Lecturecise 5
Paths, Triples, Postconditions, Preconditions

Viktor Kuncak

Loop-Free Programs as Relations: Summary

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

$\rho(v_i = t) = \{((v_1, \ldots, v_i, \ldots, v_n), (v_1, \ldots, v_i', \ldots, v_n) \mid v_i' = t\}$
$S(F) = \{\bar{v} \mid F\}, \quad \Delta_A = \{(\vec{v}, \vec{v}) \mid \vec{v} \in A\}$ (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:

$\textbf{if}(*)\ c_1\ \textbf{else}\ c_2$	$c_1 \,[]\, c_2$
$\textbf{assume}(F)$	$[F]$

Examples:

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

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

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:

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

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

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

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

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

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

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.

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

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

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

Prove or disprove.

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

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

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

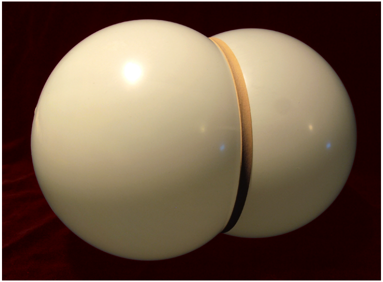
- if we know P_1, we immediately get (conclude) P_2
- if we know P_2 we need not be able to conclude P_1

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

Intuition?



Conditions “squeze” sets, making them smaller?

- perhaps better rely on logic and set theory than intuition

Hoare Triples

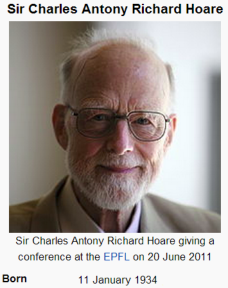
About Hoare Logic

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



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

Hoare Triple

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

Strongest postcondition:

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

Weakest precondition:

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

Hoare triples for Sets and Relations

When $P, Q \subseteq S$ (sets of states) and $r \subseteq S \times S$ (relation on states, command semantics) then the Hoare triple

$$\{P\}\ r\ \{Q\}$$

means

$$\forall s, s' \in S. \left(s \in P \wedge (s, s') \in r \rightarrow s' \in Q\right)$$

We call P precondition and Q postcondition.

The Hoare triple provides only a *partial correctness* guarantee, i.e. if P holds initially, and r executes and terminates, then Q must hold. If r does not terminate, then no guarantees on Q are provided.

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.)

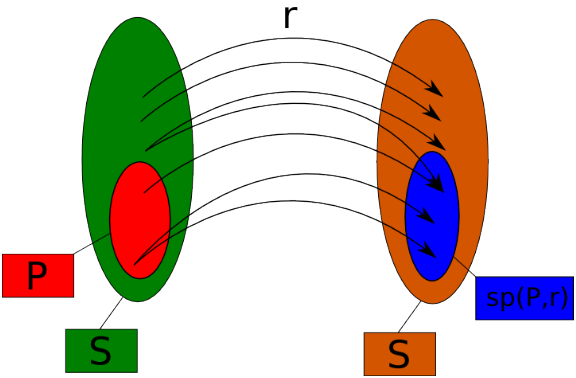
Strongest Postconditions

Strongest Postcondition

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

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

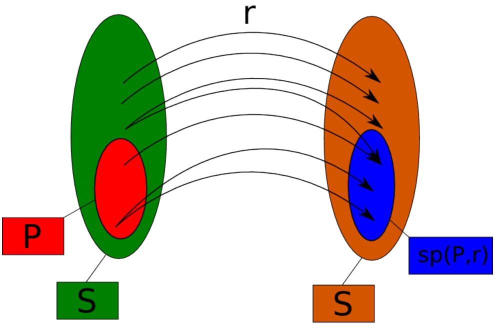
This is simply the relation image of a set.



Lemma: Characterization of sp

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

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



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

Weakest Preconditions

Backward Propagation of Errors

If we have a relation r and a set of errors E, we can check if a program meets its specification by checking:

$$sp(P, r) \cap E = \emptyset$$

Backward Propagation of Errors

If we have a relation r and a set of errors E, we can check if a program meets its specification by checking:

$$sp(P, r) \cap E = \emptyset$$

$$\forall y. \neg (y \in sp(P, r) \wedge y \in E)$$

$$\forall y.\ \neg ((\exists x. P(x) \wedge (x, y) \in r) \wedge y \in E)$$

Backward Propagation of Errors

If we have a relation r and a set of errors E, we can check if a program meets its specification by checking:

$$sp(P, r) \cap E = \emptyset$$

$$\forall y. \neg (y \in sp(P, r) \wedge y \in E)$$

$$\forall y.\ \neg ((\exists x. P(x) \wedge (x, y) \in r) \wedge y \in E)$$

$$\forall y.\ \neg \exists x. (P(x) \wedge (x, y) \in r \wedge y \in E)$$

$$\forall x, y.\ \neg (x \in P \wedge (x, y) \in r \wedge y \in E)$$

Backward Propagation of Errors

If we have a relation r and a set of errors E, we can check if a program meets its specification by checking:

$$sp(P, r) \cap E = \emptyset$$

$$\forall y. \neg(y \in sp(P, r) \wedge y \in E)$$

$$\forall y.\ \neg((\exists x. P(x) \wedge (x, y) \in r) \wedge y \in E)$$

$$\forall y.\ \neg\exists x. (P(x) \wedge (x, y) \in r \wedge y \in E)$$

$$\forall x, y.\ \neg(x \in P \wedge (x, y) \in r \wedge y \in E)$$

$$\forall x, y.\ \neg(x \in P \wedge (y, x) \in r^{-1} \wedge y \in E)$$

$$\forall x, y.\ \neg(y \in E \wedge (y, x) \in r^{-1} \wedge x \in P)$$

Backward Propagation of Errors

If we have a relation r and a set of errors E, we can check if a program meets its specification by checking:

$$sp(P, r) \cap E = \emptyset$$

$$\forall y. \neg(y \in sp(P, r) \wedge y \in E)$$

$$\forall y.\ \neg((\exists x. P(x) \wedge (x, y) \in r) \wedge y \in E)$$

$$\forall y.\ \neg\exists x. (P(x) \wedge (x, y) \in r \wedge y \in E)$$

$$\forall x, y.\ \neg(x \in P \wedge (x, y) \in r \wedge y \in E)$$

$$\forall x, y.\ \neg(x \in P \wedge (y, x) \in r^{-1} \wedge y \in E)$$

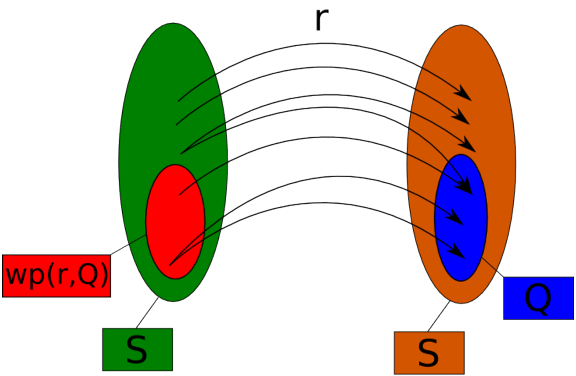
$$\forall x, y.\ \neg(y \in E \wedge (y, x) \in r^{-1} \wedge x \in P)$$

$$sp(E, r^{-1}) \cap P = \emptyset$$

$$P \subseteq sp(E, r^{-1})^c$$

In other words, we obtain an upper bound on the set of states P from which we do not reach error. We next introduce the notion of weakest precondition, which allows us to express $sp(E, r^{-1})$ from Q given as complement of error states E.

Weakest Precondition

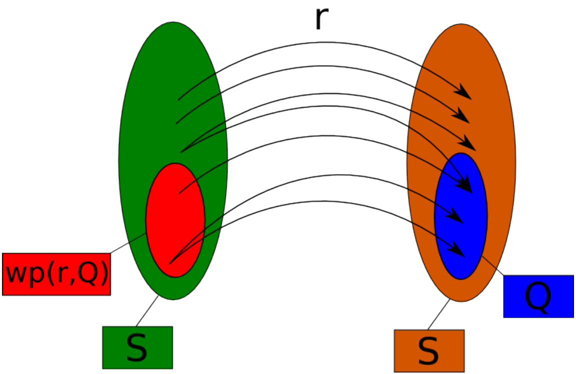


Weakest Precondition

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

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

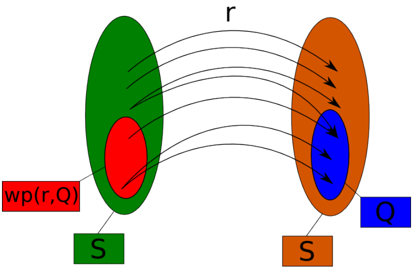
Note that this is in general not the same as $sp(Q, r^{-1})$ when then relation is non-deterministic or partial.



Lemma: Characterization of wp

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

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



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

Exercise: Postcondition of inverse versus wp

Using definitions of Hoare triple, sp, wp in Hoare logic, prove the following: If instead of good states we look at the completement set of "error states", then *wp* corresponds to doing *sp* backwards. In other words, we have the following:

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

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

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

Exercise: Three Forms of Hoare Triple

Show the following:
The following three conditions are equivalent:

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

Hoare Logic for Loop-free Code

Expanding Paths
The condition

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

is equivalent to

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

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:

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

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, assume $Q \subset S$.

(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 the following conjecture: For all sets S and relations $r \subseteq S \times S$ it holds:

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

She tried many sets and relations and did not find any counterexample. Is her conjecture true?
If so, prove it, otherwise provide a counterexample for which S is smallest.

Forward VCG

Some notation

If P is a formula on state and c a command, let $sp_F(P, c)$ be the formula version of the strongest postcondition operator. $sp_F(P, c)$ is therefore the formula Q that describes the set of states that can result from executing c in a state satisfying P.
Thus, we have that

$$sp_F(P, c) = Q$$

implies

$$sp((\{\bar{x}|P\}, \rho(c)) = \{\bar{x}|Q\}$$

We will denote the set of states satisfying a predicate by underscore s, i.e. for a predicate P, let P_s be the set of states that satisfies it:

$$P_s = \{\bar{x}|P\}$$

Forward VCG: Using Strongest Postcondition

We can use the sp_F operator to compute verification conditions: for a triple $\{P\}c\{Q\}$ we can generate the verification condition $sp_F(P, c) \rightarrow Q$.

Assume Statement

Define:

$$sp_F(P, \text{assume(F)}) = P \wedge F$$

Then

$$\begin{aligned}
& sp(P_s, \rho(assume(F))) \\
& = sp(P_s, \Delta_{F_s}) \\
& = \{\bar{x}' \mid \exists \bar{x} \in P_s.\ ((\bar{x}, \bar{x}') \in \Delta_{F_s})\} \\
& = \{\bar{x}' \mid \exists \bar{x} \in P_s.\ (\bar{x} = \bar{x}' \wedge \bar{x} \in F_s)\} \\
& = \{\bar{x}' \mid \bar{x}' \in P_s,\ \bar{x}' \in F_s\} \\
& = P_s \cap F_s.
\end{aligned}$$

Rules for Computing Strongest Postcondition

Havoc Statement

Define:

$$sp_F(P, \text{havoc}(x)) = \exists x_0 . P[x := x_0]$$

Exercise:

Precondition: $\{x \geq 2 \land y \leq 5 \land x \leq y\}$.

Code: havoc(x)

Rules for Computing Strongest Postcondition

Havoc Statement

Define:

$$sp_F(P, \text{havoc}(x)) = \exists x_0 . P[x := x_0]$$

Exercise:

Precondition: $\{x \geq 2 \wedge y \leq 5 \wedge x \leq y\}$.

Code: havoc(x)

$$\exists x_0 .\ x_0 \geq 2 \wedge y \leq 5 \wedge x_0 \leq y$$

i.e.

$$\exists x_0 .\ 2 \leq x_0 \leq y \wedge y \leq 5$$

i.e.

$$2 \leq y \wedge y \leq 5$$

Note: If we simply removed conjuncts containing x, we would get just $y \leq 5$.

Rules for Computing Strongest Postcondition

Assignment Statement
Define:

$$sp_F(P, x = e) = \exists x_0.(P[x := x_0] \wedge x = e[x := x_0])$$

Indeed:

$$\begin{aligned}
& sp(P_s, \rho(x = e)) \\
& = \{\bar{x}' \mid \exists \bar{x}.\ (\bar{x} \in P_s \wedge (\bar{x}, \bar{x}') \in \rho(x = e))\} \\
& = \{\bar{x}' \mid \exists \bar{x}.\ (\bar{x} \in P_s \wedge \bar{x}' = \bar{x}[x := e(\bar{x})])\}
\end{aligned}$$

Exercise

Precondition: $\{x \geq 5 \wedge y \geq 3\}$.
Code: x = x + y + 10

$$sp(x \geq 5 \wedge y \geq 3, x = x + y + 10) =$$

Exercise

Precondition: $\{x \geq 5 \wedge y \geq 3\}$.
Code: x = x + y + 10

$$sp(x \geq 5 \wedge y \geq 3, x = x + y + 10) =$$

$$\exists x_0.\ x_0 \geq 5 \wedge y \geq 3 \ \wedge \ x = x_0 + y + 10$$

$$\leftrightarrow \ y \geq 3 \wedge x \geq y + 15$$

Rules for Computing Strongest Postcondition

Sequential Composition
For relations we proved

$$sp(P_s, r_1 \circ r_2) = sp(sp(P_s, r_1), r_2)$$

Therefore, define

$$sp_F(P, c_1; c_2) = sp_F(sp_F(P, c_1), c_2)$$

Nondeterministic Choice (Branches)
We had $sp(P_s, r_1 \cup r_2) = sp(P_s, r_1) \cup sp(P_s, r_2)$. Therefore define:

$$sp_F(P, c_1[]c_2) = sp_F(P, c_1) \vee sp_F(P, c_2)$$

Correctness

Show by induction on c_1 that for all P:

$$sp(P_s, \rho(c_1)) = \{\bar{x}' \mid \; sp_F(P, c_1)\}$$

Size of Generated Formulas

The size of the formula can be exponential because each time we have a nondeterministic choice, we double formula size:

$$
\begin{aligned}
& sp_F(P, (c_1[]c_2); (c_3[]c_4)) = \\
& sp_F(sp_F(P, c_1[]c_2), c_3[]c_4) = \\
& sp_F(sp_F(P, c_1) \vee sp_F(P, c_2), c_3[]c_4) = \\
& sp_F(sp_F(P, c_1) \vee sp_F(P, c_2), c_3) \vee sp_F(sp_F(P, c_1) \vee sp_F(P, c_2), c_4)
\end{aligned}
$$

Reducing sp to Relation Composition

The following identity holds for relations:

$$sp(P_s, r) = ran(\Delta_P \circ r)$$

Based on this, we can compute $sp(P_s, \rho(c_1))$ in two steps:

- compute formula $F(assume(P); c_1)$
- existentially quantify over initial (non-primed) variables

Indeed, if F_1 is a formula denoting relation r_1, that is,

$$r_1 = \{(\vec{x}, \vec{x}\,'). \ F_1(\vec{x}, \vec{x}\,')\}$$

then $\exists \vec{x}. F_1(\vec{x}, \vec{x}\,')$ is formula denoting the range of r_1:

$$ran(r_1) = \{\vec{x}\,'. \ \exists \vec{x}. F_1(\vec{x}, \vec{x}\,')\}$$

Moreover, the resulting approach does not have exponentially large formulas.

More on Weakest Preconditions

Exercise: Prove wp Distributivity

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

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

Rules for WP

Rules for Computing Weakest Preconditions

We derive the rules below from the definition of weakest precondition on sets and relations

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

Assume Statement

Suppose we have one variable x, and identify the state with that variable. Note that $\rho(assume(F)) = \Delta_{F_s}$. By definition

$$\begin{aligned} wp(\Delta_{F_s}, Q_s) &= \{x \mid \forall x'.(x, x') \in \Delta_{F_s} \rightarrow x' \in Q_s\} \\ &= \{x \mid \forall x'.(x \in F_s \wedge x = x') \rightarrow x' \in Q_s\} \\ &= \{x \mid x \in F_s \rightarrow x \in Q_s\} = \{x \mid F \rightarrow Q\} \end{aligned}$$

Changing from sets to formulas, we obtain the rule for *wp* on formulas:

$$wp_F(\text{assume(F)}, Q) = (F \rightarrow Q)$$

Rules for Computing Weakest Preconditions

Assignment Statement

Consider the case of two variables. Recall that the relation associated with the assignment $x = e$ is

$$x' = e \wedge y' = y$$

Then we have, for formula Q containing x and y:

$$\begin{aligned} wp(\rho(x = e), \{(x, y) \mid Q\}) &= \{(x, y) \mid \forall x'.\forall y'.\ x' = e \wedge y' = y \rightarrow \\ &\qquad Q[x := x', y := y']\} \\ &= \{(x, y) \mid Q[x := e]\} \end{aligned}$$

From here we obtain a justification to define:

$$wp_F(x = e, Q) = Q[x := e]$$

Rules for Computing Weakest Preconditions

Havoc Statement

$$wp_F(\text{havoc(x)}, Q) = \forall x.Q$$

Sequential Composition

$$wp(r_1 \circ r_2, Q_s) = wp(r_1, wp(r_2, Q_s))$$

Same for formulas:

$$wp_F(c_1 \; ; \; c_2, Q) = wp_F(c_1, wp_F(c_2, Q))$$

Nondeterministic Choice (Branches)

In terms of sets and relations

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

In terms of formulas

$$wp_F(c_1 [] c_2, Q) = wp_F(c_1, Q) \land wp_F(c_2, Q)$$

Summary of Weakest Precondition Rules

c	$wp(c, Q)$
$x = e$	$Q[x := e]$
$havoc(x)$	$\forall x. Q$
$assume(F)$	$F \rightarrow Q$
$c_1 \,[]\, c_2$	$wp(c_1, Q) \wedge wp(c_2, Q)$
$c_1; c_2$	$wp(c_1, wp(c_2, Q))$

Size of Generated Verification Conditions

Because of the rule

$$wp_F(c_1[]c_2, Q) = wp_F(c_1, Q) \wedge wp_F(c_2, Q)$$

which duplicates Q, the size can be exponential.

$wp_F((c_1 \,[]\, c_2); (c_3 \,[]\, c_4), Q) =$

Avoiding Exponential Blowup

Propose an algorithm that, given an arbitrary program c and a formula Q, computes in polynomial time formula equivalent to $wp_F(c, Q)$

Syntactic Rules for Hoare Logic

Summary of Proof Rules

We next present (one possible) summary of proof rules for Hoare logic.

Weakening and Strengthening
Strengthening precondition:

$$\frac{\models P_1 \rightarrow P_2 \quad \{P_2\}c\{Q\}}{\{P_1\}c\{Q\}}$$

Weakening postcondition:

$$\frac{\{P\}c\{Q_1\} \quad \models Q_1 \rightarrow Q_2}{\{P\}c\{Q_2\}}$$

Loop Free Blocks

We can directly use the rules we derived for basic loop-free code. Either through weakest preconditions or strongest postconditions.

$$\{wp(c, Q)\}c\{Q\}$$

or,

$$\{P\}c\{sp(P, c)\}$$

For example, we have:

$$\{Q[x := e]\}\ (x = e)\ \{Q\}$$

$$\{\forall x.Q\}\ havoc(x)\ \{Q\}$$

$$\{(F \to Q)\}\ assume(F)\ \{Q\}$$

$$\{P\}\ assume(F)\ \{P \land F\}$$

Rules continued

Loops

$$\frac{\{I\}c\{I\}}{\{I\}\ while(*)c\{I\}}$$

Sequential Composition

$$\frac{\{P\}\,c_1\{Q\} \quad \{Q\}\,c_2\,\{R\}}{\{P\}\ c_1;c_2\ \{R\}}$$

Non-Deterministic Choice

$$\frac{\{P\}c_1\{Q\} \quad \{P\}c_2\{Q\}}{\{P\}c_1[]c_2\{Q\}}$$

While Loops

Knowing that the while loop: **while** (F) c;

is equivalent to: **while**($*$){**assume**(F); c} ;
assume($\neg$ F);

Question: What is the rule for while loops?
Hint

$$\frac{(\models P \rightarrow ?);\ \{?\}c\{?\};\ (\models ? \rightarrow Q)}{\{P\}\ while\{I\}(F)(c)\ \{Q\}}$$

While Loops

Knowing that the while loop: **while** (F) c;

is equivalent to: **while**($*$){**assume**(F); c} ;
assume($\neg$ F);

Question: What is the rule for while loops?
Hint

$$\frac{(\models P \rightarrow ?);\ \{?\}c\{?\};\ (\models ? \rightarrow Q)}{\{P\}\ while\{I\}(F)(c)\ \{Q\}}$$

It follows that the rule for while loops is:

$$\frac{(\models P \rightarrow I);\ \{I \wedge F\}c\{I\};\ (\models (I \wedge \neg F) \rightarrow Q))}{\{P\}\ while\{I\}(F)(c)\ \{Q\}}$$

Applying Proof Rules given Invariants

Let us treat $\{P\}$ as a new kind of statement, written

assert(P)

For the moment the purpose of assert is just to indicate preconditions and postconditions. When we write

```
assert(P)
c1;
assert(Q)
c2;
assert(R)
```

we expect that these Hoare triples hold:

$$\{P\}c1\{Q\}$$
$$\{Q\}c2\{R\}$$

Sufficiently annotated program

Consider the control-flow graph of a program with statements assert, assume, x=e and with graph edges expressing "[]" and ";". We will say that the program c is *sufficiently annotated* iff

- the first statement is assert(Pre)
- the last statement is assert(Post)
- every cycle in the control-flow graph contains at least one assert

Assertion path

An assertion path is a path in its control-flow graph that starts and ends with *assert*. Given the assertion path

assert(P)
c1
...
cK
assert(Q)

we omit any assert statements in the middle, obtaining from c1,...,cK statements d1,...,dL. We call

$$\{P\}d1 \ ; \ \ldots \ ; \ dL\{Q\}$$

the Hoare triple of the assertion path.

Proving Hoare triple for entire program

A *basic path* is an assertion path that contains no *assert* commands other than those at the beginning and end. Each sufficiently annotated program has finitely many basic paths.

Theorem: If the Hoare triple for each basic path is valid, then the Hoare triple $\{Pre\}c\{Post\}$ is valid.
Proof: If each basic path is valid, then each path is valid, by induction and Hoare logic rule for sequential composition. Each program is union of (potentially infinitely many) paths, so the property holds for the entire program. (Another explanation: consider any given execution and corresponding path in the control-flow graph. By induction on the length of the path we prove that all assert statements hold, up to the last one.)

Verification recipe

The verification condition of a basic path is the formula whose validity expresses the validity of the Hoare triple for this path.
Simple verification conditions for a sufficiently annotated program is the set of verification conditions for each each basic path of the program.
One approach to verification condition generation is therefore:

- start with sufficiently annotated program
- generate simple verification conditions
- prove each of the simple verification conditions

In a program of size n, what is the bound on the number of basic paths?

Verification recipe

The verification condition of a basic path is the formula whose validity expresses the validity of the Hoare triple for this path.
Simple verification conditions for a sufficiently annotated program is the set of verification conditions for each each basic path of the program.
One approach to verification condition generation is therefore:

- start with sufficiently annotated program
- generate simple verification conditions
- prove each of the simple verification conditions

In a program of size n, what is the bound on the number of basic paths?
It can be $2^{O(n)}$.

Handling the path explosion

In a program of size n, the number of basic paths can be $2^{O(n)}$.
Remedies:

- require more annotations (e.g. at each merge point)
- extreme case: assertion on each CFG vertex - this gives classical Hoare logic proof
- merge subgraphs without annotations: perform sequential composition and disjunction of formulas on edges
- generate correctness formulas for multiple paths in an acyclic subgraph at once, using propositional variables to encode the existence of paths

Exercise

Give a complete Hoare logic proof for the following program:

```
{n >= 0 && d > 0}
    q = 0
    r = n
    while ( r >= d ) {
        q = q + 1
        r = r - d
    }
{n == q * d + r && 0 <= r < d}
```

The proof should be step-by-step as in the example proof in the lecture on Hoare Logic. To prove each step you can use the syntactic rules for Hoare Logic.

Exercise

```
// {n >= 0 && d > 0}
q = 0
// {n >= 0 && d > 0 && q == 0}
r = n
// {n >= 0 && d > 0 && q == 0 && r == n}
while // {d > 0 && n == q * d + r && 0 <= r}
  (r >= d) {
// {d > 0 && n == q * d + r && d <= r}
   q = q + 1
// {d > 0 && n == (q-1) * d + r && d <= r}
   r = r - d
// {d > 0 && n == (q-1) * d + r + d && 0 <= r}
// {d > 0 && n == q * d + r && 0 <= r}
}
// {d > 0 && n == q * d + r && 0 <= r && r < d}
// {n == q * d + r && 0 <= r < d}
```

What can be omitted to still have sufficiently annotated program?