Semantics and Verification of Loops and Recursion

Viktor Kunčak

Semantics of a Program with a Loop

Compute and simplify relation for this program:

```
x = 0
while (y > 0) {
  x = x + y
  y = y − 1
}
```

$$\rho(x=0)\circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x=x+y; y=y-1))^* \circ$$
$$\Delta_{y \tilde{\leq} 0}$$

$R(x=0)$	$x'=0 \land y'=y$
$R([y>0])$	$y'>0 \land x'=x \land y'=y$
$R([y\leq 0])$	$y'\leq 0 \land x'=x \land y'=y$
$R(\ [y>0];$ $x=x+y;$ $y=y-1)$	$y>0 \land x'=x+y \land y'=y-1$
$R((\ [y>0];$ $x=x+y;$ $y=y-1)^k), k>0$	$y-(k-1)>0\land$ $x'=x+(y+(y-1)+\cdots+y-(k-1))\land y'=y-k$ *i.e.* $y\geq k \land x'=x+k(y+y-(k-1))/2\land y'=y-k$
$R((\ [y>0];$ $x=x+y;$ $y=y-1)^*)$	$(x'=x\land y'=y)\ \lor\ \exists k>0.\ y\geq k\land x'=x+k(2y-k+1))/2\land y'=y-k$ *i.e.*, eliminating $k=y-y'$, $(x'=x\land y'=y)\lor\big(y-y'>0\land y'\geq 0\land x'=x+(y-y')(y+y'+1)/2\big)$
$R(\text{program})$	$(x'=0\land y'=y\land y'\leq 0)\lor\big(y>0\land y'=0\land x'=y(y+1)/2\big)$

Remarks on Previous Solution

Intermediate components can be more complex than final result

- they must account for all possible initial states, even those never reached in actual executions

Be careful with handling base case. The following solution:

$$y' = 0 \wedge x' = y(y+1)/2$$

is “almost correct”: it incorrectly describes behavior when the initial state has, for example, $y = -2$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in $s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

```
x = 0
while (y > 0) {
  x = x + y
  y = y − 1
}
```

$$\rho(x=0)\circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x = x + y; y = y - 1))^* \circ \Delta_{y \tilde{\leq} 0}$$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in
$s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

```
x = 0
while (y > 0) {
  x = x + y
  y = y − 1
}
```

$$\rho(x = 0) \circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x = x + y; y = y - 1))^* \circ \Delta_{y \tilde{\leq} 0}$$

$\mid\sqcap$

$\mid\cap$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in $s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

```
x = 0
while (y > 0) {
  x = x + y
  y = y – 1
}
```

$$\rho(x=0)\circ$$
$$(\Delta_{y \tilde{>} 0} \circ \rho(x=x+y; y=y-1))^* \circ \Delta_{y \tilde{\leq} 0}$$

$$\sqcap$$
$$\cap$$

$$s \circ$$
$$(s \circ s \circ s)^* \circ s$$

```
x = 0
while (y > 0) {
  val y0 = y
  havoc(x,y); assume(y ≤ y0)
}
```

$$\cap$$

$$s$$

Recursion

Example of Recursion

For simplicity assume no parameters (we can simulate them using global variables)

```
def f =
 if (x > 0) {
   if (x % 2 == 0) {
     x = x / 2;
     f;
     y = y * 2
   } else {
     x = x − 1;
     y = y + x;
     f
   }
 }
```

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

Assume recursive function call denotes some relation r_f

Need to find relation r_f such that $r_f = E(r_f)$

Simpler Example of Recursion

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

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

Simpler Example of Recursion

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

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

What is $E(\emptyset)$?

Simpler Example of Recursion

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

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

What is $E(\emptyset)$?
What is $E(E(\emptyset))$?

Simpler Example of Recursion

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

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

What is $E(\emptyset)$?
What is $E(E(\emptyset))$?
$E^k(\emptyset)$?

Review from Before: Expressions E on Relations

The law

$$E(\bigcup_{i\in I} r_i) = \bigcup_{i\in I} E(r_i)$$

holds when E is built from constant relations, r, $\circ$ and $\cup$ and if I is a set of natural numbers and r_i is an increasing sequence: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

$$E(s) = E(\bigcup_{k \geq 0} r_k) \overset{?}{=} \bigcup_{k \geq 0} E(r_k) = \bigcup_{k \geq 0} r_{k+1} = \bigcup_{k \geq 1} r_k = \emptyset \cup \bigcup_{k \geq 1} r_k = s$$

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

$$E(s) = E(\bigcup_{k \geq 0} r_k) \overset{?}{=} \bigcup_{k \geq 0} E(r_k) = \bigcup_{k \geq 0} r_{k+1} = \bigcup_{k \geq 1} r_k = \emptyset \cup \bigcup_{k \geq 1} r_k = s$$

If $E(s) = s$ we say s is a **fixed point (fixpoint)** of function E

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

- $r_0 \subseteq r_1$ because $\emptyset \subseteq \ldots$. Moreover, we showed several lectures earlier that E is monotonic
- from here it follows $r_1 \subseteq r_2$ and, by induction, $r_k \subseteq r_{k+1}$

Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

$$E(s) = E(\bigcup_{k \geq 0} r_k) \overset{?}{=} \bigcup_{k \geq 0} E(r_k) = \bigcup_{k \geq 0} r_{k+1} = \bigcup_{k \geq 1} r_k = \emptyset \cup \bigcup_{k \geq 1} r_k = s$$

If $E(s) = s$ we say s is a **fixed point (fixpoint)** of function E

We will define meaning of a recursive program as a fixpoint of the corresponding E

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f: \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Solution of $x^2 - x - 3 = x$, that is, $(x-1)^2 = 4$, i.e., $|x-1| = 2$, is $x_1 = -1$ and $x_2 = 3$

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Solution of $x^2 - x - 3 = x$, that is, $(x-1)^2 = 4$, i.e., $|x-1| = 2$, is $x_1 = -1$ and $x_2 = 3$

2. Compute the fixpoint that is smaller than all other fixpoints

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Solution of $x^2 - x - 3 = x$, that is, $(x-1)^2 = 4$, i.e., $|x-1| = 2$, is $x_1 = -1$ and $x_2 = 3$

2. Compute the fixpoint that is smaller than all other fixpoints $x_1 = -1$ is the smallest.

Union of Finite Unfoldings is the **Least** Fixpoint

C - a collection (set) of sets (e.g. sets of pairs, i.e. relations)
$E : C \longrightarrow C$ such that for $r_0 \subseteq r_1 \subseteq r_2 \dots$
we have

$$E(\bigcup_i r_i) = \bigcup_i E(r_i)$$

(This holds when E is given in terms of $\circ$ and $\cup$.) Then $s = \bigcup_i E^i(\emptyset)$ is such that

1. $E(s) = s$ (we have shown this)
2. if r is arbitrary such that $E(r) \subseteq r$ (special case: if $E(r) = r$), then $s \subseteq r$
   (we will show this fact in next slide)

Showing that the Fixpoint is Least

$$s = \bigcup_i E^i(\emptyset)$$

Now take any r such that $E(r) \subseteq r$.
We will show $s \subseteq r$, that is

$$\bigcup_i E^i(\emptyset) \subseteq r \qquad (*)$$

This means showing $E^i(\emptyset) \subseteq r$, for every i. For $i = 0$ this is just $\emptyset \subseteq r$. We proceed by induction. If $E^i(\emptyset) \subseteq r$, then by monotonicity of E

$$E(E^i(\emptyset)) \subseteq E(r) \subseteq r$$

This completes the proof of $(*)$

Summary: Least Fixpoint as Meaning of Recursion

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

We define the intended meaning as $s = \bigcup_{i \geq 0} E(\emptyset)$, 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 and c one that always does:

- $\rho(q) = \emptyset$ and $\rho(c \,\square\, q) = \rho(c) \cup \emptyset = \rho(c)$
  (program that sometimes does not terminate has the same meaning as c)
- $\rho(q) = \rho(\Delta_\emptyset)$ (assume(false)), so the absence of results due to path conditions and infinite loop are represented in the same way

Summary: Least Fixpoint as Meaning of Recursion

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

We define the intended meaning as $s = \bigcup_{i \geq 0} E(\emptyset)$, 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 and c one that always does:

- $\rho(q) = \emptyset$ and $\rho(c \,\square\, q) = \rho(c) \cup \emptyset = \rho(c)$
  (program that sometimes does not terminate has the same meaning as c)
- $\rho(q) = \rho(\Delta_\emptyset)$ (assume(false)), so the absence of results due to path conditions and infinite loop are represented in the same way

Alternative: error states for non-termination (we will not pursue this approach)

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 $E(r) \subseteq r$?

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

- $E(r) \subseteq r$
- Because procedure meaning s is least, conclude $s \subseteq r$

Proving that recursive function meets specification

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

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

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

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

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
  }
```

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

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
  }
```

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

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
    f
    y = y + 2
  }
```

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

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

$$\big((x > 0 \land x_1 = x - 1 \land y_1 = y \land y_2 \geq y_1 \land y' = y_2 + 2) \\ \lor (\neg(x > 0) \land x' = x \land y' = y)\big) \rightarrow \; y' \geq y$$

- Because q appears as $E(q)$ and q, the condition appears twice.
- Proving $f \subseteq q$ by $E(q) \subseteq q$ is always sound, whether or not function f terminates; the meaning of f talks only about properties of terminating executions (relations can be partial)

Multiple Procedures: Functions on Pairs of Relations

Two mutually recursive procedures $r_1 = E_1(r_1, r_2), \quad r_2 = E_2(r_1, r_2)$
We 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 $\bar{r} = (r_1, r_2)$. 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.)

Multiple Procedures: 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.

Replacing Calls by Contracts: Example

```
def r1 = {
  if (x % 2 == 1) {
    x = x − 1
  }
  y = y + 2
  r2
} ensuring(y > old(y))
```

```
def r2 = {
  if (x != 0) {
    x = x / 2
    r1
  }
} ensuring(y >= old(y))
```

Replacing Calls by Contracts: Example

```
def r1 = {
  if (x % 2 == 1) {
    x = x − 1
  }
  y = y + 2
  r2
} ensuring(y > old(y))
```

```
def r2 = {
  if (x != 0) {
    x = x / 2
    r1
  }
} ensuring(y >= old(y))
```

Reduces to checking these two non-recursive procedures:

```
def r1 = {
  if (x % 2 == 1) {
    x = x − 1
  }
  y = y + 2
  { val x0 = x; y0 = y
    havoc(x,y)
    assume(y >= y0) }
} ensuring(y > old(y))
```

```
def r2 = {
  if (x != 0) {
    x = x / 2
    val x0 = x; y0 = y
    havoc(x,y)
    assume(y > y0)
  }
} ensuring(y >= old(y))
```