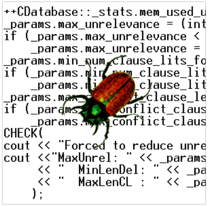
SMT Solving for Nonlinear Theories over the Reals

Edmund M. Clarke
School of Computer Science
Carnegie Mellon University



Joint Work with Sicun Gao, Soonho Kong, and Jeremy Avigad

Special thanks to Lenore Blum for her insightful comments.

Symbolic Model Checking with BDDs

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 (Clarke, Biere, Cimatti, Zhu)

- 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.

Bounded Model Checking (Cont.)

- 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.

Main Advantages of Our Approach (Cont.)

- 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.

Implementation

- 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.

Performance

- 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 4×10^6 variables, 1.2×10^7 clauses. Shortest bug of length 37 found in 69 seconds.

Temporal Logic

- We use linear temporal logic (LTL) for specifications.
- Basic LTL operators:

next time	'$\mathbf{X}$'	*eventuality*	'$\mathbf{F}$'
globally	'$\mathbf{G}$'	*until*	'$\mathbf{U}$'
release	'$\mathbf{R}$'		

Temporal Logic

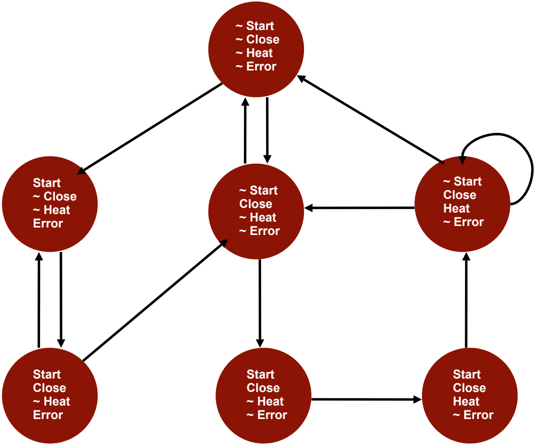
- Only consider existential LTL formulas $\mathbf{E}f$, where
  - $\mathbf{E}$ is the existential path quantifier, and
  - f is a temporal formula with no path quantifiers.
- Finding a witness for $\mathbf{E}f$ is equivalent to finding a counterexample for $\mathbf{A}\neg f$.

Kripke Structure

- 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.)
  - $\ell\colon S \to \mathcal{P}(\mathcal{A})$ is the state labeling.

The Microwave Oven Example

$$\mathbf{AG}(\mathit{start} \rightarrow (\neg \mathit{heat}\ \mathbf{U}\ \mathit{close}))$$



Definitions and Notation (Cont.)

- 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 $\pi^i = (s_i, s_{i+1}, \ldots)$.
- π is a path if $\pi(i) \to \pi(i+1)$ for all i.
- $\mathbf{E}f$ is true in M ($M \models \mathbf{E}f$) iff there is a path π 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?

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:

$$R_j(s) := \exists s_0, \ldots, s_j : s = s_j \wedge I(s_0) \wedge \bigwedge_{i=0}^{j-1} T(s_i, s_{i+1})$$

  - Thus, k is greater or equal than the diameter d if

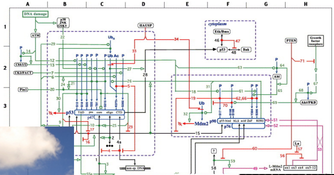
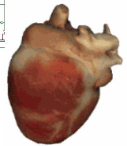
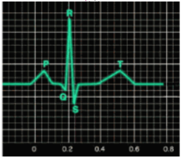
$$\forall s : R_{k+1}(s) \Longrightarrow \exists j \leq k : R_j(s)$$

This requires an efficient QBF checker!

The Cyber-Physical Challenge

- Complex aerospace, automotive, biological systems.
- They combine discrete and continuous behaviors.
- Many are safety-critical.



Bounded Model Checking for Hybrid Automata

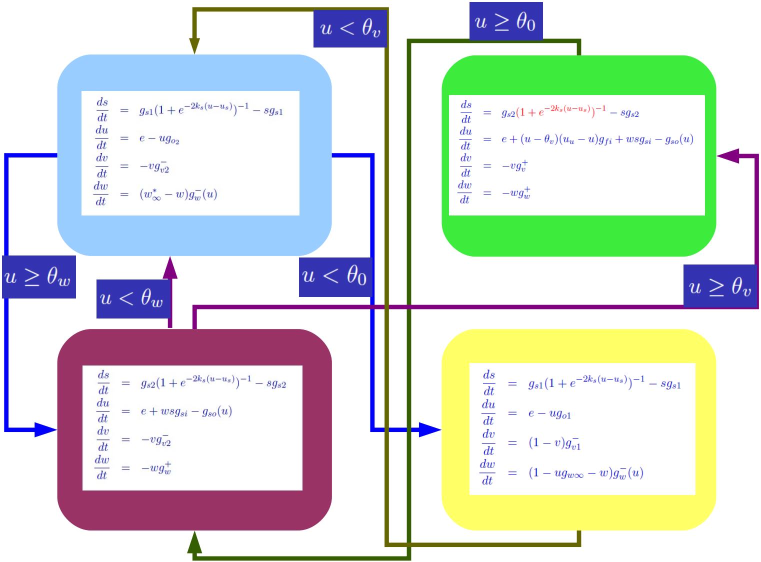
- Hybrid automata [Henzinger 1996] are widely used to model cyber-physical systems.
- They combine finite automata with continuous dynamical systems.
- Grand challenge for formal verification!
  - Reachability for simple systems is undecidable.
  - Existing tools do not scale on realistic systems.

Hybrid Systems

$\mathcal{H} = \langle X, Q, \mathsf{Init}, \mathsf{Flow}, \mathsf{Jump} \rangle$

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

Example: Cardiac-Cell Model



Reachability for Continuous Systems

Single differential equation case:

- Continuous Dynamics: $\frac{d\vec{x}(t)}{dt} = \vec{f}(\vec{x}(t), t)$
  - The solution curve:
    $\alpha : \mathbb{R} \to X,\ \alpha(t) = \alpha(0) + \int_0^t \vec{f}(\alpha(s), s)ds.$
  - Define the predicate
    $[\![\mathsf{Flow}(\vec{x}_0, t, \vec{x})]\!]^{\mathcal{M}} = \{(\vec{x}_0, t, \vec{x}) : \alpha(0) = \vec{x}_0, \alpha(t) = \vec{x}\}$

- Reachability: Is it possible to reach an unsafe state from an initial state following trajectory of differential equations?
  - $\exists \vec{x}_0, \vec{x}, t.\ (\mathsf{Init}(\vec{x}_0) \wedge \mathsf{Flow}(\vec{x}_0, t, \vec{x}) \wedge \mathsf{Unsafe}(\vec{x}))$?

Reachability for Hybrid Systems

Combining continuous and discrete behaviors, we can encode bounded reachability:

- "$\vec{x}$ is reachable after after 0 discrete jumps":

$$\mathsf{Reach}^0(\vec{x}) := \quad \exists \vec{x}_0, t.\ [\mathsf{Init}(\vec{x}_0) \wedge \mathsf{Flow}(\vec{x}_0, t, \vec{x})]$$

- Inductively, "$\vec{x}$ is reachable after $k+1$ discrete jumps" is definable as:

$$\mathsf{Reach}^{k+1}(\vec{x}) := \exists \vec{x}_k, \vec{x}'_k, t.\ [\mathsf{Reach}^k(\vec{x}_k) \wedge \mathsf{Jump}(\vec{x}_k, \vec{x}'_k) \wedge \mathsf{Flow}(\vec{x}'_k, t, \vec{x})]$$

- Unsafe within n discrete jumps:

$$\exists \vec{x}.\ (\bigvee_{i=0}^{n} \mathsf{Reach}^i(\vec{x}) \wedge \mathsf{Unsafe}(\vec{x}))\ ?$$

A Major Obstacle

We have shown how to use first-order formulas over the real numbers to encode formal verification problems for hybrid automata.

- Need to decide the truth value of formulas, which include nonlinear real functions.
  - Polynomials
  - Exponentiation and trigonometric functions
  - Solutions of ODEs, mostly no closed forms
- High complexity for polynomials; undecidable for either $\sin$ or $\cos$.

Connection to Type 2 Computability

- Negative results put a limit on symbolic decision procedures for the theory over nonlinear real functions.
- In practice (control engineering, scientific computing) these functions are routinely computed numerically.
- Can we use numerical algorithms to decide logic formulas over the reals?

Computable Real Numbers

- A real number $a \in \mathbb{R}$ is computable if it has a name $\gamma_a : \mathbb{N} \to \mathbb{Q}$ that is a total computable function.

  - $0.33\ldots$, $\sqrt{2}$, π, e, $0.1010100100010000001\ldots$

- Not all reals are computable!
  - There are only countably many Turing machines while there are uncountably many real numbers.

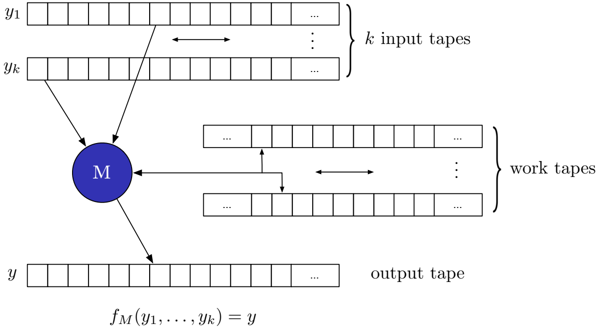
Quote from Turing's 1936 Paper

- "Equally easy to define and investigate computable functions of an integral variable or a real or computable variable."
  - A. M. Turing, On Computable Numbers with an Application to the Entscheidungsproblem, Proceedings of the London Math Society, 1936.
- A real function f is computable, if there exists a Type 2 Turing Machine that maps any name γ_a of a to a name $\gamma_{f(a)}$ of $f(a)$.

Type 2 Turing Machines

A Type 2 Turing Machine extends an ordinary (Type 1) Turing Machine in the following way.

- Both the input tapes are infinite and read-only.
- The output tape is infinite and one-way.



Connection to Type 2 Computability

- Type 2 computability gives a theoretical model of numerical computation.
  - $\exp$, $\sin$, ODEs are all Type 2 computable functions.
- We have developed a special type of decision procedure for first-order theories over the reals with Type 2 computable functions.
  - [Gao, Avigad, Clarke LICS2012, IJCAR2012].

Perturbations on Logic Formulas

Satisfiability of quantifier-free formulas under numerical perturbations:

- Consider any formula

$$\varphi : \ \bigwedge_i (\bigvee_j f_{ij}(\vec{x}) = 0)$$

  - Inequalities are turned into interval bounds on slack variables.

- For any $\delta \in \mathbb{Q}^+$, let $\vec{c}$ be a constant vector satisfying $||\vec{c}||_{\max} \leq \delta$. A δ-perturbation on φ is the formula:

$$\varphi^{\vec{c}} : \ \bigwedge_i (\bigvee_j f_{ij}(\vec{x}) = c_{ij})$$

The δ-Decision Problem

We developed a decision procedure using numerical techniques (with an error bound δ) that guarantees:

- If φ is decided as "unsatisfiable", then it is indeed unsatisfiable.
- If φ is decided as "δ-satisfiable", then:

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

If a decision procedure satisfies this property, we say it is "δ-complete".

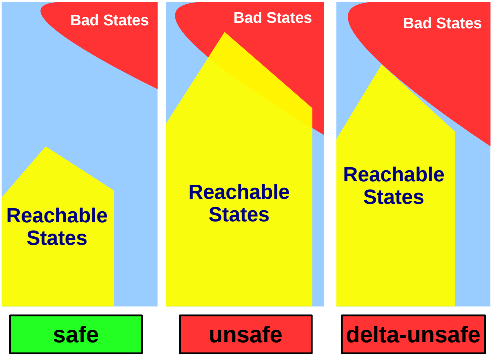
Decidability and Complexity

- The delta-decision problem is decidable for bounded first-order formulas over arbitrary Type 2 computable functions.
- Complexity: (using [Ko 1991, Weihrauch 2000, Kawamura 2010])
  - $\mathbf{NP}$-complete for existential formulas in $\{+, \times, \exp, \sin, ...\}$.
  - $\mathbf{PSPACE}$-complete for existential formulas with ODEs.
- Note the difference: The strict decision problems are all undecidable for these signatures.
- This is not bad news: Modern SAT/SMT solvers can often handle many $\mathbf{NP}$-complete problems in practice.

Delta-Complete Bounded Model Checking

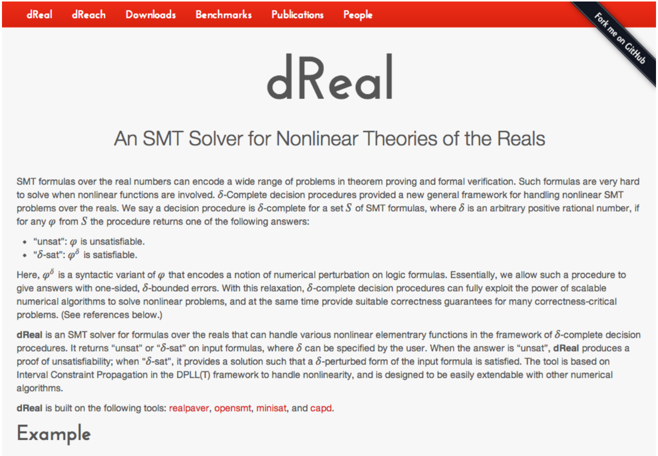
Recall that when bounded model checking a hybrid system $\mathcal{H}$, we ask if $\varphi : \text{Reach}_{\mathcal{H}}^{\leq n}(\vec{x}) \wedge \text{Unsafe}(\vec{x})$ is satisfiable.

- If φ is **unsatisfiable**, then $\mathcal{H}$ is safe up to depth n.
- If φ is δ-**satisfiable**, then $\mathcal{H}$ is unsafe under some δ-perturbation.



Practical tool: dReal

Our solver dReal is open-source at dreal.cs.cmu.edu.



dReal dReach Downloads Benchmarks Publications People

Fork me on GitHub

dReal

An SMT Solver for Nonlinear Theories of the Reals

SMT formulas over the real numbers can encode a wide range of problems in theorem proving and formal verification. Such formulas are very hard to solve when nonlinear functions are involved. δ-Complete decision procedures provided a new general framework for handling nonlinear SMT problems over the reals. We say a decision procedure is δ-complete for a set S of SMT formulas, where δ is an arbitrary positive rational number, if for any φ from S the procedure returns one of the following answers:

- "unsat": φ is unsatisfiable.
- "δ-sat": φ^δ is satisfiable.

Here, φ^δ is a syntactic variant of φ that encodes a notion of numerical perturbation on logic formulas. Essentially, we allow such a procedure to give answers with one-sided, δ-bounded errors. With this relaxation, δ-complete decision procedures can fully exploit the power of scalable numerical algorithms to solve nonlinear problems, and at the same time provide suitable correctness guarantees for many correctness-critical problems. (See references below.)

dReal is an SMT solver for formulas over the reals that can handle various nonlinear elementrary functions in the framework of δ-complete decision procedures. It returns "unsat" or "δ-sat" on input formulas, where δ can be specified by the user. When the answer is "unsat", **dReal** produces a proof of unsatisfiability; when "δ-sat", it provides a solution such that a δ-perturbed form of the input formula is satisfied. The tool is based on Interval Constraint Propagation in the DPLL(T) framework to handle nonlinearity, and is designed to be easily extendable with other numerical algorithms.

dReal is built on the following tools: realpaver, opensmt, minisat, and capd.

Example

dReal

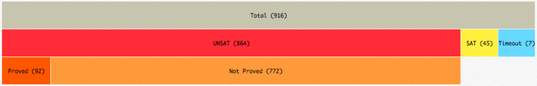
- Nonlinear signatures including exp, sin, etc., and Lipschitz-continuous ODEs.
- δ-Complete and correctness proofs are provided.
- Tight integration of DPLL(T), interval arithmetic, constraint solving, reliable integration, etc.

Example: Kepler Conjecture Benchmarks

- Around 1000 formulas. Huge combinations of nonlinear terms.
- dReal solves over 95% of the formulas. (5-min timeout each)

Flyspeck Project (Kepler's Conjecture) Benchmark Result

We have extracted 916 non-linear smt2 formulae from the Flyspeck Project.

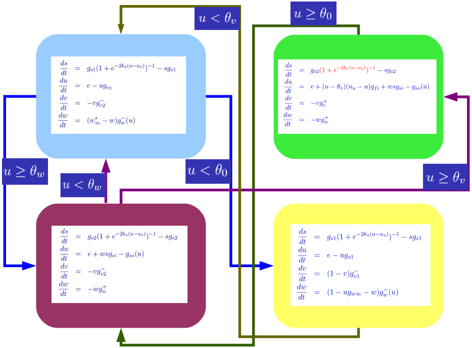


Among 916 formulae, we have 864 UNSATs, 45 δ-SATs ($\delta = 10^{-3}$), and 7 Timeouts (= 5mins). We were able to verify 92 instances of the 864 UNSAT results. All the experiments below are run on a machine of with a 48-core 2.2GHz AMD Opteron Processor and 512GB of RAM.

Filename	Formula ID	Solving Time (sec)	# of Vars	# of Arith Op	# of Non-poly Op	Proof Size (byte)	Result	Proof Checked	# of Proved Axioms	# of Subproblems Generated	Proof Checking Time (sec)	# of Proof Checking Depths
785	9414951439	0:00.01	6	80	1	951	unsat	V	3245	3244	234.950	9
814	181212899 0	0:00.01	6	95	1	1020	unsat	V	2019	2018	187.250	9
903	5766053833	0:00.25	6	2722	24	20081	unsat	V	414	413	146.230	9
815	181212899 1	0:00.01	6	95	1	1020	unsat	V	2001	2000	123.440	9
896	7720405539	0:00.27	6	2711	24	20024	unsat	V	209	208	107.800	9
811	4491491732	0:00.26	6	2731	24	20190	unsat	V	180	179	106.670	9
771	9563139965 e	0:00.27	6	2709	24	20021	unsat	V	222	221	82.500	9

Example: Cardiac-Cell Model

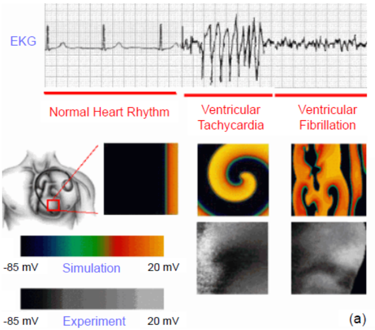
- The cardiac-cell model is a hybrid system that contains nonlinear differential equations.
  - No existing formal analysis tool can analyze this model.
- The unsafe states of the model lead to serious cardiac disorder.



Example: Cardiac-Cell Model

- Using our tool dReal, we check the safety property "globally $u < \theta_v$".

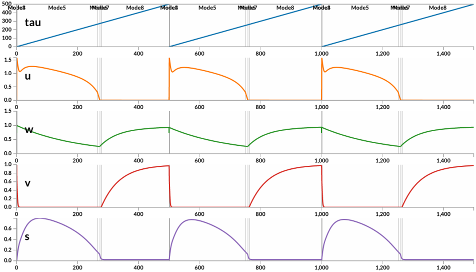
"When the property is violated, the cardiac cells lose excitability, which would trigger a spiral rotation of electrical wave and break up into a disordered collection of spirals (fibrillation)."

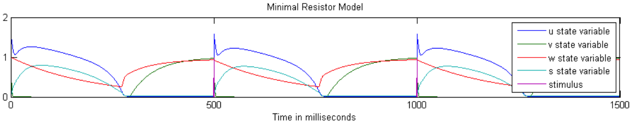


Example: Cardiac-Cell Model

Counterexample found by dReal, confirmed by experimental data.

- The formulas we solved contain over 200 highly nonlinear ODEs and over 600 variables.





Conclusion

- Turing's original goal of understanding numerical computation has become important in design and analysis of cyber-physical systems.
- We can utilize the notion of computability over the reals in formal verification of such systems.
- Practical solver: dReal (open-source at dreal.cs.cmu.edu).
- Current applications:
  - Completing formal proofs for the Kepler Conjecture
  - Finding parameters for cancer treatment models
  - Verifying safety of autonomous vehicles