Semantics and Verification of Recursion

Viktor Kunčak

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))
```

Reminder: Loop-Free Programs as Relations

command c	$R(c)$	$\rho(c)$
$(x = t)$	$x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$	
$c_1 ; c_2$	$\exists \bar{z}.\ \ R(c_1)[\bar{x}' := \bar{z}] \wedge R(c_2)[\bar{x} := \bar{z}]$	$\rho(c_1) \circ \rho(c_2)$
if$(*)$ c_1 **else** c_2	$R(c_1) \vee R(c_2)$	$\rho(c_1) \cup \rho(c_2)$
assume$(\mathbf{F})$	$F \wedge \bigwedge_{v \in V} v' = v$	$\Delta_{S(F)}$

$\rho(x_i = t) = \{((x_1, \ldots, x_i, \ldots, x_n), (x_1, \ldots, x_i', \ldots, x_n) \mid x_i' = t\}$
$S(F) = \{\bar{x} \mid F\}, \quad \Delta_A = \{(\bar{x}, \bar{x}) \mid \bar{x} \in A\}$ (diagonal relation on A)
Δ (without subscript) is identity on entire set of states (no-op)
We always have: $\rho(c) = \{(\bar{x}, \bar{x}') \mid R(c)\}$
Shorthands:

if$(*)$ c_1 **else** c_2	$c_1 \,\square\, c_2$
assume(F)	$[F]$

Examples:

$$\textbf{if}\,(F)\ c_1\ \textbf{else}\ c_2 \ \equiv\ [F]; c_1 \ \square\ [\neg F]; c_2$$
$$\textbf{if}\,(F)\ c \ \equiv\ [F]; c \ \square\ [\neg F]$$

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 $C \to C$, 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.

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.
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

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

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: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \dots$

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.

Union-Distributivity Case of Increasing Sequence

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

for $I = \{1, 2, \ldots\}$ and $r_1 \subseteq r_2 \subseteq \ldots$
Proof is by induction on the structure of the tree for expression $E(r)$ using monotonicity property. Case $E(r) = E_1(r) \circ E_2(r)$, assuming by inductive hypothesis $E_1(\bigcup_{i \in I} r_i) = \bigcup_{i \in I} E_1(r_i)$ and $E_2(\bigcup_{i \in I} r_i) = \bigcup_{i \in I} E_2(r_i)$.
Let $r = \bigcup_{i \in I} r_i$. We know by previous monotonicity argument that $\bigcup_{i \in I} E(r_i) \subseteq E(\bigcup_{i \in I} r_i)$, so we just need to show the converse direction. Let $(x, x') \in E(r)$ be arbitrary. We need to show $(x, x') \in \bigcup_{i \in I} E(r_i)$. Since $(x, x') \in E_1(r) \circ E_2(r)$, there exists z such that $(x, z) \in E_1(r)$ and $(z, x') \in E_2(r)$. By inductive hypothesis, $(x, z) \in \bigcup_{i \in I} E_1(r_i)$ and $(z, x') \in \bigcup_{i \in I} E_2(r_i)$. By definition of union there exists i_1, i_2 such that $(x, z) \in E_1(r_{i_1})$ and $(z, x') \in E_2(r_{i_2})$. Let $j = max(i_1, i_2)$. Then $r_{i_1} \subseteq r_j$ and $r_{i_2} \subseteq r_j$, so, by monotonicity of E_1 and E_2, $(x, z) \in E_1(r_j)$ and $(z, x') \in E_2(r_j)$. Thus, $(x, x') \in E_1(r_j) \circ E_2(r_j) = E(r_j)$ so $(x, x') \in \bigcup_{i \in I} E(r_i)$ as desired.

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:

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

$R(P)$ 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:

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

$R(P)$ 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
$R(y = x + 3; z = x + y + z) = y' = x + 3 \land z' = 2x + 3 + z \land x' = x$

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

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

Example: $R_{\{x,y,z\}}(x = y + 1) = (x' = y + 1 \wedge y' = y \wedge z = z')$, so

$$R_{\{x,z\}}(\{var\ y; x = y + 1\}) \;=\; \exists y, y'.\ x' = y + 1 \wedge y' = y \wedge z = z'$$

In the last formula we can eliminate y' (the result is that $y' = y$ disappears) and then eliminate y from $x' = y + 1$ i.e. $y = x' - 1$ (over integers). Thus the formula is equivalent to $z = z'$.

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

Example: $R_{\{x,y,z\}}(x = y + 1) = (x' = y + 1 \land y' = y \land z = z')$, so

$$R_{\{x,z\}}(\{var\ y; x = y + 1\}) \;\;=\;\; \exists y, y'.\ x' = y + 1 \land y' = y \land z = z'$$

In the last formula we can eliminate y' (the result is that $y' = y$ disappears) and then eliminate y from $x' = y + 1$ i.e. $y = x' - 1$ (over integers). Thus the formula is equivalent to $z = z'$.
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,

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

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

Example: $R_{\{x,y,z\}}(x=y+1) = (x'=y+1 \wedge y'=y \wedge z=z')$, so

$$R_{\{x,z\}}(\{var\ y; x=y+1\}) \;=\; \exists y, y'.\ x'=y+1 \wedge y'=y \wedge z=z'$$

In the last formula we can eliminate y' (the result is that $y'=y$ disappears) and then eliminate y from $x'=y+1$ i.e. $y=x'-1$ (over integers). Thus the formula is equivalent to $z=z'$.
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

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

Specification:

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

Adhering to specification is relation subset:

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

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):

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

Corresponding program:

Writing Specs Using Havoc and Assume: Examples

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

Formula for relation (talks only about resulting state):

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

Corresponding program:

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

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

$$\begin{array}{l} \{\ \mathit{var}\ \ y_1, \ldots, y_n; \\ \quad y_1 = x_1; \ldots; y_n = x_n; \\ \quad \mathit{havoc}(x_1, \ldots, x_n); \\ \quad \mathit{assume}(F(y_1, \ldots, y_n, x_1, \ldots, x_n))\ \} \end{array}$$

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

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
  }

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

What is $E(\emptyset)$?

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

Omega Monotonicity

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: $r_1 \subseteq r_2 \subseteq r_3 \subseteq \ldots$

In other words: E is ω-monotonic
Hence, its least fixpoint is

$$\bigcup_{k \geq 0} E^k(\emptyset)$$

Define Meaning of Recursion as the Least Fixpoint

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

Define Meaning of Recursion as the Least Fixpoint

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
  }
```

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

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

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
  }
```

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

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 $E(q) \subseteq q$:

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

- 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))
```