IA169 System Verification and Assurance

Bounded Model Checking

Jiří Barnat

Reminder – SAT and SMT

Satisfiability – SAT

- Finding a valuation of Boolean variables that makes a given formula of propositional logic true.

Satisfiability Modulo Theory – SMT

- Deciding satisfiability of a first-order formula with equality, predicates and function symbols that encode one or more theories.

Typical SMT Theories

- Unbounded integer and real arithmetic.
- Bounded integer arithmetic (bit-vectors).
- Theory of data structures (lists, arrays, ...).

Reminder – SAT and SMT Solvers

ZZZ aka **Z3**

- Tool developed by Microsoft Research.
- WWW interface — `http://www.rise4fun.com/Z3`
- Binary API for use in other tools and applications.

SMT-LIB

- Standardised language for SMT queries.
- Freely available library with a SMT implementation.

Reminder – Satisfiability and Validity

Observation

- Formula is valid if and only if its negation is not satisfiable.

Consequence

- SAT and SMT solvers can be used as tools for proving validity of formulated statements.

Model Synthesis

- SAT solvers not only decide satisfiability of formulas, but for satisfiable formulas also give the valuation which makes the formula true.
- Unlike theorem provers, they give a "counterexample" in case the statement to be proven is false.

Checking Safety Properties via SAT Reduction

Bounded Model Checking (BMC)

Hypothesis

- If the system contains an error, it can be reproduced with only a small number of controlled steps.

Method Idea

- If we use model checking for error detection, it is sensible to check whether an error (a violation of specification) appears within first k steps of execution.

Literature

- Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu: Symbolic Model Checking without BDDs. TACAS 1999: 193-207, LNCS 1579.
- Henry A. Kautz, Bart Selman: Planning as Satisfiability.Proceedings of the 10th European conference on Artificial intelligence (ECAI'92): 359-363, 1992, Kluwer.

Reduction of BMC to SAT

Prerequisites

- The set of prefixes of length k of all runs of a Kripke structure M can be encoded by a Boolean formula $[M]^k$.
- Violation of a *safety* property which can be observed within k steps of the system can be encoded as $[\neg\varphi]^k$.

Reduction to SAT

- We check the satisfiability of $[M]^k \wedge [\neg\varphi]^k$.
- Satisfiability indicates the existence of a counterexample of length k.
- Unsatisfiability shows non-existence of a counterexample of length k.

Kripkeho structure as a Boolean formula

Prerequisites

- Let $M = (S, T, I)$ be a Kripke structure with initial state $s_0 \in S$.
- Arbitrary state $s \in S$ can be represented by a bit vector of size n, that is state $s = \langle a_0, a_1, \ldots, a_{n-1} \rangle$.

Encoding M with Boolean Formulae

- $Init(s)$ – formula which is satisfiable for the valuation of variables $a_1, a_2, ..., a_n$ that describe the state s_0.
- $Trans(s, s')$ – a formula which is satisfiable for a pair of state vectors s, s', iff the valuations $a_1, a_2, ..., a_n, a'_1, a'_2, ..., a'_n$ describe states between which a transition $(s, s') \in T$ exists.

Description of System Runs of Length k

- Run of length k consists of $k+1$ states $s_0, s_1, \ldots, s_k$.
- The set of all runs of size k of the structure M is designated $[M]^k$ and described by the following formula:

$$[M]^k \equiv Init(s_0) \wedge \bigwedge_{i=1}^{k} Trans(s_{i-1}, s_i)$$

Example$[M]^3 \wedge [\neg\varphi]^3$

- $Init(s_0) \wedge Trans(s_0, s_1) \wedge Trans(s_1, s_2) \wedge Trans(s_2, s_3) \wedge \neg\varphi(s_3)$

Completeness of BMC

Completeness of BMC for Detecting Safety Violations

Problem – Undetected Violation of a Safety Property

- The violation is not reachable using a path of length k.
- Paths shorter than k are not encoded in $[M]^k$.

Upper Bound on k

- If $k \geq d$ where d is the graph diameter, all possible error locations are covered.
- The diameter of the graph is bounded by 2^n, where n is the number of bits of the state vector.

Solution

- Executing BMC iteratively for each $k \in [0, d]$.

Automated Detection of Graph Diameter

Facts

- Asking the user is unrealistic.
- Safe upper bounds are extremely overstated.
- We would like the verification procedure itself to detect whether k should be increased further.

Skeleton of an Algorithm for Complete BMC

$k = 0$
while `(true)` **do**
  if (counterexample of length k exists)
    then return `"Invalid"`
  if (all states are reachable within k steps)
    then return `"Valid"`
  $k = k + 1$
od

Notation I

Prerequisites

- Kripke structure $M = (S, T, I)$.
- States are described by bit vectors of fixed length.
- *Trans* is a SAT representation of a binary relation T.

Path of Length n

$$path(s_{[0..n]}) \quad \equiv \quad \bigwedge_{0 \leq i < n} Trans(s_i, s_{i+1})$$

Validity of Statement Q Along the Entire Path

$$all.Q(s_{[0..n]})$$

Notation II

A Loop-Free Path

$$loopFree(s_{[0..n]}) \quad \equiv \quad path(s_{[0..n]}) \ \wedge \bigwedge_{0 \leq i < j \leq n} s_i \neq s_j$$

Existence of a Path of Length n From s_0 to s_n

$$path_n(s_0, s_n) \quad \equiv \quad \exists s_1 \ldots s_{n-1}.path(s_{[0..n]})$$

Shortest Path

$$shortest(s_{[0..n]}) \quad \equiv \quad path(s_{[0..n]}) \ \wedge \neg\Big(\bigvee_{0 \leq i < n} path_i(s_0, s_n)\Big)$$

Equivalent Problem Formulation

Verification

- We would like to show that no state that would violate the specification φ is reachable from the initial configuration, i.e. we want to show that

$$\forall i.\forall s_0 \ldots s_i.\Big(Init(s_0) \wedge path(s_{[0..i]}) \implies \varphi(s_i)\Big)$$

Alternatively

- We want to show that from an error state, the initial state is not reachable when going backwards

$$\forall i.\forall s_0 \ldots s_i.\Big(Init(s_0) \Longleftarrow path(s_{[0..i]}) \wedge \neg\varphi(s_i)\Big)$$

Equivalently

$$\forall i.\forall s_0 \ldots s_i.\ \neg\Big(Init(s_0) \wedge path(s_{[0..i]}) \wedge \neg\varphi(s_i)\Big)$$

Termination Condition in the BMC Algorithm Skeleton

- No longer acyclic path from the initial state exists, that is, the following formula is unsatisfiable:

$$Init(s_0) \;\wedge\; loopFree(s_{[0..i+1]})$$

- **Holds symmetrically for backwards reachability from error states.**

Solution 1

- $\texttt{not SAT}\Big(\; loopFree(s_{[0..i+1]}) \;\wedge\; Init(s_0) \;\Big)$
  $\vee$
  $\texttt{not SAT}\Big(\; loopFree(s_{[0..i+1]}) \;\wedge\; \neg\varphi(s_{i+1}) \;\Big)$

Termination of BMC – Acyclic paths II

Higher Efficiency Termination Criterion

- When using backward reachability from $\neg\varphi$ states, paths that visit other $\neg\varphi$ states do not need to be considered.
- Symmetrically holds also for forward reachability with multiple initial states: for completeness detection, paths that visit other initial states do not need to be considered.

Solution 2

- $\texttt{not SAT}\Big(\mathit{loopFree}(s_{[0..i+1]}) \wedge \mathit{Init}(s_0) \wedge \mathit{all}.\neg\mathit{Init}(s_{[1..i+1]}) \Big)$
  $\vee$
  $\texttt{not SAT}\Big(\mathit{loopFree}(s_{[0..i+1]}) \ \wedge \ \neg\varphi(s_{i+1}) \ \wedge \ \mathit{all}.\varphi(s_{[0..i]}) \Big)$

BMC not starting with $k = 0$

Observation

- For small values of k, SAT queries give neither a counterexample nor enable termination.
- We want to start BMC with $k > 0$.

Reformulating the Counterexample Test

- The original test for counterexample existence for a given k

$$\mathtt{SAT}\Big(Init(s_0) \wedge path(s_{[0..k]}) \wedge \neg\varphi(s_k)\Big)$$

needs to be changed so that we do not miss counterexamples shorter than the initial value of k.

- New test for the existence of a counterexample:

$$\mathtt{SAT}\Big(Init(s_0) \wedge path(s_{[0..k]}) \wedge \neg all.\varphi(s_{[0..k]})\Big)$$

k-induction in BMC

Observation

- The tests can be re-formulated so that they resemble the structure of mathematical induction.
- TAUT is a tautology test (unsatisfiability of negation).

Base Case

- Test for counterexample existence.

$$\mathtt{SAT}\Big(\neg\big(\ Init(s_o) \wedge path(s_{[0..i]})\ \implies\ all.\varphi(s_{[0..i]})\ \big)\Big)$$

Inductive Step

- Test for completeness.

$$\mathtt{TAUT}\Big(\neg Init(s_0) \Longleftarrow all.\neg Init(s_{[1..(i+1)]}) \wedge loopFree(s_{[0..i+1]})\Big)$$
$$\vee$$
$$\mathtt{TAUT}\Big(\ loopFree(s_{[0..i+1]}) \wedge all.\varphi(s_{[0..i]})\ \implies\ \varphi(s_{i+1})\ \Big)$$

Acyclic vs Shortest Paths in BMC

Observation

- Graph diameter (d) is the length of the longest of the shortest paths between each pair of vertices in the graph.
- An acyclic path can be substantially longer than the graph diameter.

BMC with Shortest Paths

- BMC is correct if *loopFree* is replaced with *shortest*.
- The *shortest* predicate, however, needs quantifiers and is as such not a purely SAT application.

For more details, see ...

- Mary Sheeran, Satnam Singh, and Gunnar Stålmarck: Checking Safety Properties Using Induction and a SAT-Solver, FMCAD 2000, 108-125, LNCS 1954, Springer.

LTL and BMC

LTL Verification with BMC

Observation 1

- LTL is only well-defined for infinite runs.
- For evaluating LTL on finite paths, we use three-value logic (true, false, cannot say).
- Validity of some LTL formulas cannot be decided on any finite path (eg. $GF\ a$).

Observation 2

- Cycles that consist of only a few states are unrolled by BMC to acyclic paths of length k.
- We allow encoding lasso-shaped paths.
- That is, (k, l)-cyclic paths.

(k,l)-cyclic paths

(k,l)-cyclic runs

- A run $\pi = s_0 s_1 s_2 \ldots$ of a Kripke structure $M = (S, T, I, s_0)$ is (k, l)-cyclic if

$$\pi = (s_0 s_1 s_2 \ldots s_{l-1})(s_l \ldots s_k)^{\omega},$$

where $0 < l \leq k$ a $s_{l-1} = s_k$.

Observation

- If π is (k, l)-cyclic, π is also $(k+1, l+1)$-cyclic.
- Treating finite paths as (k, k)-cyclic is incorrect (could create a non-existent run in M).
- Every path of length k is either acyclic or (k, l)-cyclic.

Semantics of LTL for Finite Prefixes

- Let π be a run of a Kripke structure M.
- k is given.
- $\pi = \pi^0$

$$\begin{array}{rcl} \pi^i \models_{nl} X\,\varphi & \text{iff} & i < k \wedge \pi^{i+1} \models_{nl} \varphi \\ \pi^i \models_{nl} \varphi\,U\,\psi & \text{iff} & \exists j.i \leq j \leq k, \pi^j \models_{nl} \psi \text{ and} \\ & & \forall m.i \leq m < j, \pi^i \models_{nl} \varphi \end{array}$$

Semantics of $\models_k$ for LTL in BMC

- For (k, l)-cyclic paths, $\pi \models_k \varphi \iff \pi \models \varphi$ holds.
- For acyclic paths, $\pi \models_k \varphi \iff \pi^0 \models_{nl} \varphi$ holds.
- $\models_k \Longrightarrow \models_{k+1}$, $\models_k$ approximates $\models$

Goal

- We construct a Boolean formula $[M, \varphi, k]$ which is satisfiable iff Kripke structure M has a run π such that $\pi \models_k \varphi$.
- $[M, \varphi, k] \equiv [M]^k \wedge [\varphi, k]$

Encoding

- $[M]^k$ encodes all paths of length k
- $[\varphi, k] \equiv \_[\varphi, k]_0 \vee \bigvee_{l=1}^{k} {}_l[\varphi, k]_0$
- $\_[\varphi, k]_0$ encodes that the path is acyclic and $\models_{nl} \varphi$
- ${}_l[\varphi, k]_0$ encodes that the path is (k, l)-cyclic and $\models \varphi$

LTL tricks in BMC

Fragment LTL-X

- Reduces the number of transitions (smaller SAT instance).
- Similar to partial order reduction.

For the Interested

- Keijo Heljanko: Bounded Model Checking for Finite-State Systems
  `http://users.ics.aalto.fi/kepa/qmc/slides-heljanko-2.pdf`
- Keijo Heljanko and Tommi Junttila: Advanced Tutorial on Bounded Model Checking
  `http://users.ics.aalto.fi/kepa/acsd06-atpn06-bmc-tutorial/lecture1.pdf`

Section

Conclusions on BMC

Advantages of BMC

General

- Reduces to a standard SAT problem, advances in SAT solving help with BMC.
- Often finds counterexamples of minimal length (not always).
- Boolean formulas can be more compact than OBDD representation.

Verification of HW

- Thanks to k-induction, a very successful method.

Verification of SW

- Currently, according to Software Verification Competition (TACAS 2014), BMC in connection with SMT is currently among the best software verification methods (actually falsification).

Downsides of BMC

General

- Not complete in general.
- Large SAT instances are still unsolvable.

Verification of SW

- Encoding an entire CFG as a SAT instance is currently unrealistic.
- K-induction cannot be used
  (the graph is incomplete, no back edges).
- Problems with dynamic data structure analysis.
- Loop analysis is hard.
- Inefficient for full arithmetic (partially solved by SMT).

Tools and food for thought...

Tools

- CBMC – BMC for ANSI-C.
- ESBMC – uses SMT, built on top of CBMC.
- LLBMC – BMC for LLVM bitcode.

Food for Thought...

- What differentiates modern SMT-BMC from symbolic execution?
- Boundaries are not clear.

Homework

Homework

- Study structure and results of Software Verification Competition (TACAS).