $I_0 \wedge \bigwedge_{i=0}^{k-1} T_i \wedge (\bigvee_{i=0}^{k} P_i)$.

Example 1: Consider the hybrid automaton $\mathcal{H}$ shown in Figure 1. Assume that the automaton starts at location loc_1, and the initial value of x is 0. The set of bad states are defined by: $P \triangleq \bigvee_{i=0}^{k} \neg(q_i.\ell_i = \ell oc_2 \implies x \geq 2.5)$. Two intervals $[a_1, b_1]$ and $[a_2, b_2]$ describe the rectangular differential inclusions for locations ℓoc_1, and ℓoc_2, respectively. This automaton would be a *timed automaton* if all of the constants values are equal, i.e., $a_1 = b_1 = a_2 = b_2$. This automaton would be a *multi-rate timed automaton* if $a_1 = b_1$ and $a_2 = b_2$ but possibly $a_1 \neq a_2$. Otherwise, this automaton is a *rectangular hybrid automaton*. Suppose that $a_1 = 1$, $b_1 = 2$, $a_2 = 3$, and $b_2 = 4$. We introduce $k + 1$ copies $x_0, x_1, ..., x_k$ and $\ell_0, \ell_1, ..., \ell_k$, where the variable x_i gives the value of the variable x, and ℓ_i indicates the location at the state q_i, representing the i^{th} step of the BMC computation for the automaton shown in Figure 1. The BMC computation of $\mathcal{H}$ for each k up to 2 can be encoded as:

- $k = 0$: $I_0 := (\ell_0 = \ell oc_1 \wedge x_0 = 0)$;
- $k = 1$ (D_0): $(\ell_0 = \ell oc_1 \wedge \ell_1 = \ell oc_2 \wedge x_0 \leq 5 \wedge x_0 \geq 2.5 \wedge x_1 = x_0)$,
- $k = 1$ $(\mathcal{T}_0)$: $(\ell_0 = \ell oc_1 \implies (\ell_1 = \ell_0 \wedge x_0 + a_1\delta \leq x_1 \wedge x_1 \leq x_0 + b_1\delta \wedge x_1 \leq 5))$,
- $k = 2$ (D_1): $(\ell_1 = \ell oc_1 \wedge \ell_2 = \ell oc_2 \wedge x_1 \leq 5 \wedge x_1 \geq 2.5 \wedge x_2 = x_1)$,
- $k = 2$ $(\mathcal{T}_1)$: $(\ell_1 = \ell oc_1 \implies (\ell_2 = \ell_1 \wedge x_1 + a_1\delta \leq x_2 \wedge x_2 \leq x_1 + b_1\delta \wedge x_2 \leq 5))$,

where δ is a fresh, real constant.[1] We split the discrete transitions and trajectories for clarity, but the entire formula to be checked for iteration $k = 1$ would just be the disjunction of these conjuncted with the formula representing $k = 0$ and the bad set of states, i.e., $I_0 \wedge (D_0 \vee \mathcal{T}_0) \wedge P$. For $k = 2$, this full formula would be $I_0 \wedge (D_0 \vee \mathcal{T}_0) \wedge (D_1 \vee \mathcal{T}_1) \wedge P$.

For $k = 1$, we dropped the obviously infeasible transition from ℓoc_2 to ℓoc_1 from D_0, which would be found as being unsatisfiable since $\ell_0 \neq \ell oc_2$. However, the transition from ℓoc_1 to ℓoc_2 also cannot occur since $x_0 = 0$, but $x_0 \not\geq 2.5$, so that part is unsatisfiable and no discrete transitions may be taken from the set of initial states. We also dropped the continuous dynamics for ℓoc_2 from $\mathcal{T}_0$ since this would also be infeasible since $\ell_0 \neq \ell oc_2$. However, real time may elapse, and as encoded, would correspond to any choice of time δ such that $x_1 \in [a_1\delta, b_1\delta]$ and $x_1 \leq 5$. Since $a_1 = 1$ and $b_1 = 2$, at most between 2.5 and 5 seconds of real time could elapse, and either case would yield $x_1 \in [0, 5]$.

For $k = 2$, we also dropped the infeasible transition and trajectory for clarity. In this case, the transition from ℓoc_1 to ℓoc_2 is enabled since $x_1 \in [0, 5]$, so the update to ℓoc_2 may occur. However, now the continuous trajectory would be infeasible since x_1 could already be 5 and the invariant requires $x_2 \leq 5$, so no real-time $\delta > 0$ may elapse, as otherwise $x_1 + a_1\delta > 5$ is unsatisfiable for $x_1 = 5$. So, the only state update would be to ℓoc_2 owing to the discrete transition.

[1]In general, a universally quantified assertion that the invariant is satisfied for every real time along the trajectory from time t_0 to time $t_0 + \delta$, although this is unnecessary for rectangular differential inclusions with linear guards and invariants for convexity reasons [2], [6], which makes this assertion fall into the combination theory of linear real arithmetic with bitvectors (or some finite sort to encode the locations).

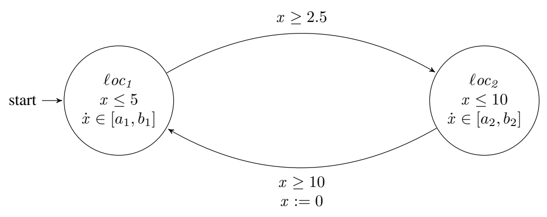

Fig. 1. The hybrid automaton $\mathcal{H}$ for Example 1.

Quantified BMC (QBMC) for Hybrid Automata: Next, we construct a quantified formula $\Omega(k)$ for BMC of $\mathcal{H}$ of length k. We introduce a vector $t = \langle t_1, t_2, ..., t_{\lceil \log_2 k \rceil} \rangle$ to index each iteration of the BMC of $\mathcal{H}$. The current state q and next state q' under the transition relation $T(V, V')$ are connected to the current state and the next state for each particular iteration t_i, for $i \in [1, \lceil \log_2 k \rceil]$. The quantified BMC formula is:

$$\Omega(k) \triangleq \exists V_0, V_1, ..., V_k, \delta \forall t \exists V, V' \mid I(V_0) \wedge T(V, V') \wedge \bigwedge_{i=0}^{k-1} t_{i+1} \rightarrow [(V = V_i) \wedge (V' = V_{i+1})] \wedge (\bigvee_{i=0}^{k} P(V_i)),$$

where we note that the existential δ encodes the real time elapse and would appear in the trajectories $\mathcal{T}$ of the disjunct $T = D \vee \mathcal{T}$.

For $k = 3$, the QBMC of the automaton of Example 1 is:

$$\begin{aligned}\Omega(3) = \exists V_0, V_1, V_2, V_3, \delta \forall t_1, t_2 \exists V, V' \mid I(V_0) \wedge T(V, V') \\ \wedge \ \{\bar{t_1} \rightarrow [(V = V_0) \wedge (V' = V_1)]\} \\ \wedge \ \{t_1 \wedge \bar{t_2} \rightarrow [(V = V_1) \wedge (V' = V_2)]\} \\ \wedge \ \{t_1 \wedge t_2 \rightarrow [(V = V_2) \wedge (V' = V_3)]\} \\ \wedge \ (P(V_0) \vee P(V_1) \vee P(V_2) \vee P(V_3)),\end{aligned} \tag{2}$$

where $V = V'$ is a shorthand indicating every variable $v \in V$ equals its corresponding counterpart $v' \in V$. In Equation 2, if the value of t_1 is 0, then there is a continuous trajectory that evolves from the initial state q_0, where $q_0.\ell_0 = \ell oc_1$ and $x_0 = 0$, to the next state q_1, where $q_1.\ell_1 = \ell oc_1$ and $x_1 \leq 5$. When $t_1 = 1$ and $t_2 = 0$, the system takes the discrete transition from the current state q_1 to the next state q_2, where $q_2.\ell_2 = \ell oc_2$ and the value of x_3 is not higher than 10. At $k = 3$, both t_1, and t_2 are true, then q_2 becomes the current state, and q_3 is the next state, where $q_3.\ell_3 = \ell oc_1$, and $x_3 \leq 5$. The discrete transition taken from q_2 to q_3 when $x \geq 10$ will reset the value of x to 0.

If it terminates, an SMT solver supporting the combined theory of bitvectors and reals with quantifiers will return SAT for the QBMC formula iff there exists an execution from an initial state to a bad state, i.e., if a bad state is reachable. Otherwise, if it terminates, it will return UNSAT if a bad state is not reachable in k steps. We note that the combination theory of linear real arithmetic with bitvectors is decidable, and Z3 is in essence a decision procedure for this theory.

IV. Experimental Results

We implement the method described in this paper as a module within HyST [21]. HyST takes as input a hybrid automaton model in an extended form of the SpaceEx XML format [16] (supporting e.g., nonlinear functions instead of only affine ones), and creates the transition relation as SMT

TABLE I
EXAMPLE 1 PERFORMANCE COMPARISON.

Tools	L	k ≤ 32		k ≤ 64		k ≤ 128	
		Time (sec)	Mem (MB)	Time (sec)	Mem (MB)	Time (sec)	Mem (MB)
QBMC	2	1.11	27.2	3.68	39.4	19.9	91.2
dReach	2	86.7	102.4	1176.4	284.7	20034	829.2
HyComp	2	0.4	97.3	0.6	101.8	1.44	109.3

formulas using the Z3 Python API. We evaluate the QBMC method described in this paper on several examples.[2] We compare the results from the QBMC method of this paper with that of dReach, which is a state-of-the-art BMC tool for nonlinear hybrid automata [23], and with that of HyComp that uses the MathSAT SMT solver [6]. All of the models for dReach and HyComp are also generated using HyST. The experiments are performed on Intel I5 2.4GHz processor with 3GB RAM, executing the method described in this paper and dReach in a VirtualBox virtual machine running Ubuntu 64-bit. Z3 version 4.3.2 was used in the evaluation. We collect the running times (Time) in seconds and the peak memory usages (Mem) in megabytes for different examples.

We first evaluate our QBMC encoding on the illustrative hybrid automata presented in Example 1, and compare the results to those of dReach and HyComp. The performances of those three different methods are shown in Table I, where QBMC denotes the QBMC presented in this paper, k is a number of steps in the BMC computation, and L is the number of discrete locations. The constants values are given as: $a_1 = 0, b_1 = 1, a_2 = 0$, and $b_2 = 2$. The results shown in Table I preliminarily indicate that our QBMC approach is capable of solving BMC significant faster than dReach, but slower than HyComp. However, our approach requires less memory usage compared to dReach and HyComp.

Next, we evaluate QBMC with several scenarios using the Fischer mutual exclusion protocol [2]. Fischer mutual exclusion is a timed distributed algorithm that ensures a mutual exclusion safety property, namely that at most one process in a network of N processes may enter a critical section simultaneously. The set of bad states is defined by: $\phi \triangleq \neg\forall i, j \in \{1, \ldots, N\} \mid (i \neq j \wedge \ q_i = cs) \ \rightarrow \ q_j \neq cs$, where q_i and q_j are variables modeling the discrete location of the automata, cs is the critical section location, and $\rightarrow$ is logical implication. We compare the performance of QBMC in solving the BMC of Fischer protocol with HyComp and dReach. Figures 2 and 3 show, respectively, the runtime and memory usage comparison among HyComp, dReach and QBMC for different numbers of processes of Fischer protocol; where QBMC-safe, QBMC-unsafe, HyComp-safe, HyComp-unsafe, dReach-safe, and dReach-unsafe denote the BMC of the safe and unsafe version of Fischer protocol using QBMC, HyComp, and dReach, respectively. Overall, HyComp is generally faster than QBMC. However, it requires a higher memory consumption than QBMC. For instance, with $k \leq 16$, the BMC of the unsafe version of Fischer protocol with 5 processes cannot terminate in HyComp due to out of memory (requiring more than 3GB). However, QBMC can solve it using less than 500 MB. Thus, we can point out that QBMC is superior than HyComp with respect to the memory usage. Moreover, Figures 2 and 3 also indicate that QBMC is able to solve BMC of hybrid automata faster and uses less memory than dReach. Due to state-space (and formula) explosion, the reduction of memory consumption is one of the major challenges to address. Since QBMC requires a smaller amount of memory usage than other quantifier-free BMC approaches, it may be effective in solving BMC of large scale problems.

[2]The preliminary implementation described in this paper, along with all the examples, is available online at: http://www.verivital.com/hyst/cfv2015.zip.

TABLE II
LYNCH-SHAVIT MUTUAL EXCLUSION PROTOCOL PERFORMANCE.

Tools	L	k ≤ 4		k ≤ 8		k ≤ 16	
		Time (sec)	Mem (MB)	Time (sec)	Mem (MB)	Time (sec)	Mem (MB)
QBMC	9^2	3.7	52.2	5.1	52.3	25.9	52.7
	9^3	15.5	65.6	31.3	87.5	1091.5	144.5
	9^4	256.1	702.8	1062.1	708.9	43578	1196.2
HyComp	9^2	0.8	121.9	1.33	132.8	9.5	170.5
	9^3	2.7	307.9	12.81	380.8	192.8	771.4
	9^4	63.9	2655.4	N/A	M/O	N/A	M/O

We also evaluate QBMC with the Lynch-Shavit mutual exclusion protocol. The Lynch-Shavit protocol is a modified version of Fischer protocol where the mutual exclusion property is time-independent. Each process of Lynch-Shavit protocol has 9 states (locations), then the Lynch-Shavit protocol with 4 processes includes 6561 discrete locations. The performance analyzing the Lynch-Shavit protocol using QBMC and Hycomp are shown in Table II, respectively. M/O presents that the peak memory usage is higher than 3GB, and N/A denotes that the information of running times is not detected due to M/O. The set of bad states of Lynch-Shavit protocol is defined similar to that of Fischer, where two processes may be in the critical section. Again, we can see the trade off between using QBMC or using HyComp. HyComp is faster than QBMC, but requires a higher memory usage. Therefore, the BMC of Lynch-Shavit protocol with 4 processes can be solved by QBMC up to $k = 16$, but cannot be solved in HyComp up to $k = 8$ due to M/O.

V. CONCLUSION AND FUTURE WORK

In this paper, we present a new SMT-based technique that encodes, in a quantified form, the BMC problem for rectangular hybrid automata (RHA), which also subsumes this encoding for timed automata. The preliminary results for the Fischer mutual exclusion protocol and Lynch-Shavit protocol indicate the capability of our method to solve the BMC problem for hybrid systems including more than a thousand locations. We compare these experimental results to those of quantifier-free BMC approaches, such as in the dReach tool that uses the dReal SMT solver, and the HyComp tool built on top of nuXmv that uses the MathSAT SMT solver. As solvers for fragments of many-sorted first-order logic such as LRA, NRA, etc. continue to improve, QBMC encodings such

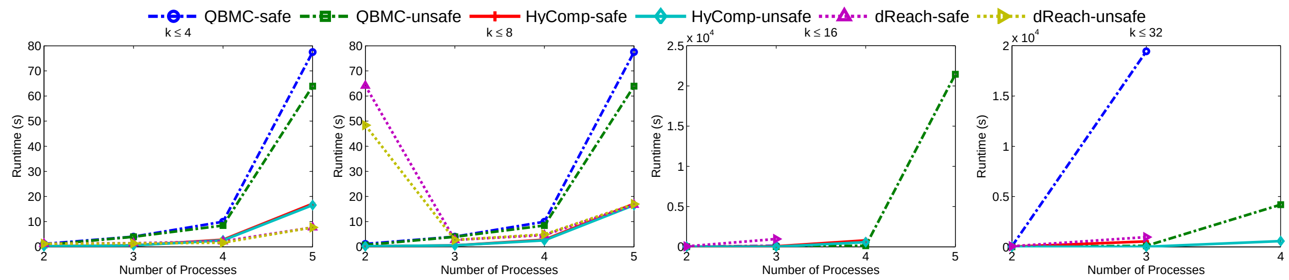

Fig. 2. Runtime comparison of HyComp, dReach and QBMC in solving the BMC of Fischer protocol.

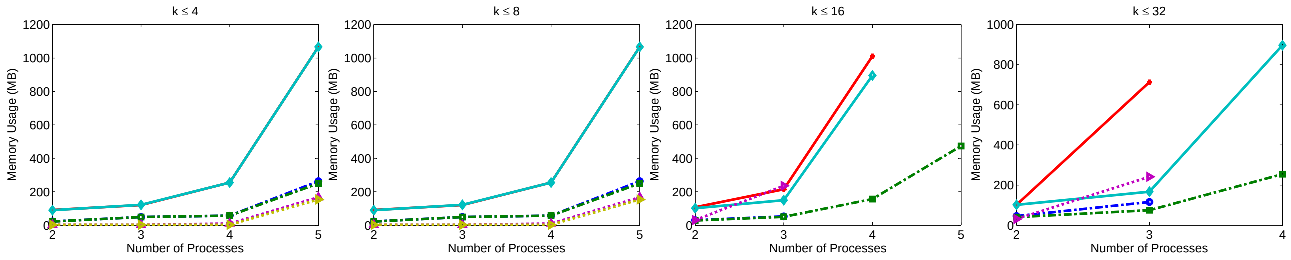

Fig. 3. Memory usage comparison of HyComp, dReach and QBMC in solving the BMC of Fischer protocol.

as the one described in this paper will become more effective, similar to how QBMC for discrete systems has been shown to be effective with QBF encodings [9]. In future work, we will conduct additional experiments and compare the results to other tools and techniques, such as UPPAAL, and also investigate more general classes of hybrid automata, such as those with linear or polynomial differential equations.

References

[1] A. Eggers, M. Fränzle, and C. Herde, "SAT modulo ODE: A direct SAT approach to hybrid systems," in *Automated Technology for Verification and Analysis*, ser. Lecture Notes in Computer Science, S. Cha, J.-Y. Choi, M. Kim, I. Lee, and M. Viswanathan, Eds. Springer Berlin / Heidelberg, 2008, vol. 5311, pp. 171–185.

[2] T. T. Johnson and S. Mitra, "A small model theorem for rectangular hybrid automata networks," in *Proceedings of the IFIP International Conference on Formal Techniques for Distributed Systems, Joint 14th Formal Methods for Open Object-Based Distributed Systems and 32nd Formal Techniques for Networked and Distributed Systems (FMOODS-FORTE)*, ser. LNCS. Springer, June 2012, vol. 7273.

[3] A. Cimatti, S. Mover, and S. Tonetta, "A quantifier-free smt encoding of non-linear hybrid automata," in *Formal Methods in Computer-Aided Design (FMCAD), 2012*, 2012, pp. 187–195.

[4] S. Gao, S. Kong, and E. Clarke, "Satisfiability modulo ODEs," in *International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, Oct. 2013.

[5] T. T. Johnson and S. Mitra, "Anonymized reachability of rectangular hybrid automata networks," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2014.

[6] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "HyComp: An SMT-based model checker for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, C. Baier and C. Tinelli, Eds. Springer Berlin Heidelberg, 2015, vol. 9035, pp. 52–67.

[7] T. A. Henzinger, "The theory of hybrid automata," in *IEEE Symposium on Logic in Computer Science (LICS)*. Washington, DC, USA: IEEE Computer Society, 1996, p. 278.

[8] T. Jussila and A. Biere, "Compressing bmc encodings with qbf," *Electronic Notes in Theoretical Computer Science*, vol. 174, no. 3, pp. 45–56, 2007.

[9] H. Mangassarian, A. Veneris, and M. Benedetti, "Robust QBF encodings for sequential circuits with applications to verification, debug, and test," *Computers, IEEE Transactions on*, vol. 59, no. 7, pp. 981–994, Jul. 2010.

[10] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB standard: Version 2.0," 2010. [Online]. Available: http://smt-lib.org/

[11] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS '08/ETAPS '08. Springer-Verlag, 2008, pp. 337–340.

[12] A. Cimatti, S. Mover, and S. Tonetta, "Smt-based scenario verification for hybrid systems," *Formal Methods in System Design*, vol. 42, no. 1, pp. 46–66, 2013.

[13] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL: A tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, ser. LNCS, R. Alur, T. Henzinger, and E. Sontag, Eds. Springer, 1996, vol. 1066, pp. 232–243.

[14] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *Journal on Software Tools for Technology Transfer*, vol. 1, pp. 110–122, 1997.

[15] X. Chen, E. Abraham, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds. Springer Berlin Heidelberg, 2013, vol. 8044, pp. 258–263.

[16] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Computer Aided Verification (CAV)*, ser. LNCS. Springer, 2011.

[17] S. Gao, J. Avigad, and E. Clarke, "Delta-decidability over the reals," in *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, 2012, pp. 305–314.

[18] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds. Springer International Publishing, 2014, vol. 8559, pp. 334–342.

[19] C. Miller, K. Gitina, and B. Becker, "Bounded model checking of incomplete real-time systems using quantified smt formulas," in *Microprocessor Test and Verification (MTV), 2011 12th International Workshop on*, Dec. 2011, pp. 22–27.

[20] T. T. Johnson and S. Mitra, "Invariant synthesis for verification of parameterized cyber-physical systems with applications to aerospace systems," in *Proceedings of the AIAA Infotech at Aerospace Conference (AIAA Infotech 2013)*, Boston, MA, Aug. 2013.

[21] S. Bak, S. Bogomolov, and T. T. Johnson, "HyST: A source transformation and translation tool for hybrid automaton models," in *Proc. of the 18th Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*. ACM, 2015.

[22] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "Hycomp: An smt-based model checker for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 52–67.

[23] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *Automated Deduction–CADE-24*. Springer, 2013, pp. 208–214.

Appendix

A. Appendix: Additional Experimental Results

In this appendix, we describe additional experimental results of the BMC of the Fischer mutual exclusion protocol using QBMC, HyComp and dReach. Figures 4 and 5 show, respectively, the runtime and memory usage comparison among HyComp, dReach and QBMC for BMC of Fischer protocol. Vertical axises are runtime in seconds and memory usage in megabytes, respectively, and horizontal axises are number of steps, k. The details of running times and memory usages of BMC for the Fischer protocol using these tools are also shown in Table III, where FS, FU denote the safe and unsafe versions of Fischer protocol, respectively, and the number following the hyphen (-) describes a number of processes for each version. In FS, a state where the set of bad states ϕ is satisfied is not reachable, while in FU, a state where ϕ is satisfied is reachable. For instance, FS-2, FU-2 are the safe and unsafe versions of the Fischer protocol with 2 processes, respectively.

Table III shows that the BMC of Fischer protocol with 64 discrete locations can be checked completely up to $k = 32$. Note that T/O means the computation time out ($\geq$ 24 hours), M/O presents that the peak memory usage is higher than 3GB, and N/A denotes that the information of times or memory usages are not detected due to M/O or T/O, respectively. The results of the BMC for unsafe versions of Fischer protocol indicate that QBMC is effective for bug detection. However, as k increases, the higher running time and the greater memory usage are required for the quantified encoding of BMC due to the increasing number of all possible paths from an initial state in the set of initial states to a bad state that does not satisfy the set of safety specifications.

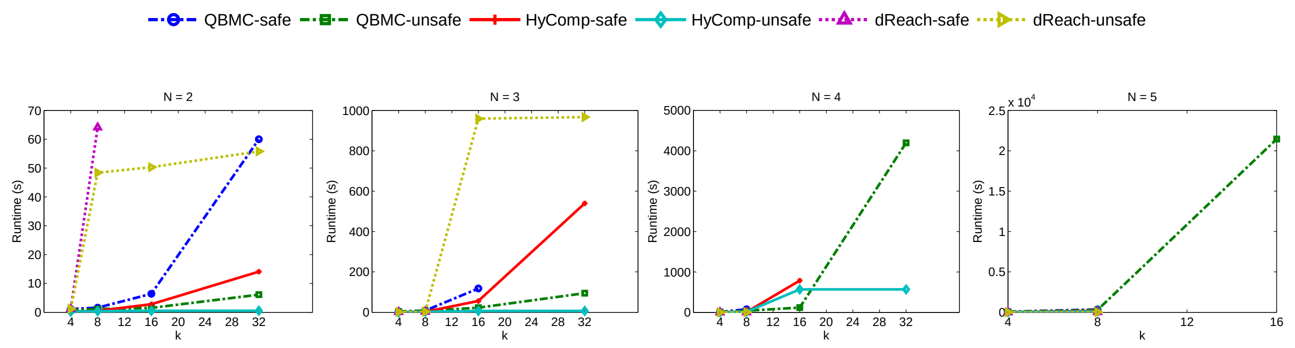

Fig. 4. Runtime comparison of HyComp, dReach and QBMC in solving the BMC of Fischer protocol.

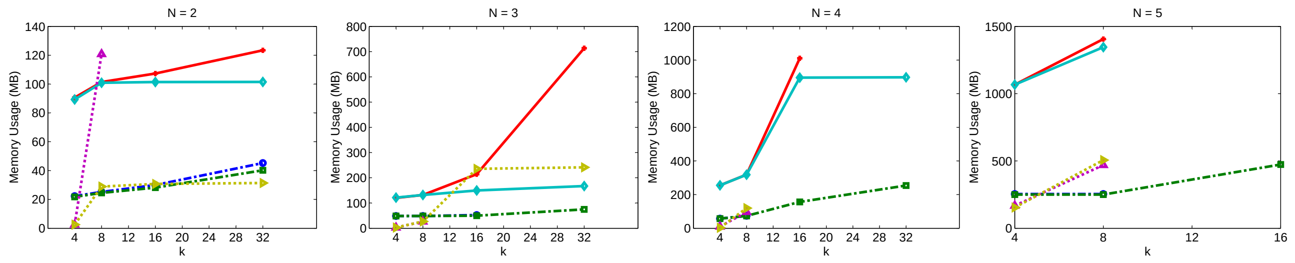

Fig. 5. Memory usage comparison of HyComp, dReach and QBMC in solving the BMC of Fischer protocol.

TABLE III
THE PERFORMANCE OF THE BMC OF FISCHER MUTUAL EXCLUSION PROTOCOL USING QBMC, HYCOMP, AND DREACH.

Tools	Example	L	$k \leq 4$		$k \leq 8$		$k \leq 16$		$k \leq 32$	
			Time (sec)	Mem (MB)	Time (sec)	Mem (MB)	Time (sec)	Mem (MB)	Time (sec)	Mem (MB)
QBMC	FS-2	4^2	1.11	22.3	1.6	25.2	6.4	30	60	45.2
	FU-2	4^2	0.7	21.73	1.1	24.7	1.52	28.2	6.1	40.2
	FS-3	4^3	4.02	48.7	8.3	48.7	117.8	52.4	19452	115.6
	FU-3	4^3	3.97	48.7	6.9	48.7	22.7	49.7	94.3	74.6
	FS-4	4^4	9.97	56.9	76.1	74.1	T/O	N/A	T/O	N/A
	FU-4	4^4	8.44	57	40.1	73.2	119.1	156.2	4197.1	254.1
	FS-5	4^5	77.51	254.3	344.4	254.4	T/O	N/A	T/O	N/A
	FU-5	4^5	63.93	249.9	288.8	249.9	21456	473.8	T/O	N/A
HyComp	FS-2	4^2	0.2	22.3	0.5	101.4	2.8	107.3	14.1	123.4
	FU-2	4^2	0.2	21.7	0.4	100.9	0.5	101.4	0.53	101.5
	FS-3	4^3	0.51	120.2	2.2	131.8	55.8	214.4	539.7	713.4
	FU-3	4^3	0.51	121.5	2.1	131.8	6.7	149.6	6.5	167.1
	FS-4	4^4	2.78	255	9.9	319.1	788	1010.4	T/O	M/O
	FU-4	4^4	2.53	255.2	13.3	318.2	569.4	895.4	568.4	897.1
	FS-5	4^5	17.13	1067	172.4	1405.9	N/A	M/O	N/A	M/O
	FU-5	4^5	16.6	1066.7	109.1	1345.4	N/A	M/O	N/A	M/O
dReach	FS-2	4^2	1.2	2.5	64.1	120.8	T/O	M/O	T/O	M/O
	FU-2	4^2	1.2	2.5	48.4	28.9	50.3	30.7	55.8	31.4
	FS-3	4^3	1.4	2.5	2.7	26.4	T/O	M/O	T/O	M/O
	FU-3	4^3	1.3	2.5	2.7	26.8	959.3	235.3	966.8	241.2
	FS-4	4^4	2.1	9.8	4.63	96.7	T/O	M/O	T/O	M/O
	FU-4	4^4	1.6	2.5	4.93	119.8	T/O	M/O	T/O	M/O
	FS-5	4^5	7.7	167.2	16.69	469.6	T/O	M/O	T/O	M/O
	FU-5	4^5	7.7	153.9	17	506.5	T/O	M/O	T/O	M/O