Semantics of Loops and Recursive Procedures

Viktor Kunčak

Properties of Program Contexts

Some Properties of Relations

$$(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$$

$$(p_1 \subseteq p_2) \rightarrow (p \circ p_1) \subseteq (p \circ p_2)$$

$$(p_1 \subseteq p_2) \wedge (q_1 \subseteq q_2) \quad \rightarrow \quad (p_1 \cup q_1) \subseteq (p_2 \cup q_2)$$

$$(p_1 \cup p_2) \circ q = (p_1 \circ q) \cup (p_2 \circ q)$$

Monotonicity of Expressions using $\cup$ and $\circ$

Consider relations that are subsets of $S \times S$ (i.e. S^2)
The set of all such relations is

$$C = \{r \mid r \subseteq S^2\}$$

Let $E(r)$ be given by any expression built from relation r and some additional relations $b_1, \ldots, b_n$, using $\cup$ and $\circ$.
Example: $E(r) = (b_1 \circ r) \cup (r \circ b_2)$
$E(r)$ is function $C \rightarrow C$, maps relations to relations
Claim: E is monotonic function on C:

$$r_1 \subseteq r_2 \rightarrow E(r_1) \subseteq E(r_2)$$

Prove of disprove.

Monotonicity of Expressions using $\cup$ and $\circ$

Consider relations that are subsets of $S \times S$ (i.e. S^2)
The set of all such relations is

$$C = \{r \mid r \subseteq S^2\}$$

Let $E(r)$ be given by any expression built from relation r and some additional relations $b_1, \ldots, b_n$, using $\cup$ and $\circ$.
Example: $E(r) = (b_1 \circ r) \cup (r \circ b_2)$
$E(r)$ is function $C \to C$, maps relations to relations
Claim: E is monotonic function on C:

$$r_1 \subseteq r_2 \to E(r_1) \subseteq E(r_2)$$

Prove of disprove.
Proof: induction on the expression tree defining E, using monotonicity properties of $\cup$ and $\circ$

Union-Distributivity of Expressions using $\cup$ and $\circ$

Claim: E distributes over unions, that is, if $r_i, i \in I$ is a family of relations,

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

Prove or disprove.

Union-Distributivity of Expressions using $\cup$ and $\circ$

Claim: E distributes over unions, that is, if $r_i, i \in I$ is a family of relations,

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

Prove or disprove.
False. Take $E(r) = r \circ r$ and consider relations r_1, r_2. The claim becomes

$$(r_1 \cup r_2) \circ (r_1 \cup r_2) = r_1 \circ r_1 \cup r_2 \circ r_2$$

that is,

$$r_1 \circ r_1 \cup r_1 \circ r_2 \cup r_2 \circ r_1 \cup r_2 \circ r_2 = r_1 \circ r_1 \cup r_2 \circ r_2$$

Taking, for example, $r_1 = \{(1,2)\}$, $r_2 = \{(2,3)\}$ we obtain

$$\{(1,3)\} = \emptyset \quad (\mathit{false})$$

Union “Distributivity” in One Direction

Lemma:

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

Union “Distributivity” in One Direction

Lemma:

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

Proof. Let $r = \bigcup_{i\in I} r_i$. Note that, for every i, $r_i \subseteq r$. We have shown that E is monotonic, so $E(r_i) \subseteq E(r)$. Since all $E(r_i)$ are included in $E(r)$, so is their union, so

$$\bigcup E(r_i) \subseteq E(r)$$

as desired.

Union-Distributivity - Refined

Does distributivity

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

hold, for each of these cases

1. If $E(r)$ is given by an expression containing r at most once?

Union-Distributivity - Refined

Does distributivity

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

hold, for each of these cases

1. If $E(r)$ is given by an expression containing r at most once? Proof: Induction on expression for $E(r)$. Only one branch of the tree may contain r. Note previous counter-example uses r twice.

Union-Distributivity - Refined

Does distributivity

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

hold, for each of these cases

1. If $E(r)$ is given by an expression containing r at most once? Proof: Induction on expression for $E(r)$. Only one branch of the tree may contain r. Note previous counter-example uses r twice.
2. If $E(r)$ contains r any number of times, but I is a set of natural numbers and r_i is an increasing sequence: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$

Union-Distributivity - Refined

Does distributivity

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

hold, for each of these cases

1. If $E(r)$ is given by an expression containing r at most once? Proof: Induction on expression for $E(r)$. Only one branch of the tree may contain r. Note previous counter-example uses r twice.
2. If $E(r)$ contains r any number of times, but I is a set of natural numbers and r_i is an increasing sequence: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$ Induction. In the previous counter-example the largest relation will contain all other $r_i \circ r_j$.
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.

Union-Distributivity - Refined

Does distributivity

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

hold, for each of these cases

1. If $E(r)$ is given by an expression containing r at most once? Proof: Induction on expression for $E(r)$. Only one branch of the tree may contain r. Note previous counter-example uses r twice.
2. If $E(r)$ contains r any number of times, but I is a set of natural numbers and r_i is an increasing sequence: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$ Induction. In the previous counter-example the largest relation will contain all other $r_i \circ r_j$.
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. Induction. Generalizes the previous case.

More on Mapping Code to Formulas

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

$$x = x + 1; \{\textbf{var } y; y = x + 3; z = x + y + z\}; x = x + z$$

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope. Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) =$

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

$$x = x + 1; \{\textbf{var } y; y = x + 3; z = x + y + z\}; x = x + z$$

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope. Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) = y' = x + 3 \land z' = 2x + 3 + z \land x' = x$

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

$$x = x + 1; \{\textbf{var } y; y = x + 3; z = x + y + z\}; x = x + z$$

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope. Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) = y' = x + 3 \land z' = 2x + 3 + z \land x' = x$

$R(\{\textbf{var } y; y = x + 3; z = x + y + z\}) =$

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

$$x = x + 1; \{\textbf{var } y; y = x + 3; z = x + y + z\}; x = x + z$$

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope. Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) = \; y' = x + 3 \land z' = 2x + 3 + z \land x' = x$

$R(\{\textbf{var } y; y = x + 3; z = x + y + z\}) = \; z' = 2x + 3 + z \land x' = x$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

$$R_V(x = t) \;=\; x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var\ y; P\}) \;=$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

$$R_V(x = t) \; = \; x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var\ y; P\}) \; = \; \exists y, y'. R_{V \cup \{y\}}(P)$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

$$R_V(x = t) \; = \; x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var\ y; P\}) \; = \; \exists y, y'. R_{V \cup \{y\}}(P)$

Exercise: express havoc(x) using var.

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

$$R_V(x = t) \; = \; x' = t \land \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var \; y; P\}) \; = \; \exists y, y'. R_{V \cup \{y\}}(P)$

Exercise: express havoc(x) using var.

$$R_V(havoc(x)) \iff R_V(\{var \; y; \; x = y\})$$

Expressing Specifications as Commands

Shorthand: Havoc Multiple Variables at Once

Variables $V = \{x_1, \ldots, x_n\}$
Translation of $R(havoc(y_1, \ldots, y_m))$:

Shorthand: Havoc Multiple Variables at Once

Variables $V = \{x_1, \ldots, x_n\}$
Translation of $R(havoc(y_1, \ldots, y_m))$:

$$\bigwedge_{v \in V \setminus \{y_1, \ldots, y_m\}} v' = v$$

Exercise: the resulting formula is the same as for:

$$havoc(y_1); \ldots; havoc(y_m)$$

Thus, the order of distinct havoc-s does not matter.

Programs and Specs are Relations

$$\begin{array}{rc} \text{program:} & x = x+2; y = x+10 \\ \text{relation:} & \{(x,y,z,x',y',z') \mid x' = x+2 \land y' = x+12 \land z' = z\} \\ \text{formula:} & x' = x+2 \land y' = x+12 \land z' = z \end{array}$$

Specification:

$$z' = z \land (x > 0 \rightarrow (x' > 0 \land y' > 0)$$

Adhering to specification is relation subset:

$$\begin{array}{rl} & \{(x,y,z,x',y',z') \mid x' = x+2 \land y' = x+12 \land z' = z\} \\ \subseteq & \{(x,y,z,x',y',z') \mid z' = z \land (x > 0 \rightarrow (x' > 0 \land y' > 0))\} \end{array}$$

Non-deterministic programs are a way of writing specifications

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \wedge x' > 0 \wedge y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \wedge y > 0)$$

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

$$z' = z \land x' > x \land y' > y$$

Corresponding program?

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

$$z' = z \land x' > x \land y' > y$$

Corresponding program?
Use local variables to store initial values.

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$

Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

$$z' = z \land x' > x \land y' > y$$

Corresponding program?

Use local variables to store initial values.

```
{ var x0; var y0;
  x0 = x; y0 = y;
  havoc(x,y);
  assume(x > x0 && y > y0)
}
```

Writing Specs Using Havoc and Assume

Global variables $V = \{x_1, \ldots, x_n\}$
Specification

$$F(x_1, \ldots, x_n, x_1', \ldots, x_n')$$

Becomes

Writing Specs Using Havoc and Assume

Global variables $V = \{x_1, \ldots, x_n\}$
Specification

$$F(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$$

Becomes

```
{ $var\ y_1, \ldots, y_n;$
  $y_1 = x_1; \ldots; y_n = x_n;$
  $havoc(x_1, \ldots, x_n);$
  $assume(F(y_1, \ldots, y_n, x_1, \ldots, x_n))$ }
```

Program Refinement and Equivalence

For two programs, define **refinement** $P_1 \sqsubseteq P_2$ iff

$$R(P_1) \rightarrow R(P_2)$$

is a valid formula.
(Some books use the opposite meaning of $\sqsubseteq$.)
As usual, $P_2 \sqsupseteq P_1$ iff $P_1 \sqsubseteq P_2$.

- $P_1 \sqsubseteq P_2$ iff $\rho(P_1) \subseteq \rho(P_2)$

Define **equivalence** $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$

- $P_1 \equiv P_2$ iff $\rho(P_1) = \rho(P_2)$

Example for $V = \{x, y\}$

$$\{var\ x0; x0 = x; havoc(x); assume(x > x0)\} \sqsupseteq (x = x + 1)$$

Proof: Use R to compute formulas for both sides and simplify.

Program Refinement and Equivalence

For two programs, define **refinement** $P_1 \sqsubseteq P_2$ iff

$$R(P_1) \rightarrow R(P_2)$$

is a valid formula.
(Some books use the opposite meaning of $\sqsubseteq$.)
As usual, $P_2 \sqsupseteq P_1$ iff $P_1 \sqsubseteq P_2$.

- $P_1 \sqsubseteq P_2$ iff $\rho(P_1) \subseteq \rho(P_2)$

Define **equivalence** $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$

- $P_1 \equiv P_2$ iff $\rho(P_1) = \rho(P_2)$

Example for $V = \{x, y\}$

$$\{var\ x0; x0 = x; havoc(x); assume(x > x0)\} \sqsupseteq (x = x + 1)$$

Proof: Use R to compute formulas for both sides and simplify.

$$x' = x + 1 \wedge y' = y \ \rightarrow\ x' > x \wedge y' = y$$

Stepwise Refinement Methodology

Start form a possibly non-deterministic specification P_0
Refine the program until it becomes deterministic and efficiently executable.

$$P_0 \sqsupseteq P_1 \sqsupseteq \ldots \sqsupseteq P_n$$

Example:

$$\begin{array}{ll} & havoc(x); assume(x > 0); havoc(y); assume(x < y) \\ \sqsupseteq & havoc(x); assume(x > 0); y = x + 1 \\ \sqsupseteq & x = 42; y = x + 1 \\ \sqsupseteq & x = 42; y = 43 \end{array}$$

In the last step program equivalence holds as well

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p \circ p_1) \subseteq (p \circ p_2)$

Theorem: if $P_1 \sqsubseteq P_2$ and $Q_1 \sqsubseteq Q_2$ then

$$(\mathit{if}\ (*) P_1\ \mathit{else}\ Q_1) \sqsubseteq (\mathit{if}\ (*) P_2\ \mathit{else}\ Q_2)$$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p \circ p_1) \subseteq (p \circ p_2)$

Theorem: if $P_1 \sqsubseteq P_2$ and $Q_1 \sqsubseteq Q_2$ then

$$(\mathit{if}\ \ (*) P_1\ \ \mathit{else}\ \ Q_1) \sqsubseteq (\mathit{if}\ \ (*) P_2\ \ \mathit{else}\ \ Q_2)$$

Version for relations: $(p_1 \subseteq p_2) \wedge (q_1 \subseteq q_2) \quad \rightarrow \quad (p_1 \cup q_1) \subseteq (p_2 \cup q_2)$

Loops

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')?

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 (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.

- $k = 0$:

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.

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

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.

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

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.

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

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.

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

Solution:

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

Heuristically Eliminating a Quantifier from formula

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

Heuristically Eliminating a Quantifier from formula

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

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

Heuristically Eliminating a Quantifier from formula

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

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

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

Heuristically Eliminating a Quantifier from formula

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

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

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

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

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

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

Heuristically Eliminating a Quantifier from formula

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

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

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

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

Formula for Loop

Meaning of

```
while (x > 0) {
  x = x − y
}
```

is given by formula

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

Formula for Loop

Meaning of

```
while (x > 0) {
  x = x − y
}
```

is given by formula

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

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

- in the formula
- in the program

Integer Programs with Loops

Integer programs with loops are Turing complete and can compute all computable functions (we can use large integers as Turing machine tape).

Even if we cannot find a closed-form integer 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^*)

Some 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

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

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

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

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

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

Unfolding Loops

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

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

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

So, $\bigcup_{n\geq 0}\big((\Delta_1 \circ r_c)^n \circ \Delta_2\big) \subseteq r_L$, 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 $r_L = \big(\Delta_1 \circ r_c \circ r_L\big) \cup \Delta_2$, so we define

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

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

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

Using Loop Semantics in Example

ρ of L:

```
while (x > 0) {
  x = x − y
}
```

is:

Using Loop Semantics in Example

ρ 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
}
```

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

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

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

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

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

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

$\sqsupseteq$ $\supseteq$

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
}
```

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

$$|\sqcap \qquad\qquad |\cap$$

```
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
  }
```

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

Simpler Example of Recursion

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

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

What is $E(\emptyset)$?

Simpler Example of Recursion

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

Simpler Example of Recursion

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

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

What 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

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

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

Meaning of a recursive program is fixpoint of the corresponding E

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \rightarrow \mathbb{R}$ 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

$$\bigcup_i E^i(\emptyset) \subseteq r \tag{$*$}$$

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 $\rho(c \square q) = \rho(c) \cup \emptyset = \rho(c)$
- also, $\rho(q) = \rho(\Delta_\emptyset)$ (assume(false))

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, 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: error states for non-termination (we will not pursue)

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

- $E(r) \subseteq r$
- 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

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

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

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

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 $E(q) \subseteq q$:

$$\begin{aligned}&\big[(x > 0 \land x_1 = x - 1 \land y_1 = y \land y_2 \geq y_1 \land y' = y_2 + 2)\\ &\lor(\neg(x > 0) \land x' = x \land y' = y)\big] \ \rightarrow\ y' \geq y\end{aligned}$$

- 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))
```