Converting Imperative Programs to Formulas

Viktor Kunčak

Verifying Imperative Programs

```
import stainless.lang._
import stainless.lang.StaticChecks._
case class FirstExample(var x: BigInt, var y: BigInt) {
  def increase : Unit = {
    x = x + 2
    y = x + 10
  }.ensuring(_ => old(this).x > 0 ==> (x > 0 && y > 0))
}
```

Verification-Condition Generation for Imperative Non-Deterministic Programs

A program fragment can be represented by a formula relating initial and final state. Consider a program with variables x, y

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

Specification: $\mathit{old}(x) > 0 \rightarrow x > 0 \land y > 0$
Adhering to specification is relation subset:

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

or validity of the following implication:

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

Construction Formula that Describe Relations

c - imperative command

$R(c)$ - formula describing relation between initial and final states of execution of c

If $\rho(c)$ describes the relation, then $R(c)$ is formula such that

$$\rho(c) = \{(\bar{v}, \bar{v}') \mid R(c)\}$$

$R(c)$ is a formula between unprimed variables $\bar{v}$ and primed variables $\bar{v}'$

Formula for Assignment

$$x = t$$

Formula for Assignment

$$x = t$$

$R(x = t)$:

$$x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Note that the formula must explicitly state which variables remain the same (here: all except x). Otherwise, those variables would not be constrained by the relation, so they could take arbitrary value in the state after the command.

Formula for if-else

After flattening,

$$\textit{if}\,(b)\ c_1\ \textit{else}\ c_2$$

Formula for if-else

After flattening,

$$\mathit{if}\,(b)\ c_1\ \mathit{else}\ c_2$$

$R(\mathit{if}\,(b)\ c_1\ \mathit{else}\ c_2)$:

$$(b \wedge R(c_1)) \vee (\neg b \wedge R(c_2))$$

Command semicolon

$$c_1 ; c_2$$

Command semicolon

$$c_1; c_2$$

Reminder about relation composition and its definition:

$$r_1 \circ r_2 = \{(a, c) \mid \exists b.(a, b) \in r_1 \wedge (b, c) \in r_2\}$$

Command semicolon

$$c_1; c_2$$

Reminder about relation composition and its definition:

$$r_1 \circ r_2 = \{(a, c) \mid \exists b.(a, b) \in r_1 \land (b, c) \in r_2\}$$

What are $R(c_1)$ and $R(c_2)$ and in terms of which variables they are expressed?

Command semicolon

$$c_1; c_2$$

Reminder about relation composition and its definition:

$$r_1 \circ r_2 = \{(a, c) \mid \exists b.(a, b) \in r_1 \wedge (b, c) \in r_2\}$$

What are $R(c_1)$ and $R(c_2)$ and in terms of which variables they are expressed?
$R(c_1; c_2) \equiv$

$$\exists \bar{z}. \;\; R(c_1)[\bar{x}' := \bar{z}] \wedge R(c_2)[\bar{x} := \bar{z}]$$

where $\bar{z}$ are freshly picked names of intermediate states.

- a useful convention: $\bar{z}$ refer to position in program source code

havoc

Definition of HAVOC

1. wide and general destruction: devastation
2. great confusion and disorder

Example of use:

y = 12; havoc(x); assume(x + x = y)

Translation, $R(havoc(x))$:

havoc

Definition of HAVOC

1. wide and general destruction: devastation
2. great confusion and disorder

Example of use:

y = 12; havoc(x); assume(x + x = y)

Translation, $R(havoc(x))$:

$$\bigwedge_{v \in V \setminus \{x\}} v' = v$$

This again illustrates "politically correct" approach to describing the destruction of values of variables: just do not mention them.

Non-deterministic choice

$$\textit{if}\,(*)\ c_1\ \textit{else}\ c_2$$

Non-deterministic choice

$$R(\mathit{if}\,(*)\ c_1\ \mathit{else}\ c_2): \quad \frac{\mathit{if}\,(*)\ c_1\ \mathit{else}\ c_2}{R(c_1) \vee R(c_2)}$$

- translation is simply a disjunction – this is why construct is interesting
- corresponds to branching in control-flow graphs

assume

$$assume(F)$$

assume

$$assume(F)$$

$R(assume(F))$:

$$F \wedge \bigwedge_{v \in V} v' = v$$

assume

$$assume(F)$$

$R(assume(F))$:

$$F \land \bigwedge_{v \in V} v' = v$$

- This command does not change any state.

assume

$$assume(F)$$

$R(assume(F))$:

$$F \wedge \bigwedge_{v \in V} v' = v$$

- This command does not change any state.
- If F does not hold, it stops with "instantaneous success".

Example of Translation

0

$(if\ (b)\ x = x + 1\ else\ y = x + 2);$

1

$x = x + 5;$

2

$(if\ (*)\ y = y + 1\ else\ x = y)$

3

becomes

$$\begin{aligned}\exists x_1, y_1, x_2, y_2.\ &((b \wedge \mathbf{x_1} = \mathbf{x} + \mathbf{1} \wedge y_1 = y) \vee (\neg b \wedge x_1 = x \wedge \mathbf{y_1} = \mathbf{x} + \mathbf{2}))\\ &\wedge (\mathbf{x_2} = \mathbf{x_1} + \mathbf{5} \wedge y_2 = y_1)\\ &\wedge ((x' = x_2 \wedge \mathbf{y'} = \mathbf{y_2} + \mathbf{1}) \vee (\mathbf{x'} = \mathbf{y_2} \wedge y' = y_2))\end{aligned}$$

Think of execution trace $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$ where

- (x_0, y_0) is denoted by (x, y)
- (x_3, y_3) is denoted by (x', y')

Justifying the name for assume(F)

Compute and simplify as much as possible each of the following expressions:

1. $R(assume(F); c)$

Justifying the name for assume(F)

Compute and simplify as much as possible each of the following expressions:

1. $R(assume(F); c) \quad = \quad F \wedge R(c)$
2. $R(c; assume(F))$

Justifying the name for assume(F)

Compute and simplify as much as possible each of the following expressions:

1. $R(assume(F); c) \quad = \quad F \wedge R(c)$
2. $R(c; assume(F)) \quad = \quad R(c) \wedge F[\bar{x} := \bar{x}']$
   where $F[\bar{x} := \bar{x}']$ denotes F with all variables replaced with primed versions

Expressing **if** through non-deterministic choice and assume

Expressing **if** through non-deterministic choice and assume

if (b) c1 **else** c2

$|||$

if $(*)$ $\{$
  assume(b);
  c1
$\}$ **else** $\{$
  assume(!b);
  c2
$\}$

Indeed, apply translation to both sides and observe that generated formulas are equivalent.

Expressing assignment through havoc and assume

Expressing assignment through havoc and assume

x = e

|||

havoc(x);
assume(x == e)

Under what conditions this holds?

Expressing assignment through havoc and assume

x = e

|||

havoc(x);
assume(x == e)

Under what conditions this holds?
$x \notin FV(e)$

Illustration of the problem: $havoc(x);\ assume(x == x + 1)$

Expressing assignment through havoc and assume

x = e

|||

havoc(x);
assume(x == e)

Under what conditions this holds?
$x \notin FV(e)$

Illustration of the problem: $havoc(x);\ assume(x == x + 1)$

Luckily, we can rewrite it into $x_{fresh} = x + 1; x = x_{fresh}$

Loop-Free Programs as Relations: Summary

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(**F**)	$F \wedge \bigwedge_{v \in V} v' = v$	$\Delta_{S(F)}$

$\rho(v_i = t) = \{((v_1, \ldots, v_i, \ldots, v_n), (v_1, \ldots, v_i', \ldots, v_n) \mid v_i' = t\}$
$S(F) = \{\bar{v} \mid F\}, \quad \Delta_A = \{(\vec{v}, \vec{v}) \mid \vec{v} \in A\}$ (diagonal relation on A)
Δ (without subscript) is identity on entire set of states (no-op)
We always have: $\rho(c) = \{(\bar{v}, \bar{v}') \mid R(c)\}$
Shorthands:

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

Examples:

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

Program Paths

Loop-Free Programs

c - a loop-free program whose assignments, havocs, and assumes are $c_1, \ldots, c_n$

The relation $\rho(c)$ is of the form $E(\rho(c_1), \ldots, \rho(c_n))$; it composes meanings of $c_1, \ldots, c_n$ using union ($\cup$) and composition ($\circ$)

```
(if (x > 0)
   x = x − 1
else
   x = 0
);
(if (y > 0)
   y = y − 1
else
   y = x + 1
)
```

```
([x > 0]; x = x − 1
  □
 ([¬(x>0)]; x = 0)
);
([y > 0]; y = y − 1
  □
 [¬(y>0)]; y = x+1
)
```

$$\big(\Delta_{S(x>0)} \circ \rho(x = x - 1) \cup \Delta_{S(\neg(x>0))} \circ \rho(x = 0)\big) \circ \big(\Delta_{S(y>0)} \circ \rho(y = y - 1) \cup \Delta_{S(\neg(y>0))} \circ \rho(y = x + 1)\big)$$

Note: $\circ$ binds stronger than $\cup$, so $r \circ s \cup t = (r \circ s) \cup t$

Normal Form for Loop-Free Programs

Composition distributes through union:

$$(r_1 \cup r_2) \circ (s_1 \cup s_2) = r_1 \circ s_1 \ \cup \ r_1 \circ s_2 \ \cup \ r_2 \circ s_1 \ \cup \ r_2 \circ s_2$$

Example corresponding to two if-else statements one after another:

$$\begin{array}{l}
\big(\Delta_1 \circ r_1 \\
\quad \cup \\
\ \Delta_2 \circ r_2 \\
\big)\circ \\
\big(\Delta_3 \circ r_3 \\
\quad \cup \\
\ \Delta_4 \circ r_4 \\
\big)
\end{array}
\qquad \equiv \qquad
\begin{array}{l}
\Delta_1 \circ r_1 \circ \Delta_3 \circ r_3 \cup \\
\Delta_1 \circ r_1 \circ \Delta_4 \circ r_4 \cup \\
\Delta_2 \circ r_2 \circ \Delta_3 \circ r_3 \cup \\
\Delta_2 \circ r_2 \circ \Delta_4 \circ r_4
\end{array}$$

Sequential composition of basic statements is called basic path.
Loop-free code describes finitely many (exponentially many) paths.

Properties of Program Contexts

Some Properties of Relations

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

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

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

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

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 $\cup$ 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.
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

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

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.

About Strength and Weakness

Putting Conditions on Sets Makes them Smaller

Let P_1 and P_2 be formulas ("conditions") whose free variables are among $\bar{x}$. Those variables may denote program state.
When we say "condition P_1 is stronger than condition P_2" it simply means

$$\forall \bar{x}.\ (P_1 \rightarrow P_2)$$

- if we know P_1, we immediately get (conclude) P_2
- if we know P_2 we need not be able to conclude P_1

Stronger condition = smaller set: if P_1 is stronger than P_2 then

$$\{\bar{x} \mid P_1\} \subseteq \{\bar{x} \mid P_2\}$$

- strongest possible condition: "false" $\leadsto$ smallest set: $\emptyset$
- weakest condition: "true" $\leadsto$ biggest set: set of all tuples

Hoare Triples

About Hoare Logic

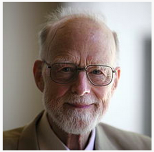
We have seen how to translate programs into relations. We will use these relations in a proof system called Hoare logic. Hoare logic is a way of inserting annotations into code to make proofs about (imperative) program behavior simpler.

Example proof:

```
//{0 <= y}
i = y;
//{0 <= y & i = y}
r = 0;
//{0 <= y & i = y & r = 0}
while //{r = (y−i)∗x & 0 <= i}
 (i > 0) (
   //{r = (y−i)∗x & 0 < i}
   r = r + x;
   //{r = (y−i+1)∗x & 0 < i}
   i = i − 1
   //{r = (y−i)∗x & 0 <= i}
)
//{r = x ∗ y}
```

Hoare Triple and Friends

Sir Charles Antony Richard Hoare



Sir Charles Antony Richard Hoare giving a conference at the EPFL on 20 June 2011

Born 11 January 1934

$P, Q \subseteq S \qquad r \subseteq S \times S$

Hoare Triple:

$$\{P\}\ r\ \{Q\} \iff \forall s, s' \in S.\, \big(s \in P \land (s, s') \in r \rightarrow s' \in Q\big)$$

$\{P\}$ does not denote a singleton set containing P but is just a notation for an "assertion" around a command. Likewise for $\{Q\}$.

Strongest postcondition:

$$sp(P, r) = \{s' \mid \exists s.\, s \in P \land (s, s') \in r\}$$

Weakest precondition:

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

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 $\{m-2*j=0\}$

Postconditions and Their Strength

What is the relationship between these postconditions?

$$\{x = 5\} \quad x := x + 2 \quad \{\mathbf{x > 0}\}$$

$$\{x = 5\} \quad x := x + 2 \quad \{\mathbf{x = 7}\}$$

Postconditions and Their Strength

What is the relationship between these postconditions?

$$\{x = 5\} \quad x := x + 2 \quad \{\mathbf{x > 0}\}$$

$$\{x = 5\} \quad x := x + 2 \quad \{\mathbf{x = 7}\}$$

- weakest conditions (predicates) correspond to largest sets
- strongest conditions (predicates) correspond to smallest sets

that satisfy a given property.

(Graphically, a stronger condition $x > 0 \wedge y > 0$ denotes one quadrant in plane, whereas a weaker condition $x > 0$ denotes the entire half-plane.)

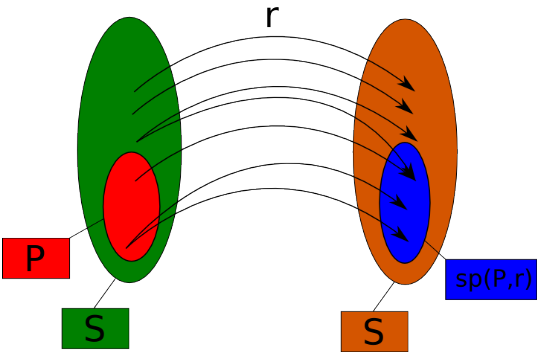
Strongest Postconditions

Strongest Postcondition

Definition: For $P \subseteq S$, $r \subseteq S \times S$,

$$sp(P, r) = \{s' \mid \exists s. s \in P \wedge (s, s') \in r\}$$

This is simply the relation image of a set.



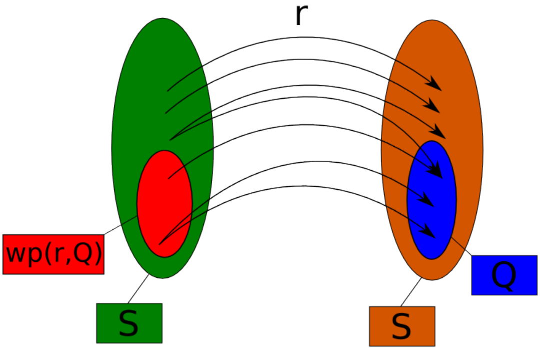
Weakest Preconditions

Weakest Precondition

Definition: for $Q \subseteq S$, $r \subseteq S \times S$,

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

Note that this is in general not the same as $sp(Q, r^{-1})$ when then relation is non-deterministic or partial.



Three Forms of Hoare Triple

Lemma: the following three conditions are equivalent:

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

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

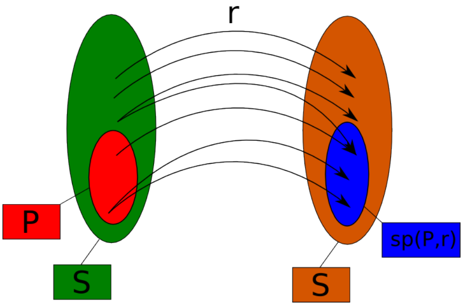
- $\forall s, s'. \ [(s \in P \land (s, s') \in r) \rightarrow s' \in Q]$
- $\forall s. \ [s \in P \rightarrow (\forall s'.(s, s') \in r \rightarrow s' \in Q)]$
- $\forall s'. \ [(\exists s. \ s \in P \land (s, s') \in P) \rightarrow s' \in Q]$

which are easy to show equivalent using basic first-order logic properties.

Lemma: Characterization of sp

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

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



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

$$sp(P, r) = \{s' \mid \exists s. s \in P \wedge (s, s') \in r\}$$

Proof of Lemma: Characterization of sp

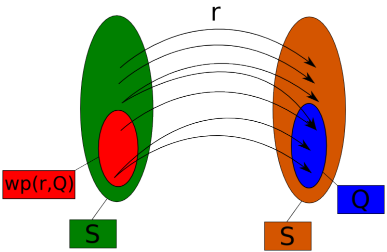
Apply Three Forms of Hoare triple. The two conditions then reduce to:

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

Lemma: Characterization of wp

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

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



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

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

Proof of Lemma: Characterization of wp

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

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

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

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

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.