Lecturecise 6
More on Postconditions and Preconditions.
Loops and Recursion

Viktor Kuncak

Review of Key Definitions

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:

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

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

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

Pointwise sp

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

Hoare Logic for Loop-free Code

$P \subseteq wp(\bigcup_{i \in J} r_i, Q)$

$sp(P, \bigcup_{i \in J} r_i) \subseteq Q$

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

Expanding Paths

The condition

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

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 $Q \subseteq S$.

$\subseteq \rightarrow$

(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:

$\Delta_S = \{(x,x) \mid x \in S\} \subseteq S \times S$

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

$dom(r) = \{x \mid \exists y.\ (x,y) \in r\}$

She tried many sets and relations and did not find any counterexample. Is her conjecture true?

If so, prove it, otherwise provide a counterexample for which S is smallest.

Formulas for Strongest Postconditions

Forward Verification Condition Generation

Computing Formulas for Strongest Postcondition

Let $\bar{x}, \bar{x}'$ range over the sets of states S
We gave definition of strongest postcondition (sp) on sets and relations $P_1 \subseteq S$ and $r \subseteq S \times S$:

$$sp(P_1, r) = \{\bar{x}' \mid \exists \bar{x}.\ \bar{x} \in P_1 \land (\bar{x}, \bar{x}') \in r\}$$

We now consider how to compute with *representations* of those sets and relations in terms of formulas. Let

- $P_1 = \{\bar{x} \mid P\}$ for some formula P with $FV(P)$ among $\bar{x}$
- $r = \rho(c) = \{(\bar{x}, \bar{x}') \mid F\}$ for some formula F with $FV(F)$ among $\bar{x}$, $\bar{x}'$

We can then conclude $sp(P_1, r) = \{\bar{x}' \mid \exists \bar{x}.\ P \land F\}$
Denote a formula equivalent to $(\exists \bar{x}.\ P \land F)[\bar{x}' := \bar{x}]$ by $\mathbf{sp_F(P, c)}$

- we renamed variables so that the result is in terms of $\bar{x}$, not $\bar{x}'$
- multiple syntactic choices for $sp(P_1, r)$; all logically equivalent

Strongest Postcondition Formula

If P is a formula on state and c a command, we define $sp_F(P, c)$ as 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

Remember: $\{P_s\}\ \rho(c)\ \{Q_s\}$ is equivalent to

$$sp(P_s, \rho(c)) \subseteq Q_s$$

A syntactic form of Hoare triple is $\{P\}c\{Q\}$

That syntactic form is therefore equivalent to proving

$$\forall \bar{x}.\ (sp_F(P, c) \rightarrow Q)$$

We can use the sp_F operator to compute verification conditions

Forward VCG: Using Strongest Postcondition

Remember: $\{P_s\}\ \rho(c)\ \{Q_s\}$ is equivalent to

$$sp(P_s, \rho(c)) \subseteq Q_s$$

A syntactic form of Hoare triple is $\{P\}c\{Q\}$

That syntactic form is therefore equivalent to proving

$$\forall \bar{x}.\ (sp_F(P, c) \rightarrow Q)$$

We can use the sp_F operator to compute verification conditions

We give rules to compute $sp_F(P, c)$ for our commands such that

$$(sp_F(P, c) = Q) \ \text{ implies } \ (sp(P_s, \rho(c)) = Q_s)$$

Finding Formula for sp_F

Given the goal of the formula

$$(sp_F(P, c) = Q) \text{ implies } (sp(P_s, \rho(c)) = Q_s)$$

All Q with $FV(Q) \subseteq \bar{x}$ satisfying $sp(P_s, \rho(c)) = Q_s$ are equivalent to formula

$$(\exists \bar{x}.\ P \wedge F)[\bar{x}' := \bar{x}] \qquad (*)$$

where $\rho(c) = \{(\bar{x}, \bar{x}') \mid F\}$

- we are looking for some syntactic simplification of $(*)$

Assume Statement

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

$E_s = \{ \bar{x} \mid E \}$

Consider

- a precondition P, with $FV(P)$ among $\bar{x}$ and
- a property E, also with $FV(E)$ among $\bar{x}$

Note that $\rho(assume(E)) = \Delta_{E_s}$. Therefore

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

So, we define:

$$sp_F(P, \mathrm{assume(E)}) = P \wedge E$$

Strongest Postcondition of Havoc

Formula for havoc. Let $\bar{x} = x_1, \ldots, x_i, \ldots, x_n$

$$R(havoc(x_i)) = \bigwedge_{v \neq x_i} v = v' \qquad = F$$

General formula for postcondition is:

$$(\exists \bar{x}.\ P \wedge F)[\bar{x}' := \bar{x}] \tag{$*$}$$

It becomes here

$$(\exists \bar{x}.\ P \wedge \bigwedge_{j \neq i} x_j = x_j')[\bar{x}' := \bar{x}]$$

Equalities over all variables except x_i are eliminated, so we obtain

$$(\exists x_i . P)[\bar{x}' := \bar{x}]$$

No primed variables left, renaming does nothing. Result: $\mathbf{(\exists x_i . P)}$.

Strongest Postcondition of Havoc

To avoid many nested quantifiers and name clashes, we rename first:

$$sp_F(P, \text{havoc}(x)) = \exists x_0 . P[x := x_0] \;\; \text{which is same as } \exists x . P$$

Strongest Postcondition of Havoc

To avoid many nested quantifiers and name clashes, we rename first:

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

Exercise:

Precondition: $\{x \geq 2 \land y \leq 5 \land x \leq y\}$.
Code: havoc(x)

Strongest Postcondition of Havoc

To avoid many nested quantifiers and name clashes, we rename first:

$$sp_F(P, \text{havoc}(x)) = \exists x_0. P[x := x_0] \;\text{ which is same as } \exists x. P$$

Exercise:
Precondition: $\{x \geq 2 \land y \leq 5 \land x \leq y\}$.
Code: havoc(x)

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

i.e.

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

i.e.

$$2 \leq y \land y \leq 5$$

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:

$$sp_F(P, c_1 \,\square\, c_2) = sp_F(P, c_1) \vee sp_F(P, c_2)$$

Correctness

We can show by easy induction on c_1 that for all P:

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

Size of Generated Formulas

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

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

Another Useful Characterization of sp

$$= \{ z \mid \exists x.\ \ x \in A \wedge (x, z) \in r \}$$

For any relation $\sigma \subseteq S \times S$ we define its range by

$$ran(\sigma) = \{\underline{s' \mid} \exists s \in S.(s, s') \in \sigma\}$$

Lemma: suppose that

- $A \subseteq S$ and $r \subseteq S \times S$
- $\Delta = \{(s, s) \mid s \in S\}$

Then

$$sp(A, r) = ran(\Delta_A \circ r)$$

$$ran(\Delta_A \circ r) = ran(\{(x,z) \mid \exists y.\ \ \underline{(x,y) \in \Delta_A} \wedge (y,z) \in r\})$$

$$= ran(\{(x,z) \mid \exists y.\ \underline{x = y} \wedge x \in A \wedge (y,z) \in r\}$$

$$= ran(\{(x,z) \mid x \in A \wedge (x,z) \in r\})$$

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 $R(assume(P); c_1)$
- 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:

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

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

Computing Weakest Precondition Formulas

Rules for Computing Weakest Preconditions

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

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

Let now $r = \rho(c) = \{(\bar{x}, \bar{x}') \mid F\}$ and $Q_s = \{\bar{x} \mid Q\}$. Then

$$wp(r, Q_s) = \{\bar{x} \mid \forall \bar{x}'.(F \rightarrow Q[\bar{x} := \bar{x}'])\}$$

Thus, we will be defining wp_F as equivalent to

$$\forall \bar{x}'.\ (F \wedge Q[\bar{x} := \bar{x}'])$$

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:

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

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

$$wp_F(c_1 \square c_2, Q) = wp_F(c_1, Q) \wedge wp_F(c_2, Q)$$

Summary of Weakest Precondition Rules

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

Size of Generated Verification Conditions

Because of the rule

$$wp_F(c_1 \square c_2, Q) = wp_F(c_1, Q) \wedge wp_F(c_2, Q)$$

which duplicates Q, the size can be exponential.

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

Avoiding Exponential Blowup

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

Loops

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x - y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?
Let k be the number of times loop executes.

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?

Let k be the number of times loop executes.

- $k = 0$:

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?

Let k be the number of times loop executes.

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

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?

Let k be the number of times loop executes.

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

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?
Let k be the number of times loop executes.

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

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?
Let k be the number of times loop executes.

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

Loops: Example

Consider the set of variables $V = \{x, y\}$ and this program L:

```
while (x > 0) {
  x = x − y
}
```

When the loop terminates, what is the (strongest) relation $\rho(L)$ between state (x, y) before loop started executing and the final state (x', y')?
Let k be the number of times loop executes.

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

Solution:

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

Heuristically Eliminating a Quantifier from non-PA formula

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

This implies $y > 0$.

Heuristically Eliminating a Quantifier from non-PA formula

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

This implies $y > 0$.

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

Heuristically Eliminating a Quantifier from non-PA formula

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

This implies $y > 0$.

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

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

Heuristically Eliminating a Quantifier from non-PA formula

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

This implies $y > 0$.

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

$$\exists k.\, y > 0 \wedge k > 0 \wedge x > 0 \wedge y | (x - x') \wedge k = (x - x')/y \wedge x' \leq 0 \wedge y' = y$$

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

Heuristically Eliminating a Quantifier from non-PA formula

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

This implies $y > 0$.

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

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

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

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

Integer Programs with Loops

Even if loop body is in Presburger arithmetic, the semantics of a loop need not be.

Integer programs with loops are Turing complete and can compute all computable functions.

Even if we cannot find Presburger arithmetic formula, we may be able to find

- a formula in a richer logic
- a property of the meaning of the loop
  (e.g. formula for the superset)

To help with these tasks, we give mathematical semantics of loops
Useful concept for this is transitive closure: $r^* = \bigcup_{n \geq 0} r^n$
(We may or may not have a general formula for r^n or r^*)

Towards meaning of loops: unfolding

Loops can describe an infinite number of basic paths
(for a larger input, program takes a longer path)
Consider loop

$$L \equiv \textbf{while}(F)c$$

We would like to have

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

For $r_L = \rho(L)$, $r_c = \rho(c)$, $\Delta_f = \Delta_{S(F)}$, $\Delta_{nf} = \Delta_{S(\neg F)}$ we have

$$\begin{aligned} r_L &= (\Delta_f \circ r_c \circ r_L) \cup \Delta_{nf} \\ &= (\Delta_f \circ r_c \circ ((\Delta_f \circ r_c \circ r_L) \cup \Delta_{nf})) \cup \Delta_{nf} \\ &= \quad \Delta_{nf} \cup \\ &\quad\;\; (\Delta_f \circ r_c) \circ \Delta_{nf} \cup \\ &\quad\;\; (\Delta_f \circ r_c)^2 \circ r_L \end{aligned}$$

Unfolding Loops

$$\begin{aligned} r_L \quad = \quad & \Delta_{nf} \cup \\ & (\Delta_f \circ r_c) \circ \Delta_{nf} \cup \\ & (\Delta_f \circ r_c)^2 \circ \Delta_{nf} \cup \\ & (\Delta_f \circ r_c)^3 \circ r_L \end{aligned}$$

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

$$(\Delta_f \circ r_c)^n \circ \Delta_{nf} \subseteq r_L$$

So, $(\Delta_f \circ r_c)^* \circ \Delta_{nf} \subseteq r_L$.
We define r_L to be:

$$r_L = (\Delta_f \circ r_c)^* \circ \Delta_{nf}$$

THEREFORE:

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

Using Loop Semantics in Example

ρ of L:

```
while (x > 0) {
  x = x − y
}
```

is:

Using Loop Semantics in Example

ρ of L:

```
while (x > 0) {
  x = x − y
}
```

is:

$$(\Delta_{S(x>0)} \circ \rho(x = x - y))^{*} \circ \Delta_{S(\neg(x>0))}$$

Compute each relation:

$$\begin{aligned}
\Delta_{S(x>0)} &= \{((x, y), (x, y)) \mid x > 0\} \\
\Delta_{S(\neg(x>0))} &= \{((x, y), (x, y)) \mid x \leq 0\} \\
\rho(x = x - y) &= \{((x, y), (x - y, y)) \mid x, y \in \mathbb{Z}\} \\
\Delta_{S(x>0)} \circ \rho(x = x - y) &= \\
(\Delta_{S(x>0)} \circ \rho(x = x - y))^{k} &= \\
(\Delta_{S(x>0)} \circ \rho(x = x - y))^{*} &= \\
\rho(L) &=
\end{aligned}$$

Semantics of a Program with Loop

Compute and simplify relation for this program:

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

$$
\begin{array}{l}
\rho(x = 0) \circ \\
(\Delta_{S(y>0)} \circ \rho(x = x + y; y = y - 1))^* \circ \\
\Delta_{S(y \leq 0)}
\end{array}
$$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics.

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

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

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

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

Recursion

Example of Recursion

For simplicity assume no parameters
(we can simulate them using global variables)

```
def f =
 if (x > 0) {
   if (x % 2 == 0) {
     x = x / 2;
     f;
     y = y * 2
   } else {
     x = x - 1;
     y = y + x;
     f
   }
 }
```

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

Assume recursive function call denotes some relation r_f

Need to find relation r_f such that $r_f = E(r_f)$

Simpler Example of Recursion

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

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

Simpler Example of Recursion

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

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

What is $E(\emptyset)$?

Simpler Example of Recursion

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

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

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
  }
```

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

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

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?

Sequence of Bounded Recursions

Consider the sequence of relations $r_0 = \emptyset$, $r_k = E^k(\emptyset)$.
What is the relationship between r_k and r_{k+1}?
Define

$$s = \bigcup_{k \geq 0} r_k$$

Then

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

If $E(s) = s$ we say s is a **fixed point (fixpoint)** of function E

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

Exercise with Fixpoints of Real Functions

1. Find all fixpoints of function $f : \mathbb{R} \to \mathbb{R}$ defined as

$$f(x) = x^2 - x - 3$$

2. Compute the fixpoint that is smaller than all other fixpoints

Union of Finite Unfoldings is Least Fixpoint

C - a collection (set) of sets (e.g. sets of pairs, i.e. relations)
$E : C \to C$ such that for $r_0 \subseteq r_1 \subseteq r_2 \ldots$
we have

$$E(\bigcup_i r_i) = \bigcup_i E(r_i)$$

Then $s = \bigcup_i E^i(\emptyset)$ is such that

1. $E(s) = s$ (we have shown this)
2. if r is such that $E(r) \subseteq r$ (special case: if $E(r) = r$), then $s \subseteq r$

Prove this theorem.

Least Fixpoint

$$s = \bigcup_i E^i(\emptyset)$$

Suppose $E(r) \subseteq r$.
Showing $s \subseteq r$

$$\bigcup_i E^i(\emptyset) \subseteq r$$

Consequence of s being smallest

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

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

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

Consequence of s being smallest

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

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

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

- $E(r) \subseteq r$
- then because procedure meaning s is least, $s \subseteq r$

Proving that recursive function meets specification

Prove that if s is the relation denoting the recursive function below, then

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

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

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

Multiple Procedures

Two mutually recursive procedures $r_1 = E_1(r_1), \quad r_2 = E_2(r_2)$

Extend the approach to work on pairs of relations:

$$(r_1, r_2) = (E_1(r_1), E_2(r_2))$$

Define $\bar{E}(r_1, r_2) = (E_1(r_1), E_2(r_2))$, let $\bar{r} = (r_1, r_2)$

$$\bar{E}(\bar{r}) \sqsubseteq \bar{r}$$

where $(r_1, r_2) \sqsubseteq (r_1', r_2')$ iff $r_1 \subseteq r_1'$ and $r_2 \subseteq r_2'$
Even though pairs of relations are not sets, we can analogously define set-like operations on them. Most theorems still hold.

Generalizing: the entire theory works when we have certain ordering relation

Lattices as a generalization of families of sets