Bounded Model Checking with SAT/SMT

06-12-2011

SAT/SMT Summer School, MIT

Edmund M. Clarke, Jr.

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

(Joint work with Sean Gao)
Outline – Something Old, Something New

- Bounded Model Checking Using SAT
- Bounded Model Checking for Hybrid Systems
  - How to use numerical methods safely.
Method used by most “industrial strength” model checkers:

- uses Boolean encoding for state machine and sets of states.
- can handle much larger designs – hundreds of state variables.
- BDDs traditionally used to represent Boolean functions.
Problems with BDDs

- BDDs are a canonical representation. Often become too large.
- Variable ordering must be uniform along paths.
- Selecting right variable ordering very important for obtaining small BDDs.
  - Often time consuming or needs manual intervention.
  - Sometimes, no space efficient variable ordering exists.

BMC is an alternative approach to symbolic model checking that uses SAT procedures.
Advantages of SAT Procedures

- SAT procedures also operate on Boolean expressions but do not use canonical forms.
- Do not suffer from the potential space explosion of BDDs.
- Different split orderings possible on different branches.
- Very efficient implementations available.
Bounded model checking uses a SAT procedure instead of BDDs.

We construct Boolean formula that is satisfiable iff there is a counterexample of length $k$.

We look for longer and longer counterexamples by incrementing the bound $k$.

After some number of iterations, we may conclude no counterexample exists and specification holds.

For example, to verify safety properties, number of iterations is bounded by diameter of finite state machine.
Main Advantages of Our Approach

- Bounded model checking finds counterexamples fast. This is due to depth first nature of SAT search procedures.

- It finds counterexamples of minimal length. This feature helps user understand counterexample more easily.

- It uses much less space than BDD based approaches.

- Does not need manually selected variable order or costly reordering. Default splitting heuristics usually sufficient.

- Bounded model checking of LTL formulas does not require a tableau or automaton construction.
Implemented a tool BMC in 1999.

- It accepts a subset of the SMV language.

- Given $k$, BMC outputs a formula that is satisfiable iff counterexample exists of length $k$.

- If counterexample exists, a standard SAT solver generates a truth assignment for the formula.
There are many examples where BMC significantly outperforms BDD based model checking.

In some cases BMC detects errors instantly, while SMV fails to construct BDD for initial state.

Armin’s example: Circuit with 9510 latches, 9499 inputs. BMC formula has $4 \times 10^6$ variables, $1.2 \times 10^7$ clauses. Shortest bug of length 37 found in 69 seconds.
We use linear temporal logic (LTL) for specifications.

Basic LTL operators:

- next time ‘X’
- globally ‘G’
- release ‘R’
- eventuality ‘F’
- until ‘U’

Only consider existential LTL formulas $E f$, where

- $E$ is the existential path quantifier, and
- $f$ is a temporal formula with no path quantifiers.

Recall that $E$ is the dual of the universal path quantifier $A$.

Finding a witness for $E f$ is equivalent to finding a counterexample for $A \neg f$. 
System described as a Kripke structure $M = (S, I, T, \ell)$, where

- $S$ is a finite set of states and $I$ a set of initial states,
- $T \subseteq S \times S$ is the transition relation, (We assume every state has a successor state.)
- $\ell: S \rightarrow \mathcal{P}(A)$ is the state labeling.

The Microwave Oven Example:

$\mathsf{AG}(\textit{start} \rightarrow (\neg\textit{heat} \mathbin{U} \textit{close}))$
In symbolic model checking, a state is represented by a vector of state variables \( s = (s(1), \ldots, s(n)) \).

We define propositional formulas \( f_I(s) \), \( f_T(s,t) \) and \( f_p(s) \) as follows:

- \( f_I(s) \) iff \( s \in I \),
- \( f_T(s,t) \) iff \( (s,t) \in T \), and
- \( f_p(s) \) iff \( p \in \ell(s) \).

We write \( T(s,t) \) instead of \( f_T(s,t) \), etc.
Definitions and Notation (Cont.)

- If \( \pi = (s_0, s_1, \ldots) \), then \( \pi(i) = s_i \) and \( \pi^i = (s_i, s_{i+1}, \ldots) \).

- \( \pi \) is a path if \( \pi(i) \rightarrow \pi(i + 1) \) for all \( i \).

- \( Ef \) is true in \( M \) (\( M \models Ef \)) iff there is a path \( \pi \) in \( M \) with \( \pi \models f \) and \( \pi(0) \in I \).

- Model checking is the problem of determining the truth of an LTL formula in a Kripke structure. Equivalently,

  Does a witness exist for the LTL formula?
Example To Illustrate New Technique

Two-bit counter with an erroneous transition:

- Each state $s$ is represented by two state variables $s[1]$ and $s[0]$.
- In initial state, value of the counter is 0. Thus, $I(s) = \neg s[1] \land \neg s[0]$.
- Let $inc(s, s') = (s'[0] \leftrightarrow \neg s[0]) \land (s'[1] \leftrightarrow (s[0] \oplus s[1]))$
- Define $T(s, s') = inc(s, s') \lor (s[1] \land \neg s[0] \land s'[1] \land \neg s'[0])$
- Have deliberately added erroneous transition!!
▶ Suppose we want to know if counter will eventually reach state (11).

▶ Can specify the property by $\text{AF}q$, where $q(s) = s[1] \land s[0]$.

On all execution paths, there is a state where $q(s)$ holds.

▶ Equivalently, we can check if there is a path on which counter never reaches state (11).

▶ This is expressed by $\text{EG}p$, where $p(s) = \neg s[1] \lor \neg s[0]$.

There exists a path such that $p(s)$ holds globally along it.
In bounded model checking, we consider paths of length $k$.

We start with $k = 0$ and increment $k$ until a witness is found.

Assume $k$ equals 2. Call the states $s_0$, $s_1$, $s_2$.

We formulate constraints on $s_0$, $s_1$, and $s_2$ in propositional logic.

Constraints guarantee that $(s_0, s_1, s_2)$ is a witness for $\text{EG}_p$ and, hence, a counterexample for $\text{AF}_q$. 
First, we constrain \((s_0, s_1, s_2)\) to be a valid path starting from the initial state.

Obtain a propositional formula

\[
[\llbracket M \rrbracket] = I(s_0) \land T(s_0, s_1) \land T(s_1, s_2).
\]
Second, we constrain the shape of the path.

The sequence of states $s_0, s_1, s_2$ can be a loop or lasso.

If so, there is a transition from $s_2$ to the initial state $s_0, s_1$ or itself.

We write $L_l = T(s_2, s_l)$ to denote the transition from $s_2$ to a state $s_l$ where $l \in [0, 2]$.

We define $L$ as $\bigvee_{l=0}^{2} L_l$. Thus $\neg L$ denotes the case where no loop exists.
▶ The temporal property $Gp$ must hold on $(s_0, s_1, s_2)$.

▶ If no loop exists, $Gp$ does not hold and $[[Gp]]$ is false.

▶ To be a witness for $Gp$, the path must contain a loop (condition $L$, given previously).

▶ Finally, $p$ must hold at every state on the path

$$[[Gp]] = p(s_0) \land p(s_1) \land p(s_2).$$

▶ We combine all the constraints to obtain the propositional formula

$$[[M]] \land ((\neg L \land false) \lor \bigvee_{l=0}^{2} (L_l \land [[Gp]])),$$
In this example, the formula is satisfiable.

Truth assignment corresponds to counterexample path (00), (01), (10) followed by self-loop at (10).

If self-loop at (10) is removed, then formula is unsatisfiable.
Diameter

- Diameter $d$: Least number of steps to reach all reachable states. If the property holds for $k \geq d$, the property holds for all reachable states.
- Finding $d$ is computationally hard:
  - State $s$ is reachable in $j$ steps:
    
    $$R_j(s) := \exists s_0, \ldots, s_j : s = s_j \land I(s_0) \land \bigwedge_{i=0}^{j-1} T(s_i, s_{i+1})$$

    - Thus, $k$ is greater or equal than the diameter $d$ if
      $$\forall s : R_{k+1}(s) \implies \exists j \leq k : R_j(s)$$

This requires an efficient QBF checker!
Hybrid systems combine finite automata with continuous dynamical systems.

- They are widely used to model cyber-physical systems. (e.g., aerospace, automotive, and biological systems)
- They pose a grand challenge to formal verification.
  - Reachability for simple systems is undecidable.
  - Existing tools do not scale on realistic systems.
    - Less than ten variables and mostly constant dynamics.
Hybrid Systems

\[ \mathcal{H} = \langle X, Q, \text{Init}, \text{Flow}, \text{Jump} \rangle \]

- A state space \( X \subseteq \mathbb{R}^k \) and a finite set of modes \( Q \).
- \( \text{Init} \subseteq Q \times X \): initial configurations
- \text{Flow}: continuous flows
  - Each mode \( q \) is equipped with differential equations \( \frac{d\vec{x}}{dt} = f_q(\vec{x}, t) \).
- \text{Jump}: discrete jumps
  - The system can be switched from \( (q, \vec{x}) \) to \( (q', \vec{x}') \), resetting modes and variables.

Continuous flows are interleaved with discrete jumps.
Controller of an automated guided vehicle [Lee and Seshia, 2011]
Logical encoding is not limited to discrete systems.

- **Continuous Dynamics:** \[ \frac{d\vec{x}(t)}{dt} = \vec{f}(\vec{x}(t), t) \]

- The solution curve:
  \[ \alpha : \mathbb{R} \rightarrow X, \alpha(t) = \alpha(0) + \int_0^t \vec{f}(\alpha(s), s)ds. \]

- Define the predicate
  \[ \text{Flow}_f(\vec{x}_0, t, \vec{x}) = \{ (\vec{x}_0, t, \vec{x}) : \alpha(0) = \vec{x}_0, \alpha(t) = \vec{x} \} \]

Reachability:

\[ \exists \vec{x}_0, \vec{x}, t. (\text{Init}(\vec{x}_0) \land \text{Flow}_f(\vec{x}_0, t, \vec{x}) \land \text{Unsafe}(\vec{x})) \]
Combining continuous and discrete behaviors, we can encode bounded reachability for hybrid systems:

- “$\vec{x}$ is reachable after 0 discrete jumps” is definable as:
  \[
  \text{Reach}^0(\vec{x}) := \exists \vec{x}_0, t. \ [\text{Init}(\vec{x}_0) \land \text{Flow}(\vec{x}_0, t, \vec{x})]
  \]

- Inductively, “$\vec{x}$ is reachable after $k + 1$ discrete jumps” is definable as:
  \[
  \text{Reach}^{k+1}(\vec{x}) := \exists \vec{x}_k, \vec{x}_k', t. \ [\text{Reach}^k(\vec{x}_k) \land \text{Jump}(\vec{x}_k, \vec{x}_k') \land \text{Flow}(\vec{x}_k', t, \vec{x})]
  \]

Reachability within $n$ discrete jumps:

\[
\exists \vec{x}. \left( \bigvee_{i=0}^{n} \text{Reach}^i(\vec{x}) \land \text{Unsafe}(\vec{x}) \right)
\]
The formulas that we have shown are first-order formulas over reals. Because of the dynamical systems involved, they usually contain a rich set of nonlinear functions:

- polynomials
- exponentiation and trigonometric functions
- solutions of ODEs, mostly no analytic forms
Symbolic decision procedures are unlikely to scale on realistic problems.

- The arithmetic theory \((\times/+)\) is decidable but already highly complex.
  - Double-exponential (PSPACE for SMT, theoretically).
  - Very active research in the past twenty years. (Cylindrical Decomposition, Gröbner Bases, Positivstellensatz,...)
  - Available solvers: Hard to scale to more than ten variables.

- The general first-order theory over exp, sin, ODEs, ...
  - Wildly undecidable.
However, large systems of real equalities/inequalities/ODEs are routinely solved numerically.

- They are perfect for simulation, but usually regarded inappropriate for verification because of their inevitable numerical errors.
  - (Platzer and Clarke, HSCC 2008)
- Is there a way of using them still?
- We need to start with a good formalization of “numerical algorithms”.
What does it mean to say a function $f$ over reals is “numerically computable”? 

- There exists an algorithm $M_f$, such that given a good approximation of $x$, $M_f$ can find a good approximation of $f(x)$.
  
  - “A real function is computable if we can draw it faithfully on a computer!”

- This leads to the well-developed framework of Computable Analysis (a.k.a. Type-II Computability) over real numbers. [A. Turing, A. Grzegorczyk, K. Weihrauch, S. Cook]
Any real number $a$ is encoded by a name $\gamma_a : \mathbb{N} \to \mathbb{Q}$ satisfying

$$\forall i, \ |a - \gamma_a(i)| < 2^{-i}$$

A Type-II Turing machine extends the ordinary by allowing input and output tapes to be both infinite. The working tape remains finite.

Note that each symbol on the output tape of a Type-II machine needs to be written down after finitely many operations in the machine.
A function $f$ is Type-II computable, if there exists a Type-II Turing machine $\mathcal{M}_f$, such that given any name of $\gamma_{\vec{x}}$ of $\vec{x} \in \text{dom}(f)$,

$\mathcal{M}_f$ outputs a name of $\gamma_{f(\vec{x})}$ of $f(\vec{x})$. 

\[ f_M(y_1, \ldots, y_k) = y \]
Let $\mathcal{F}$ be the set of all Type-II computable functions.

This is a very general framework: $\mathcal{F}$ contains polynomials, $\exp$, $\sin$, and solutions of Lipschitz-continuous ODEs.

Consider the first-order the structure $\mathbb{R}_\mathcal{F} = \langle \mathbb{R}, 0, 1, \mathcal{F}, < \rangle$ and the corresponding language $\mathcal{L}_\mathcal{F}$.

Can we solve SMT problems in $\mathcal{L}_\mathcal{F}$ over $\mathbb{R}_\mathcal{F}$?

This would allow us to solve formulas that arise in bounded model checking of hybrid systems.
Suppose we want to decide a formula in $\mathcal{L}_F$:

$$\exists x \in I. (f(x) = 0 \land g(x) = 0).$$

($I \subseteq \mathbb{R}$ is a bounded interval where $f$ and $g$ are defined).

- Numerical algorithms can never compute $f(x)$ and $g(x)$ precisely for all $x$.
- But how about fixing some error bound $\delta$, and relaxing the formula to:

$$\exists x \in I. (|f(x)| < \delta \land |g(x)| < \delta)?$$
We can consider formulas whose satisfiability is invariant under numerical perturbations. Formally:

- Consider any formula $\varphi := \bigwedge_i (\bigvee_j f_{ij}(\vec{x}) = 0)$.
  - Inequalities are turned into interval bounds on slack variables.

- A $\delta$-perturbation on $\varphi$ is a constant vector $\vec{c}$ satisfying $||\vec{c}|| < \delta$ ($|| \cdot ||$ denotes the maximum norm)
  
  $$\varphi^{\vec{c}} := \bigwedge_i (\bigvee_j f_{ij}(\vec{x}) = c_{ij})$$

- We say $\varphi$ is $\delta$-robust, if its satisfiability is invariant under $\delta$-perturbations:
  
  For any $\delta$-perturbation $\vec{c}$, $\exists \vec{x}. \varphi \leftrightarrow \exists \vec{x}. \varphi^{\vec{c}}$. 
As it turns out, robust formulas in $\mathcal{L}_F$ have nice computational properties.

- **Theorem:**
  Satisfiability of robust bounded SMT problems over $\mathbb{R}_F$ is decidable.
  
  - This is significant given the richness of $F$: exp, sin, ODEs...

- Decidability can be extended to quantified formulas.

- (Reasonably low) complexity results are in progress.
For general formulas, we can produce decision procedures using numerical oracles (with an error bound $\delta$) that guarantee:

- If $\varphi$ is decided as “unsatisfiable”, then it is indeed unsatisfiable.
- If $\varphi$ is decided as “satisfiable”, then:

  Under some $\delta$-perturbation $\vec{c}$, $\varphi^{\vec{c}}$ is satisfiable.

If a decision procedure satisfies this property, we say it is “$\delta$-complete”.
Recall that when bounded model checking a hybrid system $\mathcal{H}$, we ask if

$$\varphi : \text{Reach}_{\mathcal{H}}^n(\vec{x}) \land \text{Unsafe}(\vec{x})$$

is satisfiable.

- If $\varphi$ is unsatisfiable, then $\mathcal{H}$ is safe up to depth $n$.
- If $\varphi$ is satisfiable, then $\mathcal{H}$ is unsafe.
Consequently, using a $\delta$-complete decision procedure we can guarantee:

- If $\varphi$ is “unsatisfiable”, then $\mathcal{H}$ is safe up to depth $n$.
- If $\varphi$ is “satisfiable”, then $\mathcal{H}$ is unsafe under some $\delta$-perturbation!

Consequently, if a system can become unsafe under some $\delta$-perturbation, we will be able to detect such unsafety.

- This can not be achieved using precise symbolic algorithms.
We are developing the practical SMT solver \texttt{dReal}.

- DPLL(T) + Interval Constraint Propagation (ICP).
  - ICP = Interval Arithmetic + Constraint Propagation
    - Floating-point arithmetic (no need for precise arithmetic)
    - ICP can handle highly complex nonlinear constraint systems with thousands of variables.
  - The DPLL(T) framework: SAT solver + ICP solver.

- Currently solvable signature: \(+/\times\ exp,\ sin\). \cite{Gao et al. FMCAD 2010}

- In progress: Numerically stable ODEs.