More on Relations Hoare Logic

Viktor Kunčak

November 8, 2019

Exercise: Which Hoare triples are valid?

Assume all variables to be over integers.

1. $\{j = a\}$ j =j+1 $\{a = j + 1\}$

2. $\{i = j\}$ i=j+i $\{i > j\}$

3. $\{j = a + b\}$ i=b; j=a $\{j = 2 * a\}$

4. $\{i > j\}$ j=i+1; i=j+1 $\{i > j\}$

5. $\{i \mathrel{!=} j\}$ **if** i>j **then** m=i–j **else** m=j–i $\{m > 0\}$

6. $\{i = 3*j\}$ **if** i>j **then** m=i–j **else** m=j–i $\{m - 2*j = 0\}$

Review: Three Forms of Hoare Triple

Lemma: the following three conditions are equivalent:

- $\{P\}r\{Q\}$
- $P \subseteq wp(r, Q)$
- $sp(P, r) \subseteq Q$

Review: Three Forms of Hoare Triple

Lemma: the following three conditions are equivalent:

- $\{P\}r\{Q\}$
- $P \subseteq wp(r, Q)$
- $sp(P, r) \subseteq Q$

Proof. The three conditions expand into the following three formulas

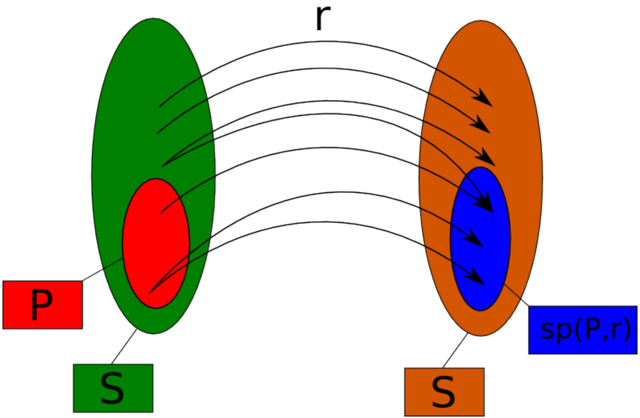
- $\forall s, s'.\ [(s \in P \land (s, s') \in r) \to s' \in Q]$
- $\forall s.\ [s \in P \to (\forall s'.(s, s') \in r \to s' \in Q)]$
- $\forall s'.\ [(\exists s.\ s \in P \land (s, s') \in r) \to s' \in Q]$

which are easy to show equivalent using basic first-order logic properties, such as $(P \land Q \longrightarrow R) \longleftrightarrow (P \longrightarrow (Q \longrightarrow R))$, $(\forall u.(A \longrightarrow B)) \longleftrightarrow (A \longrightarrow \forall u.B)$ when $u \notin FV(A)$, and $(\forall u.(A \longrightarrow B)) \longleftrightarrow ((\exists u.A) \longrightarrow B)$ when $u \notin FV(B)$.

Lemma: Characterization of sp

$sp(P,r)$ is the the smallest set Q such that $\{P\}r\{Q\}$, that is:

- $\{P\}r\{sp(P,r)\}$
- $\forall Q \subseteq S.\ \{P\}r\{Q\} \rightarrow sp(P,r) \subseteq Q$



$$\{P\}\ r\ \{Q\} \Leftrightarrow \forall s,s' \in S.(s \in P \wedge (s,s') \in r \rightarrow s' \in Q)$$

Proof of Lemma: Characterization of sp

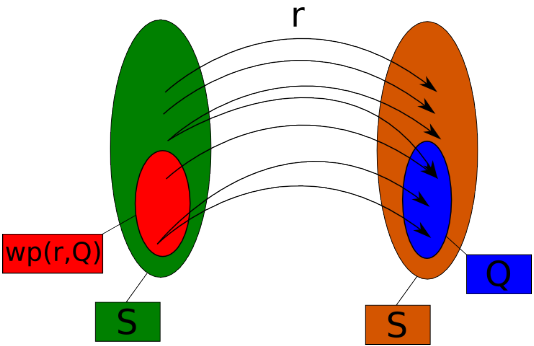
Apply Three Forms of Hoare triple. The two conditions then reduce to:

- $sp(P, r) \subseteq sp(P, r)$
- $\forall P \subseteq S.\ sp(P, r) \subseteq Q \rightarrow sp(P, r) \subseteq Q$

Lemma: Characterization of wp

$wp(r, Q)$ is the largest set P such that $\{P\}r\{Q\}$, that is:

- $\{wp(r, Q)\}r\{Q\}$
- $\forall P \subseteq S.\ \{P\}r\{Q\} \rightarrow P \subseteq wp(r, Q)$



$$\{P\}\ r\ \{Q\} \Leftrightarrow \forall s, s' \in S.(s \in P \wedge (s, s') \in r \rightarrow s' \in Q)$$

$$wp(r, Q) = \{s \mid \forall s'.(s, s') \in r \rightarrow s' \in Q\}$$

Proof of Lemma: Characterization of wp

Apply Three Forms of Hoare triple. The two conditions then reduce to:

- $wp(r, Q) \subseteq wp(r, Q)$
- $\forall P \subseteq S.\ P \subseteq wp(r, Q) \rightarrow P \subseteq wp(r, Q)$

Exercise: Postcondition of inverse versus wp

Lemma:

$$S \setminus wp(r, Q) = sp(S \setminus Q, r^{-1})$$

In other words, when instead of good states we look at the completement set of "error states", then *wp* corresponds to doing *sp* backwards.

Note that $r^{-1} = \{(y, x) \mid (x, y) \in r\}$ and is always defined.

Exercise: Postcondition of inverse versus wp

Lemma:

$$S \setminus wp(r, Q) = sp(S \setminus Q, r^{-1})$$

In other words, when instead of good states we look at the completement set of "error states", then *wp* corresponds to doing *sp* backwards.

Note that $r^{-1} = \{(y,x) \mid (x,y) \in r\}$ and is always defined.

Proof of the lemma: Expand both sides and apply basic first-order logic properties.

More Laws on Preconditions and Postconditions

Disjunctivity of sp

$$sp(P_1 \cup P_2, r) = sp(P_1, r) \cup sp(P_2, r)$$

$$sp(P, r_1 \cup r_2) = sp(P, r_1) \cup sp(P, r_2)$$

Conjunctivity of wp

$$wp(r, Q_1 \cap Q_2) = wp(r, Q_1) \cap wp(r, Q_2)$$

$$wp(r_1 \cup r_2, Q) = wp(r_1, Q) \cap wp(r_2, Q)$$

Pointwise wp

$$wp(r, Q) = \{s \mid s \in S \wedge sp(\{s\}, r) \subseteq Q\}$$

Pointwise sp

$$sp(P, r) = \bigcup_{s \in P} sp(\{s\}, r)$$

Hoare Logic for Loop-free Code

Expanding Paths

The condition

$$\{P\} \Big(\bigcup_{i \in J} r_i \Big) \{Q\}$$

is equivalent to

$$\forall i. i \in J \rightarrow \{P\} r_i \{Q\}$$

Proof: By definition, or use that the first condition is equivalent to $sp(P, \bigcup_{i \in J} r_i) \subseteq Q$ and $\{P\} r_i \{Q\}$ to $sp(P, r_i) \subseteq Q$

Transitivity

If $\{P\} s_1 \{Q\}$ and $\{Q\} s_2 \{R\}$ then also $\{P\} s_1 \circ s_2 \{R\}$.
We write this as the following inference rule:

$$\frac{\{P\} s_1 \{Q\}, \quad \{Q\} s_2 \{R\}}{\{P\} s_1 \circ s_2 \{R\}}$$

Hoare Logic for Loops

The following inference rule holds:

$$\frac{\{P\}s\{P\}, \quad n \geq 0}{\{P\}s^n\{P\}}$$

Proof is by transitivity.
By Expanding Paths condition, we then have:

$$\frac{\{P\}s\{P\}}{\{P\}\bigcup_{n\geq 0} s^n\{P\}}$$

In fact, $\bigcup_{n\geq 0} s^n = s^*$, so we have

$$\frac{\{P\}s\{P\}}{\{P\}s^*\{P\}}$$

This is the rule for non-deterministic loops.

Loops with Conditions

Note that $\{P\}\ assume(b)\ \{P \cap b_s\}$

Define $\rho(while(b)c) = (\Delta_{b_s} \circ r)^* \circ \Delta_{(\neg b)_s}$ where $r = \rho(c)$.

From the rule for non-deterministic loops we have:

$$\frac{\{P\}\Delta_{b_s} \circ r\{P\}}{\{P\}(\Delta_{b_s} \circ r)*\{P\}}$$

We can thus show:

$$\frac{\dfrac{\{P \cap b_s\}\ r\ \{P\}}{\{P\}\ \Delta_{b_s}\ \{P \cap b_s\}\ r\ \{P\}}}{\{P\}\ (\Delta_{b_s} \circ r)^*\ \{P\}\ \Delta_{(\neg b)_s}\ \{P \cap (\neg b)_s\}}$$

i.e.

$$\frac{\{P \cap b_s\}\ r\ \{P\}}{\{P\}\ \underbrace{(\Delta_{b_s} \circ r)^* \circ \Delta_{(\neg b)_s}}_{\rho(while(b)c)}\ \{P \cap (\neg b)_s\}}$$

if we use formulas and commands instead of sets and relations:

$$\frac{\{P \wedge b\}c\{P\}}{\{P\}while(b)c\{P \wedge \neg b\}}$$

Exercise

We call a relation $r \subseteq S \times S$ functional if $\forall x, y, z \in S.(x, y) \in r \wedge (x, z) \in r \rightarrow y = z$. For each of the following statements either give a counterexample or prove it. In the following, $Q \subseteq S$.

(i) for any r, $wp(r, S \setminus Q) = S \setminus wp(r, Q)$

(ii) if r is functional, $wp(r, S \setminus Q) = S \setminus wp(r, Q)$

(iii) for any r, $wp(r, Q) = sp(Q, r^{-1})$

(iv) if r is functional, $wp(r, Q) = sp(Q, r^{-1})$

(v) for any r, $wp(r, Q_1 \cup Q_2) = wp(r, Q_1) \cup wp(r, Q_2)$

(vi) if r is functional, $wp(r, Q_1 \cup Q_2) = wp(r, Q_1) \cup wp(r, Q_2)$

(vii) for any r, $wp(r_1 \cup r_2, Q) = wp(r_1, Q) \cup wp(r_2, Q)$

(viii) Alice has a conjecture: For all sets S and relations $r \subseteq S \times S$ it holds:

$$\Big(S \neq \emptyset \wedge dom(r) = S \wedge \Delta_S \cap r = \emptyset\Big) \rightarrow \Big(r \circ r \cap ((S \times S) \setminus r) \neq \emptyset\Big)$$

where $\Delta_S = \{(x, x) \mid x \in S\}$, $dom(r) = \{x \mid \exists y.(x, y) \in r\}$. She tried many sets and relations and did not find any counterexample. Is her conjecture true? If so, prove it; if false, provide a counterexample for which S is as small as possible.

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 $\circ$

For a program with k integer variables, $S = \mathbb{Z}^k$
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 ∪ and ∘

For a program with k integer variables, $S = \mathbb{Z}^k$
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

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

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.

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: $z = x + y + z$, $R(c)$ 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:

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

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

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

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: $z = x + y + z$,
$R(c)$ 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,

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

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

Exercise: express havoc(x) using var.

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

Expressing Specifications as Commands

Shorthand: Havoc Multiple Variables at Once

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

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):

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

Corresponding program:

Writing Specs Using Havoc and Assume: Examples

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

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

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

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

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:

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

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