Loop Semantics and its Approximation

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 (smallest) 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 (smallest) 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 (smallest) 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 (smallest) 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 (smallest) 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 (smallest) 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 (smallest) 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.

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

Loops: Example

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

while $(\mathrm{x} > 0)$ {
  $\mathrm{x} = \mathrm{x} - \mathrm{y}$
}

When the loop terminates, what is the (smallest) 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 formula

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

Heuristically Eliminating a Quantifier from formula

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

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

Heuristically Eliminating a Quantifier from formula

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

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

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

Heuristically Eliminating a Quantifier from formula

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

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

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

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

Apply one-point rule to eliminate k

Heuristically Eliminating a Quantifier from formula

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

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

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

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

Apply one-point rule to eliminate k

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

which is also equivalent to simply

Heuristically Eliminating a Quantifier from formula

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

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

Note that $x - x' > 0$ and $k > 0$ so from $ky = x - x'$ we get $y > 0$.

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

Apply one-point rule to eliminate k

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

which is also equivalent to simply

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

Formula for Loop

Meaning of

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

is given by formula

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

Formula for Loop

Meaning of

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

is given by formula

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

What happens if initially $x > 0 \wedge y \leq 0$?

- in the formula
- in the program

Integer Programs with Loops

Integer programs with loops are Turing complete and can compute all computable functions (we can use large integers as Turing machine tape).

Even if we cannot find a closed-form integer 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^*)

Some facts about relations

Let $r \subseteq S \times S$ and $\Delta = \{(x,x) \mid x \in S\}$. Then

$$\Delta \circ r = r = r \circ \Delta$$

We say that r is **reflexive** iff $\forall x \in S.(x,x) \in r$.

- equivalently, reflexivity means $\Delta \subseteq r$

Relation r is **transitive** iff

$$\forall x, y, z.\ ((x,y) \in r \wedge (y,z) \in r \rightarrow (x,z) \in r)$$

which is the same as saying $r \circ r \subseteq r$

Transitive Closure of a Relation

$r \subseteq S \times S$. Define $r^0 = \Delta$ and $r^{n+1} = r \circ r^n$. Then $(x_0, x_n) \in r^n$ iff $\exists x_1, \ldots, x_{n-1}$ such that $(x_i, x_{i+1}) \in r$ for $0 \leq i \leq n-1$.
Define reflexive transitive closure of r by

$$r^* = \bigcup_{n \geq 0} r^n$$

Properties that follow from the definition:

- $(x_0, x_n) \in r^*$ iff there exists $n \geq 0$ and $\exists x_1, \ldots, x_{n-1}$ such that $(x_i, x_{i+1}) \in r$ for $0 \leq i \leq n-1$ (a path in the graph r)
- r^* is a reflexive and transitive relation
- If s is a reflexive transitive relation and $r \subseteq s$, then $r^* \subseteq s$
  - r^* is the smallest reflexive transitive relation containing r
- $(r^{-1})^* = (r^*)^{-1}$
- $r_1 \subseteq r_2$ implies $r_1^* \subseteq r_2^*$
- $r^* = \Delta \cup (r \circ r^*)$ and, likewise, $r^* = \Delta \cup (r^* \circ r)$

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 $r_L = \rho(L)$, $r_c = \rho(c)$, $\Delta_1 = \Delta_{\tilde{F}}$, $\Delta_2 = \Delta_{\tilde{\neg F}}$ we have

$$\begin{aligned} r_L &= (\Delta_1 \circ r_c \circ r_L) \cup \Delta_2 \\ &= (\Delta_1 \circ r_c \circ ((\Delta_1 \circ r_c \circ r_L) \cup \Delta_2)) \cup \Delta_2 \\ &= \Delta_2 \cup \\ &\quad (\Delta_1 \circ r_c) \circ \Delta_2 \cup \\ &\quad (\Delta_1 \circ r_c)^2 \circ r_L \end{aligned}$$

Unfolding Loops

$$\begin{aligned} r_L \;=\; & \Delta_2 \cup \\ & (\Delta_1 \circ r_c) \circ \Delta_2 \cup \\ & (\Delta_1 \circ r_c)^2 \circ \Delta_2 \cup \\ & (\Delta_1 \circ r_c)^3 \circ r_L \end{aligned}$$

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

$$(\Delta_1 \circ r_c)^n \circ \Delta_2 \subseteq r_L$$

So, $\bigcup_{n\geq 0}\big((\Delta_1 \circ r_c)^n \circ \Delta_2\big) \subseteq r_L$, that is

$$\Big(\bigcup_{n\geq 0}(\Delta_1 \circ r_c)^n\Big) \circ \Delta_2 \subseteq r_L$$

We do not wish to have unnecessary elements in relation, so we try

$$r_L = (\Delta_1 \circ r_c)^* \circ \Delta_2$$

and this does satisfy $r_L = (\Delta_1 \circ r_c \circ r_L) \cup \Delta_2$, so we define

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

Why loop semantics satisfies the condition

We defined

$$r_L = (\Delta_1 \circ r_c)^* \circ \Delta_2$$

Show that $(\Delta_1 \circ r_c \circ r_L) \cup \Delta_2$ equals r_L, as we expect from recursive definition of a while loop.

Why loop semantics satisfies the condition

We defined

$$r_L = (\Delta_1 \circ r_c)^* \circ \Delta_2$$

Show that $(\Delta_1 \circ r_c \circ r_L) \cup \Delta_2$ equals r_L, as we expect from recursive definition of a while loop.

Using property $r^* = \Delta \cup r \circ r^*$ we have

$$\begin{aligned} r_L &= (\Delta_1 \circ r_c)^* \circ \Delta_2 \\ &= [\Delta \cup \Delta_1 \circ r_c \circ (\Delta_1 \circ r_c)^*] \circ \Delta_2 \\ &= \Delta_2 \cup [\Delta_1 \circ r_c \circ (\Delta_1 \circ r_c)^* \circ \Delta_2] \\ &= \Delta_2 \cup \Delta_1 \circ r_c \circ r_L \end{aligned}$$

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:

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

Compute each relation:

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

Semantics of a Program with a Loop

Compute and simplify relation for this program:

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

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

$R(x=0)$	$x'=0\wedge y'=y$
$R([y>0])$	$y'>0\wedge x'=x\wedge y'=y$
$R([y\leq 0])$	$y'\leq 0\wedge x'=x\wedge y'=y$
$R(\ [y>0];\ x=x+y;\ y=y-1)$	$y>0\wedge x'=x+y\wedge y'=y-1$
$R((\ [y>0];\ x=x+y;\ y=y-1)^k), k>0$	$y-(k-1)>0\wedge$ $x'=x+(y+(y-1)+\cdots+y-(k-1))\wedge y'=y-k$ *i.e.* $y\geq k\wedge x'=x+k(y+y-(k-1))/2\wedge y'=y-k$
$R((\ [y>0];\ x=x+y;\ y=y-1)^*)$	$(x'=x\wedge y'=y)\ \vee\ \exists k>0.\ y\geq k\wedge x'=x+k(2y-k+1))/2\wedge y'=y-k$ *i.e.*, eliminating $k=y-y'$, $(x'=x\wedge y'=y)\vee(y-y'>0\wedge y'\geq 0\wedge x'=x+(y-y')(y+y'+1)/2)$
$R(\text{program})$	$(x'=0\wedge y'=y\wedge y'\leq 0)\vee(y>0\wedge y'=0\wedge x'=y(y+1)/2)$

Remarks on Previous Solution

Intermediate components can be more complex than final result

- they must account for all possible initial states, even those never reached in actual executions

Be careful with handling base case. The following solution:

$$y' = 0 \wedge x' = y(y+1)/2$$

is "almost correct": it incorrectly describes behavior when the initial state has, for example, $y = -2$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in $s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

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

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

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in $s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

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

$\sqsubseteq\sqcap$

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

$\sqsubseteq\cap$

Approximate Semantics of Loops

Instead of computing exact semantics, it can be sufficient to compute approximate semantics. Observation: $r_1 \subseteq r_2 \rightarrow r_1^* \subseteq r_2^*$ (monotonicity still holds).
Suppose we only wish to show that the semantics is included in $s = \{(x, y, x', y') \mid y' \leq y\}$. Note $s \circ s \subseteq s$, $s^* \subseteq s$. Then

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

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

$\sqsubseteq \sqcap$

$\sqsubseteq \cap$

```
x = 0
while (y > 0) {
  val y0 = y
  havoc(x,y); assume(y ≤ y0)
}
```

$s \circ$
$(s \circ s \circ s)^* \circ s$

$\sqsubseteq \cap$

s