Lecture 8
More Recursion. Bounded Model Checking

Viktor Kuncak

Summary: Least Fixpoint as Meaning of Recursion

A recursive program is a recursive definition of a relation $E(r) = r$

$E(\bigcup_i B_i) = \bigcup_i E(B_i) \qquad B_0 \subseteq B_1 \subseteq \ldots \qquad \omega\text{-continuity}$

We define the intended meaning as $s = \bigcup_{i \geq 0} E^i(\emptyset)$, which satisfies $E(s) = s$ and also is the least among all relations r such that $E(r) \subseteq r$ (therefore, also the least among r for which $E(r) = r$)

We picked **least** fixpoint, so if the execution cannot terminate on a state x, then there is no x' such that $(x, x') \in s$.

This model is simple (just relations on states) though it has some limitations: let q be a program that never terminates, then

- $\rho(q) = \emptyset$ and $\rho(c \,\square\, q) = \rho(c) \cup \emptyset = \rho(c)$
  (we cannot observe optional non-termination in this model)
- also, $\rho(q) = \rho(\Delta_\emptyset)$ (assume(false)), so the absence of results due to path conditions and infinite loop are represented in the same way

Alternative: special error states for non-termination

Procedure Meaning is the Least Relation

```
def f =
  if (x > 0) {
    x = x - 1
    f
    y = y + 2
  }
```

$$
\begin{aligned}
E(r_f) = \ & (\Delta_{x \tilde{>} 0} \circ (\\
& \quad \rho(x = x - 1) \circ \\
& \quad r_f \circ \\
& \quad \rho(y = y + 2)) \\
&) \cup \Delta_{x \tilde{\leq} 0}
\end{aligned}
$$

What does it mean that $E(r) \subseteq r$?

Procedure Meaning is the Least Relation

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$E(r_f) = \Big(\Delta_{x \tilde{>} 0} \circ \big(\rho(x = x - 1) \circ r_f \circ \rho(y = y + 2)\big)\Big) \cup \Delta_{x \tilde{\leq} 0}$$

What does it mean that $E(r) \subseteq r$?
Plugging r instead of the recursive call results in something that conforms to r

Justifies modular reasoning for recursive functions

To prove that recursive procedure with body E satisfies specification r, show

- $E(r) \subseteq r$
- then because procedure meaning s is least, $s \subseteq r$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

$$((x, y), (x', y')) \in s \to y' \geq y$$

$r, s \subseteq (\mathbb{Z}^2 \times \mathbb{Z}^2)$

```
def f =
  if (x > 0) {
    x = x - 1
    f
    y = y + 2
  }
```

$$\begin{aligned} E(r_f) = \ & (\Delta_{x \tilde{>} 0} \circ (\\ & \quad \rho(x = x - 1) \circ \\ & \quad r_f \circ \\ & \quad \rho(y = y + 2)) \\ &) \cup \Delta_{x \tilde{\leq} 0} \end{aligned}$$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

$$((x, y), (x', y')) \in s \to y' \geq y$$

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$\begin{aligned} E(r_f) = \ & (\Delta_{x \tilde{>} 0} \circ (\\ & \quad \rho(x = x - 1) \circ \\ & \quad r_f \circ \\ & \quad \rho(y = y + 2)) \\ &) \cup \Delta_{x \tilde{\leq} 0} \end{aligned}$$

Solution: let specification relation be
$q = \{((x, y), (x', y')) \mid y' \geq y\}$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

$$((x,y),(x',y')) \in s \rightarrow y' \geq y$$

```
def f =
  if (x > 0) {
    x = x − 1
    f
    y = y + 2
  }
```

$$\begin{aligned} E(r_f) = \; & (\Delta_{x \tilde{>} 0} \circ (\\ & \quad \rho(x = x - 1) \circ \\ & \quad r_f \circ \\ & \quad \rho(y = y + 2)) \\ &) \cup \Delta_{x \tilde{\leq} 0} \end{aligned}$$

Solution: let specification relation be
$q = \{((x,y),(x',y')) \mid y' \geq y\}$
Prove $E(q) \subseteq q$ - given by a quantifier-free formula

Formula for Checking Specification

```
def f =
  if (x > 0) {
    x = x − 1     ← x1, y1
    f     ← y2
    y = y + 2
  }
```

Specification: $q = \{((x,y),(x',y')) \mid y' \geq y\}$

Formula to prove, generated by representing $E(q) \subseteq q$:

$$\forall x,y,x',y'. \Big\{ \Big[\exists x_1,y_1,x_2,y_2 \, (x > 0 \land x_1 = x - 1 \land y_1 = y \land y_2 \geq y_1 \land y' = y_2 + 2) \lor (\neg(x > 0) \land x' = x \land y' = y) \Big] \rightarrow y' \geq y \Big\}$$

- Because q appears as $E(q)$ and q, the condition appears twice.
- Proving this is always sound, whether or not function terminates; it talks about properties of all terminating executions (unlike e.g. Leon, we never rely on termination; relations can be partial)

Multiple Procedures

Two mutually recursive procedures $r_1 = E_1(r_1, r_2), \quad r_2 = E_2(r_1, r_2)$

Extend the approach to work 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 $\bar{r} = (r_1, r_2)$

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

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

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

$\sqcup$

The entire theory works when we have a partial order $\sqsubseteq$ with some "good properties". **Lattices** as a generalization of families of sets.

Bounded Model Checking and k-Induction

Concrete program semantics and verification

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

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

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

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

which is equivalent to

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

Unfolding for Counterexamples: Bounded Model Checking

$\bigcup_k E^k(\emptyset) \subseteq G$

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

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 $B = B'$
}

$F(B) \subseteq B$ $\quad B \subseteq G$

$F^{k+1}(\emptyset) \subseteq F^k(\emptyset)$

$F^k(\emptyset) \subseteq F^{k+1}(\emptyset)$

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) ⊆ G)) match
    case Unsat => return Valid
    case Assignment(v0) =>
      checksat(!(Fk(∅) ⊆ G)) match
        case Assignment(v) => return Counterexample(v)
        case Unsat => Fk = Fk ∘ 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: abstraction