Lecturecise 18
Bounded Model Checking. Reachability Graphs. Interpolation

2013

Concrete program semantics and verification

States per program point are given by $(c_1, \ldots, c_n) \in C^n$ for some concrete lattice $(C, \subseteq)$, where $C = 2^S$.
For each program there is a monotonic ω-continuous function $F : C^n \to C^n$ such that

$$\bar{c}_* = \bigcup_{n \geq 0} F^n(\emptyset, \ldots, \emptyset)$$

is the set of reachable states for each program point.
(Safety) verification can be stated as saying that the semantics remains within the set of good states G, that is $c_* \subseteq G$, or

$$\left(\bigcup_{n \geq 0} F^n(\emptyset, \ldots, \emptyset) \right) \subseteq G$$

which is equivalent to

$$\forall n.\ F^n(\emptyset, \ldots, \emptyset) \subseteq G$$

Unfolding for Counterexamples: Bounded Model Checking

$$\forall n.\ F^n(\emptyset, \ldots, \emptyset) \subseteq G$$

The above condition is false iff there exists k and $\bar{c} \in C^n$ such that

$$\bar{c} \in F^k(\emptyset, \ldots, \emptyset) \wedge \bar{c} \notin G$$

For a fixed k this can often be expressed as a quantifier-free formula.
Example: replace a loop $([c]s) * [!c]$ with finite unrolding $([c]s)^k[!c]$
Example: $n = 1$, $S = \mathbb{Z}^2$, $C = 2^S$, and $F : C \to C$ describes the program:
x=0;while(*)x=x+y

$$F(B) = \{(x, y) \mid x = 0\} \cup \{(x + y, y) \mid (x, y) \in B\}$$

We have $F(\emptyset) = \{(x, y) \mid x = 0\} = \{(0, y) \mid y \in \mathbb{Z}\}$

$$F^2(\emptyset) = \{(0, y) \mid y \in \mathbb{Z}\} \cup \{(y, y) \mid y \in \mathbb{Z}\}$$

$$F^3(\emptyset) = \{(x, y) \mid x = 0 \vee x = y \vee x = 2 * y\}$$

Formula for Bounded Model Checking

Let $P_B(x, y)$ be a formula in Presburger arithmetic such that $B = \{(x, y) \mid P_B(x, y)\}$ then the formula

$$x = 0 \vee (\exists x_0, y_0 . x = x_0 + y_0 \wedge y = y_0 \wedge P_B(x_0, y_0))$$

describes $F(B)$. Suppose the set $F^k(B)$ can be described by a PA formula P_k. If G is given by a formula P_G then the program can reach error in k steps iff

$$P_k \wedge \neg P_G$$

is satisfiable.
Suppose P_G is $x \leq y$. For $k = 3$ we obtain

$$(x = 0 \vee x = y \vee x = 2 * y) \wedge \neg(x \leq y)$$

By checking satisfiability of the formula we obtain counterexample values $x = -1, y = -2$.

Bounded Model Checking Algorithm

$B = \emptyset$
while $(*)$ $\{$
  checksat($!(B \subseteq G)$) **match**
    case Assignment(v) **=>** **return** Counterexample(v)
    case Unsat **=>**
      $B' = F(B)$
      if $(B' \subseteq B)$ **return** Valid
      else $B = B'$
$\}$

Good properties

- subsumes testing up to given depth for all possible initial states
- for a buggy program k, can be small, Leon and other tools can find many bugs fast
- a semi-decision procedure for finding all possible errors:

Bounded Model Checking is Bounded

Bad properties

- can prove correctness only if $F^{n+1}(\emptyset) = F^n(\emptyset)$
- errors after initializations of long arrays require unfolding for large n. This program requires unfolding past all loop iterations, even if the property does not depend on the loop:

```
i = 0
z = 0
while (i < 1000) {
  a(i) = 0
}
y = 1/z
```

- For large k formula F^k becomes large, so deep bugs are hard to find

Transition Relation and CFG

(V, E, L) where $L : E \to \mathit{Formula}$ and variables are Vars
Formula $T(\bar{x}, v, \bar{x}', v')$ describing one step of execution:

- from CFG node v and values of variables $\bar{x}$
- to CFG node v' and values of variables $\bar{x}'$

$$\begin{aligned} T(\bar{x}, v, \bar{x}', v') &\equiv (L(v, v'))(\bar{x}, \bar{x}') \\ &\equiv \bigvee_{(w,w')\in E} (v = w \wedge v' = w' \wedge L(w, w')(\bar{x}, \bar{x}')) \end{aligned}$$

If $I(\bar{x}, v)$ is a formula describing states reachable in some number of steps, then states reachable in one more step are given by this formula

$$\exists \bar{x}, v.\ (I(\bar{x}, v) \wedge T(\bar{x}, v, \bar{x}', v')$$

whose free variables are $\bar{x}', v'$.
Execution fragment $\bar{x}_i, v_i, \bar{x}_{i+1}, v_{i+1}, \ldots, \bar{x}_{i+k}, v_{i+k}$ is given by formula $P_{i,k}$:

$$\bigwedge_{j=0}^{k-1} T(\bar{x}_{i+j}, v_{i+j}, \bar{x}_{i+j+1}, v_{i+j+1})$$

Bounded Model Checking for Transition Relation

We have derived formula $P_{i,k}$ describing paths by iterating transition relation T

To check whether

- starting from the program entry point v_{entry} with initial variables satisfying $Init(\bar{x}_0)$
- the program can reach in k steps control flow graph point v_{error} with values of variables satisfying $Error(\bar{x})$

we check the satisfiability of the formula

$$(v_0 = v_{error} \wedge Init(\bar{x}_0)) \quad \wedge \quad P_{0,k} \quad \wedge \quad (v_k = v_{error} \wedge Error(\bar{x}_k))$$

Unfolding for Proving Correctness: k-Induction

$$\text{Goal:} \quad \forall n.\ F^n(\emptyset, \ldots, \emptyset) \subseteq G \tag{1}$$

Suppose that, for some $k \geq 1$

$$F^k(G) \subseteq G \tag{2}$$

By induction on p,

$$F^{pk}(G) \subseteq G$$

Suppose also

$$\forall q < k.\ F^q(\bar{\emptyset}) \subseteq G \tag{3}$$

By monotonicity of F^{pk} then for every $p \geq 0$ and $q < k$

$$F^{pk+q}(\bar{\emptyset}) = F^{pk}(F^q(\bar{\emptyset})) \subseteq F^{pk}(G) \subseteq G$$

Every non-negative integer can be decomposed as $pk + q$, so (1) holds.
Algorithm: check (2) and (3) for increasing k

k-induction Algorithm

Prove or find counterexample for:

$$\forall n.\ F^n(\emptyset, \ldots, \emptyset) \subseteq G$$

$Fk = F$
while $(*)$ $\{$
  checksat(!($Fk(G) \subseteq G$)) **match**
    case Unsat **=>** **return** Valid
    case Assignment(v0) **=>**
      checksat(!($Fk(\emptyset) \subseteq G$)) **match**
        case Assignment(v) **=>** **return** Counterexample(v)
        case Unsat **=>** $Fk = Fk \circ F'$ *// unfold one more*
$\}$

$F'(c)$ can be $F(c)$ or $F(c) \cap G$
Saving work: preserve the state of solver in both checksats across different k
Lucky test:
if (!($lfp(F)(initState(v0)) \subseteq G$)) return Counterexample(v0)

Divergence in k-Induction

$Fk = F$
while $(*)$ {
  checksat(!($Fk(G) \subseteq G$)) **match**
    case Unsat **=>** **return** Valid
    case Assignment(v0) **=>**
      checksat(!($Fk(\emptyset) \subseteq G$)) **match**
        case Assignment(v) **=>** **return** Counterexample(v)
        case Unsat **=>** $Fk = Fk \circ F'$ // *unfold one more*
}

Subsumes bounded model checking, so finds all counterexamples
Often cannot find proofs when $lfp(F) \subseteq G$. Then G may be too weak to be inductive, $(F')^n(G)$ may remain too weak:

$$F^n(\bar{\emptyset}) \subseteq lfp(F) \subseteq (F')^n(G)$$

Need weakening of $F^n(\emptyset)$ or strengthening of $(F')^n(G)$

Taking Approximate Postcondition

Suppose we did not find counterexample yet and we have sequence

$$c_0 \subseteq c_1 \subseteq \ldots c_k \subseteq G$$

where $c_i = F^i(\bar{\emptyset})$, so

$$F(c_i) = c_{i+1}$$

Instead of simply increasing k, we try to obtain larger values by finding another solution a_0 of constraints

$$c_0 \subseteq a_0, \quad F^{k-1}(a_0) \subseteq G$$

so we obtain a sequence

$$a_0 \subseteq F(a_0) \subseteq \ldots \subseteq F^{k-1}(a_0) \subseteq G$$

- if $F(F^{k-1}(a_0)) \subseteq F^{k-1}(a_0)$, then $F^{k-1}(a_0)$ is inductive invariant
- if $F(F^{k-1}(a_0)) \subseteq G$, repeat the process: find a new initial element a_1 by solving $a_0 \subseteq a_1$, $F^{k-1}(a_1) \subseteq G$
- if not $F(F^{k-1}(a_0)) \subseteq G$, then we "overshot" the specification G. We then increase k and restart.

Solving Inclusion Constraints

The previous procedure also finds all counterexamples of length up to k, and uses specification in a different way than k-induction.

Key question: how to obtain interesting solutions of inequality constraints

Solution: interpolation

Abstract Reachability Tree

Consider a control-flow graph (V, E, L) where $L : E \to \mathit{Formula}$ describes the statement on CFG edges using variables $\bar{x}$ and $\bar{x}'$.

Given a set of predicates $\mathcal{P}$, the complete abstract reachability graph (cARG) (V_A, E_A) for (V, E, L) and $\mathcal{P}$ is given by

- $V_A = V \times 2^{\mathcal{P}}$. Thus, each ARG node $(v, a) \in V_A$ has a CFG node $v \in V$ and a set of predicates $a \subseteq \mathcal{P}$
- $((v, a), (v', a') \in E_A$ iff
  - $(v, v') \in E$
  - $a' = \{P \in \mathcal{P} \mid \forall x, \bar{x}.\ ((\bigwedge a) \wedge L(v, v') \rightarrow P)\}$

Total number of nodes in cARG can be as much as $|V| \times 2^{|\mathcal{P}|}$

In practice:

- we construct subgraphs of cARG, exploring additional edges using some exploration strategy
- we do not use all predicates at all program points, but discover predicates on the fly, using a set of predicates specific to each cARG node

For example, given predicates $\{x \geq 0, x > 0, x = 0\}$ the successors of the node $(v_0, \emptyset)$ under a statement $L(v_0, v_1) \equiv (x' = 1)$ is $(v_1, \{x > 0\})$.

Splitting in ARG

The above exploration strategy does not discover all disjunctions of invariants.

Given an edge $(v, v') \in E$ and an ARG node (v, a) we can achieve more precise representation of the command $L(v, v')$ by introducing not one but a set of abstract edges such that

$$(\bigwedge a) \wedge L(v, v') \rightarrow \bigvee_{((v,a),(v',a')) \in E_A} a'$$

For example, given predicates $\{x < 0, x > 0, x = 0\}$ and edge $x = x + 1$, if we do not use splitting the successors of the node $(v_0, \emptyset)$ under a statement $L(v_0, v_1) \equiv x' = x + 1$ is $(v_1, \emptyset)$ because no single predicate is guaranteed to hold. On the other hand, if we are allowed to use multiple edges, we can introduce instead three edges into E_A:

$$((v_0, \emptyset), (v_1, \{x < 0\}))$$
$$((v_0, \emptyset), (v_1, \{x > 0\}))$$
$$((v_0, \emptyset), (v_1, \{x = 0\}))$$

Predicate Sequence that Eliminates False Path

ARG construction only checks feasibility of one step at a time
Therefore, ARG is finite, but also some paths can be infeasible
Consider sequence of nodes

$$v_0, v_1, \ldots, v_k$$

The condition that this path is feasible is

$$\bigwedge_{i=0}^{k-1} L(v_i, v_{i+1})[\bar{x} := \bar{x}_i, \bar{x}' := \bar{x}_i']$$

If the path is not feasible it means that the above formula is unsatisfiable.
Then there is a Hoare triple proof for it:

$$\{I_0\}\ L(v_0, v_1)\ \{I_1\}\ L(v_1, v_2)\ \{I_2\}\ \ldots \{I_{k-1}\}\ L(v_{k-1}, v_k)\ \{I_k\}$$

where $I_0 \equiv \mathit{true}$ and $I_k \equiv \mathit{false}$.
How to find such predicates $I_1, \ldots, I_n$?

Interpolation Sequence

$$\{I_0\}\ L(v_0, v_1)\ \{I_1\}\ L(v_1, v_2)\ \{I_2\}\ \ldots \{I_{k-1}\}\ L(v_{k-1}, v_k)\ \{I_k\}$$

where $I_0 \equiv \mathit{true}$ and $I_k \equiv \mathit{false}$.
Finding predicates I_j:

- define I_j as the strongest postcondition of I_0 with respect to the composition of statements $L(v_0, v_1)$, ..., $L(v_0, v_j)$.
- define I_j as the weakest precondition of false with respect to the composition of statements $L(v_j, v_{j+1})$,...,$L(v_{k-1}, k)$.
- in general, use the notion of interpolating sequence

Sequence of Length Two: Binary Interpolation

Fix some class of formulas $\mathcal{F}$ (e.g. quantifier-free formulas)
Binary interpolation for $A, B \in \mathcal{F}$ is formula $I \in \mathcal{F}$ such that, for all free variables

- $A \to I$
- $I \to B$
- I has only variables that are common for both A and B

Claim: if we can find binary interpolants, we can find interpolating sequences.
Claim: if logic has quantifier elimination, then we can find binary interpolants.