Lecture 17b:
Bounded Model Checking and k-Induction

Viktor Kunčak

Exercise: Expressing Transitive Closure Using Fixedpoint

Let S be some set of states.

Let $C = 2^{S \times S}$

Define $F : C \to C$ such that

$$\bigcup_{n \geq 0} F(\emptyset) \quad = \quad r^*$$

Exercise: Expressing Transitive Closure Using Fixedpoint

Let S be some set of states.
Let $C = 2^{S \times S}$
Define $F : C \to C$ such that

$$\bigcup_{n \geq 0} F(\emptyset) \quad = \quad r^*$$

Solution: define $F(x) = \Delta \cup (x \circ r)$
Correctness follows from this lemma: for all $n \geq 1$,

$$F^n(\emptyset) = \bigcup_{i=0}^{n-1} r^i$$

Proof by induction on n. For $n = 1$, $F(\emptyset) = \Delta = \bigcup_{i=0}^{0} r^i$.
Assume it holds for n. Then

$$F^{n+1}(\emptyset) = \Delta \cup (F^n \circ r) = \Delta \cup \left(\bigcup_{i=0}^{n-1} r^i \right) \circ r = \Delta \cup \left(\bigcup_{i=1}^{n} r^i \right) = \bigcup_{i=0}^{n} r^{i+1}$$

Review: Semantics for Mutually Recursive Procedures

Two mutually recursive procedures $r_1 = E_1(r_1, r_2), \quad r_2 = E_2(r_1, r_2)$
We can represent this as a single equation on pairs of relations:

$$(r_1, r_2) = (E_1(r_1, r_2), E_2(r_1, r_2))$$

Define $\bar{E}(r_1, r_2) = (E_1(r_1, r_2), E_2(r_1, r_2))$, let $\bar{r} = (r_1, r_2)$. We define semantics of procedures as the least solution of

$$\bar{E}(\bar{r}) = \bar{r}$$

where $(r_1, r_2) \sqsubseteq (r_1', r_2')$ means $r_1 \subseteq r_1'$ and $r_2 \subseteq r_2'$
Even though pairs of relations are not sets but pairs of sets, we can define set-like operations on them, e.g.

$$(r_1, r_2) \sqcup (r_1', r_2') = (r_1 \cup r_1', \ r_2 \cup r_2')$$

The entire theory works when we have a partial order $\sqsubseteq$ with some "good properties". (**Lattice** elements are a generalization of sets.)

Least Fixedpoint and Consequences

Two mutually recursive procedures $r_1 = E_1(r_1, r_2), \quad r_2 = E_2(r_1, r_2)$
For $E(r_1, r_2) = (E_1(r_1, r_2), E_2(r_1, r_2))$, semantics is

$$(s_1, s_2) = \bigsqcup_{i \geq 0} \bar{E}^i(\emptyset, \emptyset)$$

It follows that for any c_1, c_2 if

$$E_1(c_1, c_2) \subseteq c_1 \quad \text{and} \quad E_2(c_1, c_2) \subseteq c_2$$

then $s_1 \subseteq c_1$ and $s_2 \subseteq c_2$.

Induction-like principle: To prove that mutually recursive relations satisfy two contracts, prove those contracts for the relation body definitions in which recursive calls are replaced by those contracts.

Bounded Model Checking

Concrete program semantics and verification

For each program there is a (monotonic, ω-continuous) function $F : C^n \to C^n$ such that

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

describes 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

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

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 $([c]s)^k[!c]$
Specifically, for $n = 1$, $S = \mathbb{Z}^2$, $C = 2^S$, 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, tools can find many bugs fast
- a semi-decision procedure for finding all error inputs

Bounded Model Checking is Bounded

Bad properties

- can prove correctness only if $F^{n+1}(\emptyset) = F^n(\emptyset)$ for a finite n
- 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

k-Induction

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, for every $p \geq 1$,

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

By monotonicity of F, if $n \leq pk$ then

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

Therefore, (1) holds.
Algorithm: check (2) for increasing $k \in \{1, 2, \ldots\}$

Summary: Using F^k for Proofs and Counterexamples

Exact semantics is: $\bigcup_{n \geq 0} F^n(\bar{\emptyset})$
Specification is G
If for some k:

- $\neg(F^k(\bar{\emptyset}) \subseteq G)$ then we prove that specification **does not** hold (and there is a "k-step" execution in $G \subseteq F^k(\bar{\emptyset})$ showing this)
- $F^k(G) \subseteq G$, then we prove that specification **holds** by showing that it holds in all base cases up to k and assuming it holds for all recursive steps at depth k and deeper (k-induction)

Least fixedpoint of F^k is the same as least fixedpoint of F: $F^i(\bar{\emptyset}) \subseteq F^{ki}(\bar{\emptyset})$, so $\bigcup$ gives same result as sequences are monotonic.
Each F^k defines the program with the meaning same as F but syntactically more obvious as k grows and we unfold more.

k-induction Algorithm

For monotonic F, 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, thanks to previous checks, $F(c) \cap G$
Save work: preserve solver state in checksats across different k
Lucky test: if (!($lfp(F)(initState(v0)) \subseteq G$)) return Counterexample(v0)

Explanation for Sequences in k-Induction

$\bar{\emptyset} \subseteq F(\bar{\emptyset})$, so $F^i(\bar{\emptyset}) \subseteq F^{i+1}(\bar{\emptyset})$. We have an *ascending* sequence:

$$\bar{\emptyset} \subseteq F(\bar{\emptyset}) \subseteq F^2(\bar{\emptyset}) \subseteq \ldots \subseteq F^i(\bar{\emptyset}) \subseteq F^{i+1}(\bar{\emptyset}) \subseteq \ldots$$

In general, it need not be $G \subseteq F(G)$ nor $F(G) \subseteq G$.
Define $F'(c) = F(c) \cap G$. Clearly $F'(c) \subseteq F(c)$. Moreover,

$$c_1 \subseteq c_2 \rightarrow F'(c_1) \subseteq F'(c_2)$$

$$F'(G) = (F(G) \cap G) \subseteq G$$

So F' is monotonic and $F'(G) \subseteq G$. If $c_i = (F')^i(G)$ we have *descending* sequence:

$$G = c_0 \supseteq c_1 \supseteq c_2 \supseteq \ldots$$

Since $Fk = F \circ (F')^i$ in iteration i, algorithm checks $F(c_i) \subseteq G$, so there are more chances to succeed as i grows

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
But, it often *cannot* find proofs when $lfp(F) \subseteq G$. G may be too weak to be inductive, $(F')^n(G)$ may remain too weak:

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

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

Approximate Postconditions

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 sequence a_i satisfying $a_i \subseteq a_{i+1}$ and

$$F(a_i) \subseteq a_{i+1}$$

for $0 \leq i \leq k$, and with $a_k \subseteq G$.
$c_0 \subseteq a_0$ and, by induction, $c_i \subseteq a_i$
If $a_{i+1} = a_i$ for some i, then $F(a_i) = a_i$ so

$$lfp(F) \subseteq a_i \subseteq a_k \subseteq G$$

so we have proven $lfp(F) \subseteq G$, i.e., program satisfies spec.
We can also dually require $a_{i-1} \subseteq F(a_i)$, ensuring $a_i \subseteq F^{k-i}(G)$.