# Lecture 7
# Loops and Recursion

Viktor Kuncak

# Loops

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

**while** $(\mathsf{x} > 0)$ $\{$
  $\mathsf{x} = \mathsf{x} - \mathsf{y}$
$\}$

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?
Let $k$ be the number of times loop executes.

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

**while** ($\mathsf{x} > 0$) {
  $\mathsf{x} = \mathsf{x} - \mathsf{y}$
}

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?
Let $k$ be the number of times loop executes.

- $k = 0$:

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

```
while (x > 0) {
  x = x - y
}
```

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?

Let $k$ be the number of times loop executes.

- $k = 0$: $x \leq 0 \land x' = x \land y' = y$

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?
Let $k$ be the number of times loop executes.

- $k = 0$: $x \leq 0 \wedge x' = x \wedge y' = y$
- $k = 1$:

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

**while** (x > 0) {
  x = x − y
}

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?
Let $k$ be the number of times loop executes.

- $k = 0$: $x \leq 0 \land x' = x \land y' = y$
- $k = 1$: $x > 0 \land x' = x - y \land y' = y \land x' \leq 0$

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?

Let $k$ be the number of times loop executes.

- $k = 0$: $x \leq 0 \wedge x' = x \wedge y' = y$
- $k = 1$: $x > 0 \wedge x' = x - y \wedge y' = y \wedge x' \leq 0$
- $k > 0$:

# Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program $L$:

```
while (x > 0) {
  x = x - y
}
```

When the loop terminates, what is the (smallest) relation $\rho(L)$ between state $(x, y)$ before loop started executing and the final state $(x', y')$?

Let $k$ be the number of times loop executes.

- $k = 0$: $x \leq 0 \wedge x' = x \wedge y' = y$
- $k = 1$: $x > 0 \wedge x' = x - y \wedge y' = y \wedge x' \leq 0$
- $k > 0$: $x > 0 \wedge x' = x - ky \wedge x' \leq 0 \wedge y' = y$

Solution:

$$(x \leq 0 \wedge x' = x \wedge y' = y) \vee$$
$$(\exists k.\ k > 0 \wedge x > 0 \wedge x' = x - ky \wedge x' \leq 0 \wedge y' = y)$$

# Heuristically Eliminating a Quantifier from formula (no longer in PA)

$$\exists k.\ k > 0 \wedge x > 0 \wedge x' = x - ky \wedge x' \leq 0 \wedge y' = y$$

# Heuristically Eliminating a Quantifier from formula (no longer in PA)

$$\exists k.\ k > 0 \land x > 0 \land x' = x - ky \land x' \leq 0 \land y' = y$$

$$\exists k.\ k > 0 \land x > 0 \land ky = x - x' \land x' \leq 0 \land y' = y$$

# Heuristically Eliminating a Quantifier from formula (no longer in PA)

$$\exists k.\ k > 0 \wedge x > 0 \wedge x' = x - ky \wedge x' \leq 0 \wedge y' = y$$

$$\exists k.\ k > 0 \wedge x > 0 \wedge ky = x - x' \wedge x' \leq 0 \wedge y' = y$$

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

# Heuristically Eliminating a Quantifier from formula (no longer in PA)

$$\exists k.\ k > 0 \wedge x > 0 \wedge x' = x - ky \wedge x' \leq 0 \wedge y' = y$$

$$\exists k.\ k > 0 \wedge x > 0 \wedge ky = x - x' \wedge x' \leq 0 \wedge y' = y$$

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

$$\exists k.\ k > 0 \wedge y > 0 \wedge x > 0 \wedge y | (x - x') \wedge k = (x - x')/y \wedge x' \leq 0 \wedge y' = y$$

Apply one-point rule to eliminate $k$

# Heuristically Eliminating a Quantifier from formula (no longer in PA)

$$\exists k.\ k > 0 \land x > 0 \land x' = x - ky \land x' \leq 0 \land y' = y$$

$$\exists k.\ k > 0 \land x > 0 \land ky = x - x' \land x' \leq 0 \land y' = y$$

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

$$\exists k.\ k > 0 \land y > 0 \land x > 0 \land y | (x - x') \land k = (x - x')/y \land x' \leq 0 \land y' = y$$

Apply one-point rule to eliminate $k$

$$((x - x')/y) > 0 \land y > 0 \land x > 0 \land y | (x - x') \land x' \leq 0 \land y' = y$$

which is also equivalent to simply

# Heuristically Eliminating a Quantifier from formula (no longer in PA)

$$\exists k.\ k > 0 \wedge x > 0 \wedge x' = x - ky \wedge x' \leq 0 \wedge y' = y$$

$$\exists k.\ k > 0 \wedge x > 0 \wedge ky = x - x' \wedge x' \leq 0 \wedge y' = y$$

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

$$\exists k.\ k > 0 \wedge y > 0 \wedge x > 0 \wedge y | (x - x') \wedge k = (x - x')/y \wedge x' \leq 0 \wedge y' = y$$

Apply one-point rule to eliminate $k$

$$((x - x')/y) > 0 \wedge y > 0 \wedge x > 0 \wedge y | (x - x') \wedge x' \leq 0 \wedge y' = y$$

which is also equivalent to simply

$$y > 0 \wedge x > 0 \wedge y | (x - x') \wedge x' \leq 0 \wedge y' = y$$

# Formula for Loop

Meaning of

```
while (x > 0) {
  x = x - y
}
```

is given by formula

$$
\begin{aligned}
&(x \leq 0 \land x' = x \land y' = y) \lor \\
&(y > 0 \land x > 0 \land y | (x - x') \land x' \leq 0 \land y' = y)
\end{aligned}
$$

# Formula for Loop

Meaning of

```
while (x > 0) {
  x = x − y
}
```

is given by formula

$$(x \leq 0 \wedge x' = x \wedge y' = y) \vee$$
$$(y > 0 \wedge x > 0 \wedge y | (x - x') \wedge x' \leq 0 \wedge y' = y)$$

What happens if initially $x > 0 \wedge y \leq 0$ ?

- in the formula
- in the program

# Integer Programs with Loops

Even if loop body is in Presburger arithmetic, the semantics of a loop need not be.

Integer programs with loops are Turing complete and can compute all computable functions.

Even if we cannot find Presburger arithmetic formula, we may be able to find

- a formula in a richer logic
- a property of the meaning of the loop
  (e.g. formula for the superset)

To help with these tasks, we give mathematical semantics of loops
Useful concept for this is transitive closure: $r^* = \bigcup_{n \geq 0} r^n$
( We may or may not have a general formula for $r^n$ or $r^*$ )

# A few more facts about relations

Let $r \subseteq S \times S$ and $\Delta = \{(x,x) \mid x \in S\}$. Then

$$\Delta \circ r = r = r \circ \Delta$$

We say that $r$ is **reflexive** iff $\forall x \in S.(x,x) \in r$.

- equivalently, reflexivity means $\Delta \subseteq r$

Relation $r$ is **transitive** iff

$$\forall x, y, z.\ ((x,y) \in r \wedge (y,z) \in r \rightarrow (x,z) \in r)$$

which is the same as saying $r \circ r \subseteq r$

# Transitive Closure of a Relation

$r \subseteq S \times S$. Define $r^0 = \Delta$ and $r^{n+1} = r \circ r^n$. Then $(x_0, x_n) \in r^n$ iff $\exists x_1, \ldots, x_{n-1}$ such that $(x_i, x_{i+1}) \in r$ for $0 \leq i \leq n-1$.
Define reflexive transitive closure of $r$ by

$$r^* = \bigcup_{n \geq 0} r^n$$

Properties that follow from the definition:

- $(x_0, x_n) \in r^*$ iff there exists $n \geq 0$ and $\exists x_1, \ldots, x_{n-1}$ such that $(x_i, x_{i+1}) \in r$ for $0 \leq i \leq n-1$ (a path in the graph $r$)
- $r^*$ is a reflexive and transitive relation
- If $s$ is a reflexive transitive relation and $r \subseteq s$, then $r^* \subseteq s$
    - $r^*$ is the smallest reflexive transitive relation containing $r$
- $(r^{-1})^* = (r^*)^{-1}$
- $r_1 \subseteq r_2$ implies $r_1^* \subseteq r_2^*$
- $r^* = \Delta \cup (r \circ r^*)$ and, likewise, $r^* = \Delta \cup (r^* \circ r)$

# Towards meaning of loops: unfolding

Loops can describe an infinite number of basic paths
(for a larger input, program takes a longer path)
Consider loop

$$L \equiv \textbf{while}(F)c$$

We would like to have

$$\begin{array}{rcl} L & \equiv & \textbf{if}\,(F)\;(c\,;L) \\ & \equiv & \textbf{if}\,(F)\;(c\,;\textbf{if}\,(F)\;(c\,;L)) \end{array}$$

For $r_L = \rho(L)$, $r_c = \rho(c)$, $\Delta_1 = \Delta_{\tilde{F}}$, $\Delta_2 = \Delta_{\tilde{\neg F}}$ we have

$$\begin{array}{rcl} r_L & = & (\Delta_1 \circ r_c \circ r_L) \cup \Delta_2 \\ & = & (\Delta_1 \circ r_c \circ ((\Delta_1 \circ r_c \circ r_L) \cup \Delta_2)) \cup \Delta_2 \\ & = & \;\Delta_2 \cup \\ & & (\Delta_1 \circ r_c) \circ \Delta_2 \cup \\ & & (\Delta_1 \circ r_c)^2 \circ r_L \end{array}$$

# Unfolding Loops

$$
\begin{aligned}
r_L \quad = \quad & \Delta_2 \cup \\
& (\Delta_1 \circ r_c) \circ \Delta_2 \cup \\
& (\Delta_1 \circ r_c)^2 \circ \Delta_2 \cup \\
& (\Delta_1 \circ r_c)^3 \circ r_L
\end{aligned}
$$

We prove by induction that for every $n \geq 0$,

$$(\Delta_1 \circ r_c)^n \circ \Delta_2 \subseteq r_L$$

So, $\bigcup_{n \geq 0} \left( (\Delta_1 \circ r_c)^n \circ \Delta_2 \right) \subseteq r_L$, that is

$$\Big( \bigcup_{n \geq 0} (\Delta_1 \circ r_c)^n \Big) \circ \Delta_2 \subseteq r_L$$

We do not wish to have unnecessary elements in relation, so we try

$$r_L = (\Delta_1 \circ r_c)^* \circ \Delta_2$$

and this does satisfy $r_L = (\Delta_1 \circ r_c \circ r_L) \cup \Delta_2$, so we define

$$\rho(\mathbf{while}(F)c) = (\Delta_{\tilde{F}} \circ \rho(c))^* \circ \Delta_{\tilde{\neg F}}$$

# Why loop semantics satisfies the condition

We defined

$$r_L = (\Delta_1 \circ r_c)^* \circ \Delta_2$$

Show that $(\Delta_1 \circ r_c \circ r_L) \cup \Delta_2$ equals $r_L$, as we expect from recursive definition of a while loop.

# Why loop semantics satisfies the condition

We defined

$$r_L = (\Delta_1 \circ r_c)^* \circ \Delta_2$$

Show that $(\Delta_1 \circ r_c \circ r_L) \cup \Delta_2$ equals $r_L$, as we expect from recursive definition of a while loop.

Using property $r^* = \Delta \cup r \circ r^*$ we have

$$\begin{aligned} r_L &= (\Delta_1 \circ r_c)^* \circ \Delta_2 \\ &= [\Delta \ \cup \ \Delta_1 \circ r_c \circ (\Delta_1 \circ r_c)^*] \circ \Delta_2 \\ &= \Delta_2 \cup [\Delta_1 \circ r_c \circ (\Delta_1 \circ r_c)^* \circ \Delta_2] \\ &= \Delta_2 \cup \Delta_1 \circ r_c \circ r_L \end{aligned}$$

# Using Loop Semantics in Example

$\rho$ of $L$:

**while** (x $>$ 0) $\{$
  x $=$ x $-$ y
$\}$

is:

# Using Loop Semantics in Example

$\rho$ of $L$:

**while** (x $>$ 0) {
  x $=$ x $-$ y
}

is:

$$(\Delta_{x \tilde{>} 0} \circ \rho(x = x - y))^* \circ \Delta_{\neg(x \tilde{>} 0)}$$

Compute each relation:

$$\begin{array}{rcl}
\Delta_{x \tilde{>} 0} & = & \{((x,y),(x,y)) \mid x > 0\} \\
\Delta_{\neg(x \tilde{>} 0)} & = & \{((x,y),(x,y)) \mid x \leq 0\} \\
\rho(x = x - y) & = & \{((x,y),(x-y,y)) \mid x,y \in \mathbb{Z}\} \\
\Delta_{x \tilde{>} 0} \circ \rho(x = x - y) & = & \\
(\Delta_{x \tilde{>} 0} \circ \rho(x = x - y))^k & = & \\
(\Delta_{x \tilde{>} 0} \circ \rho(x = x - y))^* & = & \\
\rho(L) & = &
\end{array}$$

# 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
}
```

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

| | |
|---|---|
| $R(x = 0)$<br>$R([y > 0])$<br>$R([y \leq 0])$ | $x' = 0 \wedge y' = y$<br>$y' > 0 \wedge x' = x \wedge y' = y$<br>$y' \leq 0 \wedge x' = x \wedge y' = y$ |
| $R($ $[y > 0];$<br>$x = x + y;$<br>$y = y - 1)$ | $y > 0 \wedge x' = x + y \wedge y' = y - 1$ |
| $R(($ $[y > 0];$<br>$x = x + y;$<br>$y = y - 1)^{k}), k > 0$ | $y - (k-1) > 0 \wedge$<br>$x' = x + (y + (y-1) + \cdots + y - (k-1)) \wedge y' = y - k$<br>*i.e.*<br>$y \geq k \wedge x' = x + k(y + y - (k-1))/2 \wedge y' = y - k$ |
| $R(($ $[y > 0];$<br>$x = x + y;$<br>$y = y - 1)^{*})$ | $(x' = x \wedge y' = y) \vee$<br>$\exists k > 0.$<br>$y \geq k \wedge x' = x + k(2y - k + 1))/2 \wedge y' = y - k$<br>*i.e.* $(k = y - y')$<br>$(x' = x \wedge y' = y) \vee (y - y' > 0 \wedge y' \geq 0 \wedge x' = x + (y - y')(y + y' + 1)/2)$<br>*i.e.* |
| $R(\text{program})$ | $(x' = 0 \wedge y' = y \wedge y' \leq 0) \vee (y > 0 \wedge y' = 0 \wedge x' = y(y+1)/2)$ |

# 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. This solution is "almost correct" but incorrectly describes behavior when the initial state has, for example, $y = -2$:

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

# Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics.

Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$

Suppose we only wish to show that the semantics satisfies $y' \leq y$

```
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: $r_1 \subseteq r_2 \to r_1^* \subseteq r_2^*$

Suppose we only wish to show that the semantics satisfies $y' \leq y$

```
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: $r_1 \subseteq r_2 \to r_1^* \subseteq r_2^*$

Suppose we only wish to show that the semantics satisfies $y' \leq y$

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

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

$\sqcap$ (rotated: $\sqsubseteq$) $\qquad\qquad$ $\cap$ (rotated: $\subseteq$)

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

$$\rho(x = 0)\circ$$
$$(\Delta_{y \tilde{>} 0} \circ \{(x, y, x', y') \mid y' \leq y\})^* \circ$$
$$\Delta_{y \tilde{\leq} 0}$$

# 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
  }
```

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

# Simpler Example of Recursion

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

# Simpler Example of Recursion

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

# Simpler Example of Recursion

```
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 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, for each of these cases

1. If $E(r)$ is given by an expression containing $r$ at most once.
2. $\Rightarrow$ If $E(r)$ contains $r$ any number of times, but $I$ is a set of natural numbers and $r_i$ is an increasing sequence: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$
3. If $E(r)$ contains $r$ any number of times, but $r_i, i \in I$ is a **directed family** of relations: for each $i, j$ there exists $k$ such that $r_i \cup r_j \subseteq r_k$, and $I$ is possibly uncountably infinite.

# 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

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

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

Meaning of a recursive program is 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 Least Fixpoint

$C$ - a collection (set) of sets (e.g. sets of pairs, i.e. relations)
$E : C \to C$ such that for $r_0 \subseteq r_1 \subseteq r_2 \ldots$
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 such that $E(r) \subseteq r$ (special case: if $E(r) = r$), then $s \subseteq r$ (we show this next)

# 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 $E(r) \subseteq r$ (and therefore, also the least among those $r$ 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 $(x, x') \in s$

- Let $q$ be a program that never terminates, then
- $\rho(q) = \emptyset$ and $\rho(c \,[]\, q) = \rho(c) \cup \emptyset = \rho(c)$
- also, $\rho(q) = \rho(\Delta_{\emptyset})$ (assume(false))