Little Engines of Proof

N. Shankar, L. de Moura, H. Ruess, A. Tiwari

shankar@csl.sri.com

URL: http://www.csl.sri.com/~shankar/LEP.html

Computer Science Laboratory

SRI International

Menlo Park, CA

Overview

Model checkers are used to verify/refute properties of transition systems.

Transition systems are used to model hardware and software.

Bounded Model Checking is a special kind of model checker.

The decision procedures described on previous lectures are used to implement this kind of model checker.

Transition Systems

Transition system $M = (S, I, T)$

S: set of states.

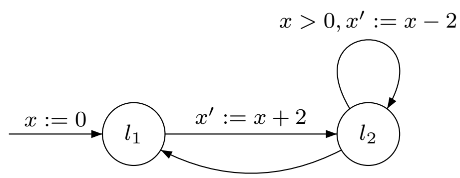
$I \subseteq S$: set of initial states. Example:

$$I(s) \quad = \quad s.x = 0 \wedge s.pc = l_1$$

$T \subseteq S \times S$: transition relation. Example:

$$\begin{aligned} T(s, s') \quad = \quad & (s.pc = l_1 \wedge s'.x = s.x + 2 \wedge s'.pc = l_2) \vee \\ & (s.pc = l_2 \wedge s.x > 0 \wedge s'.x = s.x - 2 \wedge s'.pc = l_2) \vee \\ & (s.pc = l_2 \wedge s'.x = s.x \wedge s'.pc = l_1) \end{aligned}$$

Transition Systems (cont.)



$\pi(s_0, \ldots, s_n)$ is a path iff $I(s_0)$ and $T(s_i, s_{i+1})$ for $0 \leq i < n$. Example:

$$(l_1, 0) \rightarrow (l_2, 2) \rightarrow (l_1, 2) \rightarrow (l_2, 4) \rightarrow (l_2, 2) \rightarrow (l_2, 0) \rightarrow (l_1, 0)$$

Invariants & Model Checkers

A state s_k is reachable iff there is a path $\pi(s_0, \ldots, s_k)$.

Invariants characterize properties that are true of all reachable states in a system.

Any superset of the set of reachable states is an invariant.

Example: $s.x \geq 0$.

A counterexample for an invariant φ is a path $\pi(s_0, \ldots, s_k)$ such that $\neg\varphi(s_k)$.

Model Checkers can verify/refute invariants.

There are different kinds of model checkers:

- Explicit State
- Symbolic (based on BDDs)
- Bounded (based on DP)

Bounded Model Checking: Invariants

Given.

- Transition system $M = (S, I, T)$
- Invariant φ
- Natural number k

Problem.

Is there a counterexample of length k for the invariant φ?

There is a counterexample for the invariant φ if the following formula is satisfiable:

$$I(s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge (\neg\varphi(s_1) \vee \ldots \vee \neg\varphi(s_k))$$

Bounded Model Checking (cont.)

BMC is mainly used for refutation.

Users want counterexamples. The decision procedure (DP) must be able to generate models for satisfiable formulas.

BMC is a complete method for finite systems when the diameter (longest shortest path) of the system is known.

The diameter is usually to expensive to be computed.

The recurrence diameter (longest loop-free path) is usually used as a completeness threshold.

The recurrence diameter can be much longer than the diameter.

Recurrence Diameter

A system M contains a loop-free path of length n iff

$$\pi(s_0, \ldots, s_n) \wedge \bigwedge_{0 \leq i < j \leq n} s_i \neq s_j$$

The recurrence diameter is the smallest n such that the formula above is unsatisfiable.

The diameter of infinite systems (i.e., infinite state space) may be infinite.

Verifying Invariants: Induction

An invariant is inductive if:

- $I(s) \rightarrow \varphi(s)$ (base step)
- $\varphi(s) \wedge T(s, s') \rightarrow \varphi(s')$ (inductive step)

Invariants are not usually inductive.

The inductive step is violated.

Example: $(l_2, 1) \rightarrow (l_2, -1)$

Verifying Invariants: k-Induction

An invariant φ is k-inductive if:

- $I(s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \rightarrow \varphi(s_1) \wedge \ldots \wedge \varphi(s_k)$
- $\varphi(s_1) \wedge \ldots \wedge \varphi(s_k) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_k, s_{k+1}) \rightarrow \varphi(s_{k+1})$

It is harder to violate the inductive step.

The base case is BMC.

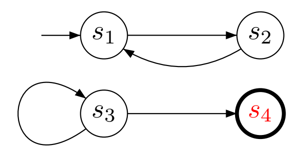
If φ is k_1-inductive then it is also k_2-inductive for $k_2 \geq k_1$.

Verifying Invariants: k-Induction (cont.)

Can be used to verify finite and infinite systems.

Not complete even for finite systems: Self-loops in unreachable states.

Example:



Bad state s_4

Counterexamples $\underbrace{s_3 \leadsto s_3 \leadsto \ldots \leadsto s_3}_{k} \leadsto s_4$

Verifying Invariants: k-Induction (cont.)

Completeness for finite systems: consider only loop-free paths.

Not complete for infinite systems. Example:

- $(l_2, 1) \rightarrow (l_2, -1)$
- $(l_2, 3) \rightarrow (l_2, 1) \rightarrow (l_2, -1)$
- $(l_2, 5) \rightarrow (l_2, 3) \rightarrow (l_2, 1) \rightarrow (l_2, -1)$
- ...

Invariant Strengthening

Failed k-induction for φ yields

Explicit counterexample

$$s_n \leadsto s_{n+1} \leadsto s_{n+k} \leadsto s_{n+k+1}$$

Assume s_n to be unreachable and strengthen $\varphi \wedge \neg(s_n)$

Symbolic counterexample described by a conjunction of constraints

$$P(s_n, s_{n+1}, \dots, s_{n+k}, s_{n+k+1})$$

- Projection

$$Q(s_n) := \exists s_{n+1}, \dots, s_{n+k+1}.\ P(s_n, s_{n+1}, \dots, s_{n+k}, s_{n+k+1})$$

- Strengthening
  $\varphi \wedge \neg qe(\{s_n, s_{n+1}, s_{n+k}\}, P(s_n, \dots, s_{n+k}, s_{n+k+1}))$

Satisfiability for Boolean Constraint Formulas

Given. Propositional formula with constraints as literals.

Assumption. DP decides the satisfiability problem for conjunctions of constraints.

Problem. Efficient satisfiability for Boolean combinations of constraints.

Experiment. Using a combination of BDDs with linear arithmetic decision procedures in PVS2.4 for finding counterexamples in a modified train-gate controller example

k = 2	70s
k = 3	8500s

$\leadsto$ new techniques required

Reduction to SAT

Let φ be a Boolean constraint formulas ($\mathsf{Bool}(C)$) with constraints c_i.

Translations.

- α replaces constraints with (fresh) propositional variables.
- γ substitutes constraints for corresponding variables.

Example.

$$\alpha(x = y \wedge f(x) = f(y)) \ \leadsto\ p \wedge q$$

A Boolean assignment ν induces a set of constraints $\gamma(\nu)$

$$\text{if } \nu = [p \mapsto 0, q \mapsto 1] \text{ then } \gamma(\nu) = \{x \neq y,\ f(x) = f(y)\}$$

Boolean Reduction Theorem

Inconsistencies. ($l_i \in \mathit{Lits}(\varphi)$)

$$\{l_1, \dots, l_n\} \in I(\varphi)$$
iff
$$\gamma(l_1) \wedge \dots \wedge \gamma(l_n) \text{ is } C\text{-inconsistent}$$

Theorem.

- $\varphi \in \mathsf{Bool}(C)$ and
- $\alpha(\varphi) \wedge (\bigwedge_{\{l_1,\dots,l_n\} \in I(\varphi)} (\neg l_1 \vee \dots \vee \neg l_n))$

are equisatisfiable

Usually, an exponential number of inconsistency tests is required.

Are there "good" enumeration strategies?

Lazy Theorem Proving

procedure lazy-th(φ)

$p := \alpha(\varphi)$;

loop

$\nu := SAT(p)$;

if $\nu = \bot$ **then return** $\bot$

if $\gamma(\nu) \not\models \bot$ **then return** $\gamma(\nu)$

else $I := \bigvee_{c \in \gamma(\nu)} \neg\alpha(c)$; $p := p \wedge I$

endloop;

return $\bot$

Optimizations. don't cares, and DP explanations.

Eager Theorem Proving

Converts a φ Boolean constraint formulas (Bool(C)) in a equisatisfiable boolean formula.

Uses a SAT solver (or BDD package) to check the satisfiability of the formula.

Uses techniques described on previous lectures.

- Small domain instantiation.
- Ackermann's trick.
- Separation constraints.

Lazy vs. Eager Theorem Proving

Eager:

- Modular implementation (it is easy to replace the SAT solver).
- Does not support richer theories (e.g., linear arithmetic).
- Translation to SAT can consume a lot of time and memory.
- Useless "information" is included in the boolean formula.
- Can use the best SAT solver.

Lazy:

- Non-modular implementation to obtain efficiency (it is not easy to replace the SAT solver).
- Commonly used SAT heuristics are inefficient in the lazy integration.
- Supports richer theories.

Lazy Quantifier Elimination

procedure qe($vars$, φ)

$\psi :=$ *false*

loop

$c :=$ *lazy-th*(φ)

if $c =$ *false* **then return** ψ

$c' := \mathcal{C}$-*qe*$(vars, c)$

$\psi := \psi \vee c'$

$\varphi := \varphi \wedge \neg c'$

Lazy convertion to DNF.

Avoids the generation of infeasible conjunctions of literals.

Uses lemma genaration capabilities found in SAT solvers and Lazy theorem provers.

Lazy Quantifier Elimination (cont.)

Example:

$$\begin{aligned}&\exists x_1, y_1.\\ &\quad [((x_0 = 1 \vee x_0 = 3 \vee y_0 > 1) \wedge x_1 = x_0 - 1 \wedge y_1 = y_0 + 1)\\ &\quad\ \vee ((x_0 = -1 \vee x_0 = -3) \wedge x_1 = x_0 + 2 \wedge y_1 = y_0 - 1)]\\ &\quad \wedge x_1 < 0\end{aligned}$$

First solution: $c := y_0 > 1 \wedge x_1 = x_0 - 1 \wedge y_1 = y_0 + 1 \wedge x_1 < 0$.

Eliminating x_1 and y_1 yields: $c' := y_0 > 1 \wedge x_0 - 1 < 0$.

Next solution: $c := x_0 = -3 \wedge x_1 = x_0 + 2 \wedge y_1 = y_0 - 1 \wedge x_1 < 0$.

Eliminating x_1 and y_1 yields: $c' := x_0 = -3 \wedge x_0 + 2 < 0$.

There are no further solutions, the result is:
$(y_0 > 1 \wedge x_0 - 1 < 0) \vee (x_0 = -3 \wedge x_0 + 2 < 0)$.

Conclusion

BMC: $depth \leq 100$ in practice.

BMC usually consumes less memory than a symbolic model checker (BDD-based).

BMC is usually very efficient for shallow bugs.

BMC is usually not affected by "irrelevant" parts (garbage) of the specification.

BMC can be "defeated" by simple examples where there is a lot of interdependency between state variables.

BMC is used also for test case generation.

Open problem: The Lazy and Eager theorem proving are not "stable" as the state-of-the-art SAT solvers. For instance, they are too sensitive to how the transition relation is specified.