Lecture 3: Converting Imperative Programs to Formulas

Viktor Kuncak

Verification-Condition Generation for Imperative Non-Deterministic Programs

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

program:	$x = x + 2; y = x + 10$
relation:	$\{(x, y, z, x', y', z') \mid x' = x + 2 \land y' = x + 12 \land z' = z\}$
formula:	$x' = x + 2 \land y' = x + 12 \land z' = z$

Specification: $z = old(z) \land (old(x) > 0 \to (x > 0 \land y > 0))$
Adhering to specification is relation subset:

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

or validity of the following implication:

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

Imperative Presburger Arithmetic Programs

F - formulas, t - terms - as in functional programs so far
Fixed number of mutable integer variables $V = \{x_1, \ldots, x_n\}$
Imperative statements:

- $\mathbf{x = t}$: change $x \in V$ to have value given by t; leave vars in $V \setminus \{x\}$ unchanged
- $\mathbf{if(F)c_1\ else\ c_2}$: if F holds, execute c_1 else execute c_2
- $\mathbf{c_1; c_2}$: first execute c_1, then execute c_2

Statements for introducing and restricting non-determinism:

- **havoc(x)**: non-deterministically change $x \in V$ to have an arbitrary value; leave vars in $V \setminus \{x\}$ unchanged
- $\mathbf{if}(*)\ \mathbf{c_1\ else\ c_2}$: arbitrarily choose to run c_1 or c_2
- **assume(F)**: block all executions where F does not hold

Given such loop-free program c with conditionals, compute a polynomial-sized formula $R(c)$ of form: $\exists \bar{z}.F(\bar{x}, \bar{z}, \bar{x}')$ describing relation between initial values of variables $x_1, \ldots, x_n$ and final values of variables $x'_1, \ldots, x'_n$

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,

$$if\,(b)\ c_1\ \ else\ \ c_2$$

Formula for if-else

After flattening,

$$if\,(b)\ c_1\ \ else\ \ c_2$$

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

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

Command semicolon

$$c_1 \,;\, c_2$$

Command semicolon

$$c_1 ; c_2$$

Reminder about relation composition and its definition:

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

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:

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

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

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

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

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

Non-deterministic choice

$$if\,(*)\ c_1\ \ else\ \ c_2$$

$R(if\,(*)\ c_1\ \ else\ \ c_2)$:

$$R(c_1) \vee R(c_2)$$

- 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

$$assume(F)$$

$R(assume(F))$:

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

- 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

$$
\begin{array}{l}
{}^{0} \\
(\mathit{if}\ (b)\ x = x + 1\ \mathit{else}\ y = x + 2); \\
{}^{1} \\
x = x + 5; \\
{}^{2} \\
(\mathit{if}\ (*)\ y = y + 1\ \mathit{else}\ x = y) \\
{}^{3}
\end{array}
$$

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')

Imperative Presburger Arithmetic Programs

F - formulas, t - terms - as in functional programs so far
Fixed number of mutable integer variables $V = \{x_1, \ldots, x_n\}$
Imperative statements:

- $\mathbf{x = t}$: change $x \in V$ to have value given by t; leave vars in $V \setminus \{x\}$ unchanged
- $\mathbf{if(F)c_1\ else\ c_2}$: if F holds, execute c_1 else execute c_2
- $\mathbf{c_1; c_2}$: first execute c_1, then execute c_2

Statements for introducing and restricting non-determinism:

- **havoc**$(\mathbf{x})$: non-deterministically change $x \in V$ to have an arbitrary value; leave vars in $V \setminus \{x\}$ unchanged
- **if**$(*)$ $\mathbf{c_1}$ **else** $\mathbf{c_2}$: arbitrarily choose to run c_1 or c_2
- **assume**$(\mathbf{F})$: block all executions where F does not hold

Given such loop-free program c with conditionals, compute a polynomial-sized formula $R(c)$ of form: $\exists \bar{z}. F(\bar{x}, \bar{z}, \bar{x}')$ describing relation between initial values of variables $x_1, \ldots, x_n$ and final values of variables $x_1', \ldots, x_n'$

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

$\equiv$

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 $x_{fresh} = x + 1; x = x_{fresh}$

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

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

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope.
Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) =$

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

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

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope.
Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) =$
$y' = x + 3 \land z' = 2x + 3 + z \land x' = x$

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

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

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope.
Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) =$
$y' = x + 3 \land z' = 2x + 3 + z \land x' = x$

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

Local Mutable Variables

Assume our global variables are $V = \{x, z\}$
Program P introduces a local variable y inside a nested block:

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

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope.
Inside the block we have variables $V_1 = \{x, y, z\}$. For assignment statement c: $\quad z = x + y + z$,
$R(c)$ is a relation between x, y, z and x', y', z'.
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) =$
$y' = x + 3 \wedge z' = 2x + 3 + z \wedge x' = x$

$R(\{\textbf{var } y; y = x + 3; z = x + y + z\}) = \; z' = 2x + 3 + z \wedge x' = x$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

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

Then define
$R_V(\{var\ \ y; P\}) \ =$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

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

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

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

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

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

Exercise: express havoc(x) using var.

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

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

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

Exercise: express havoc(x) using var.

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

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables V
For example,

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

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

Exercise: express havoc(x) using var.

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

Exercise: give transformation that lifts all variables to be global

Expressing Specifications as Commands

Shorthand: Havoc Multiple Variables at Once

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

Shorthand: Havoc Multiple Variables at Once

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

$$\bigwedge_{v \in V \setminus \{y_1, \ldots, y_m\}} v' = v$$

Exercise: the resulting formula is the same as for:

$$havoc(y_1); \ldots; havoc(y_m)$$

Thus, the order of distinct havoc-s does not matter.

Programs and Specs are Relations

program:	$x = x + 2; y = x + 10$
relation:	$\{(x, y, z, x', y', z') \mid x' = x + 2 \wedge y' = x + 12 \wedge z' = z\}$
formula:	$x' = x + 2 \wedge y' = x + 12 \wedge z' = z$

Specification:

$$z' = z \wedge (x > 0 \rightarrow (x' > 0 \wedge y' > 0)$$

Adhering to specification is relation subset:

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

Non-deterministic programs are a way of writing specifications

Writing Specs Using Havoc and Assume: Examples

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

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

Corresponding program:

Writing Specs Using Havoc and Assume: Examples

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

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

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Writing Specs Using Havoc and Assume: Examples

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

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

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

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

Corresponding program?

Writing Specs Using Havoc and Assume: Examples

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

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

Corresponding program:

$$havoc(x, y); assume(x > 0 \wedge y > 0)$$

Formula for relation:

$$z' = z \wedge x' > x \wedge y' > y$$

Corresponding program?
Use local variables to store initial values.

Writing Specs Using Havoc and Assume: Examples

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

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

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

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

Corresponding program?
Use local variables to store initial values.

```
{ var x0; var y0;
  x0 = x; y0 = y;
  havoc(x,y);
  assume(x > x0 && y > y0)
}
```

Writing Specs Using Havoc and Assume

Global variables $V = \{x_1, \ldots, x_n\}$
Specification

$$F(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$$

Becomes

Writing Specs Using Havoc and Assume

Global variables $V = \{x_1, \ldots, x_n\}$
Specification

$$F(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$$

Becomes

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

Program Refinement and Equivalence

For two programs, define **refinement** $P_1 \sqsubseteq P_2$ iff

$$R(P_1) \rightarrow R(P_2)$$

is a valid formula.

(Some books use the opposite meaning of $\sqsubseteq$.)

As usual, $P_2 \sqsupseteq P_1$ iff $P_1 \sqsubseteq P_2$.

- $P_1 \sqsubseteq P_2$ iff $\rho(P_1) \subseteq \rho(P_2)$

Define **equivalence** $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$

- $P_1 \equiv P_2$ iff $\rho(P_1) = \rho(P_2)$

Example for $V = \{x, y\}$

$$\{var\ x0; x0 = x; havoc(x); assume(x > x0)\} \sqsupseteq (x = x + 1)$$

Proof: Use R to compute formulas for both sides and simplify.

Program Refinement and Equivalence

For two programs, define **refinement** $P_1 \sqsubseteq P_2$ iff

$$R(P_1) \rightarrow R(P_2)$$

is a valid formula.
(Some books use the opposite meaning of $\sqsubseteq$.)
As usual, $P_2 \sqsupseteq P_1$ iff $P_1 \sqsubseteq P_2$.

- $P_1 \sqsubseteq P_2$ iff $\rho(P_1) \subseteq \rho(P_2)$

Define **equivalence** $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$

- $P_1 \equiv P_2$ iff $\rho(P_1) = \rho(P_2)$

Example for $V = \{x, y\}$

$$\{var\ x0; x0 = x; havoc(x); assume(x > x0)\} \sqsupseteq (x = x + 1)$$

Proof: Use R to compute formulas for both sides and simplify.

$$x' = x + 1 \wedge y' = y \ \rightarrow \ x' > x \wedge y' = y$$

Stepwise Refinement Methodology

Start form a possibly non-deterministic specification P_0
Refine the program until it becomes deterministic and efficiently executable.

$$P_0 \sqsupseteq P_1 \sqsupseteq \ldots \sqsupseteq P_n$$

Example:

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

In the last step program equivalence holds as well

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \to (p_1 \circ p) \subseteq (p_2 \circ p)$

Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$
Version for relations: $(p_1 \subseteq p_2) \to (p \circ p_1) \subseteq (p \circ p_2)$

Theorem: if $P_1 \sqsubseteq P_2$ and $Q_1 \sqsubseteq Q_2$ then

$$(if\ \ (*) P_1\ \ else\ \ Q_1) \sqsubseteq (if\ \ (*) P_2\ \ else\ \ Q_2)$$

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p_1 \circ p) \subseteq (p_2 \circ p)$

Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$
Version for relations: $(p_1 \subseteq p_2) \rightarrow (p \circ p_1) \subseteq (p \circ p_2)$

Theorem: if $P_1 \sqsubseteq P_2$ and $Q_1 \sqsubseteq Q_2$ then

$$(if\ (*) P_1\ else\ Q_1) \sqsubseteq (if\ (*) P_2\ else\ Q_2)$$

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

Checking Commutativity of Commands

Associativity of Commands

Under what conditions on commands c_1, c_2 is

$$c_1; (c_2; c_3) \;\equiv\; (c_1; c_2); c_3$$

Associativity of Commands

Under what conditions on commands c_1, c_2 is

$$c_1; (c_2; c_3) \;\equiv\; (c_1; c_2); c_3$$

always

Commutativity of Commands

Under what conditions on commands c_1, c_2 is

$$c_1; c_2 \ \equiv \ c_2; c_1$$

Commutativity of Commands

Under what conditions on commands c_1, c_2 is

$$c_1; c_2 \equiv c_2; c_1$$

In general, when the resulting relations are equal and formulas equivalent, i.e. iff

$$R(c_1; c_2) \iff R(c_2; c_1)$$

is a valid formula (true for all variables).

Commutativity of Commands

Under what conditions on commands c_1, c_2 is

$$c_1; c_2 \;\equiv\; c_2; c_1$$

In general, when the resulting relations are equal and formulas equivalent, i.e. iff

$$R(c_1; c_2) \iff R(c_2; c_1)$$

is a valid formula (true for all variables).
Example: does this hold?

$$(x = x + 1; y = x + 2) \;\equiv\; (y = x + 2; x = x + 1)$$

Show formulas for each sides

Commutativity of Commands

Under what conditions on commands c_1, c_2 is

$$c_1; c_2 \;\equiv\; c_2; c_1$$

In general, when the resulting relations are equal and formulas equivalent, i.e. iff

$$R(c_1; c_2) \iff R(c_2; c_1)$$

is a valid formula (true for all variables).
Example: does this hold?

$$(x = x + 1; y = x + 2) \;\equiv\; (y = x + 2; x = x + 1)$$

Show formulas for each sides—not equivalent:

$$x' = x + 1 \land y' = x + 3 \qquad x' = x + 1 \land y' = x + 2$$

Examples of Commutativity of Commands

Show the formula for each example and check if the commutativity equivalence holds

Example 1:

$$(x = 2*x+7*z;\ y = 5*y+z) \ \equiv\ (y = 5*y+z;\ x = 2*x+7*z)$$

Examples of Commutativity of Commands

Show the formula for each example and check if the commutativity equivalence holds

Example 1:

$$(x = 2{*}x{+}7{*}z;\ y = 5{*}y{+}z) \;\equiv\; (y = 5{*}y{+}z;\ x = 2{*}x{+}7{*}z)$$

Can you state a generalization of the above example?

Examples of Commutativity of Commands

Show the formula for each example and check if the commutativity equivalence holds

Example 1:

$$(x = 2*x+7*z;\; y = 5*y+z) \;\equiv\; (y = 5*y+z;\; x = 2*x+7*z)$$

Can you state a generalization of the above example?
Example 2:

$$(x = x + 1; x = x + 5) \;\equiv\; (x = x + 5; x = x + 1)$$

Examples of Commutativity of Commands

Show the formula for each example and check if the commutativity equivalence holds

Example 1:

$$(x = 2*x+7*z;\ y = 5*y+z) \ \equiv \ (y = 5*y+z;\ x = 2*x+7*z)$$

Can you state a generalization of the above example?
Example 2:

$$(x = x + 1; x = x + 5) \ \equiv \ (x = x + 5; x = x + 1)$$

Requires knowing properties of $+$.

Preserving Domain in Refinement

What is the domain of a relation?

Given relation $r \subseteq A \times B$ for any sets A, B, we define domain of r as

$$dom(r) = \{a \mid \exists b.\ (a, b) \in r\}$$

when r is a total function, then $dom(r) = A$

- a typical case if r is an entire program

Let $r = \{(\bar{x}, \bar{x}') \mid F\}$, $FV(F) \subseteq Var \cup Var'$, $Var' = \{x' \mid x \in Var\}$.
Then, $dom(r) = \{\bar{x} \mid \exists \bar{x}'.F\}$

- computing domain = existentially quantifying over primed vars

Example: for $Var = \{x, y\}$, $R(x = x + 1) = x' = x + 1 \wedge y' = y$.
The formula for the domain is: $\exists x', y'.\ x' = x + 1 \wedge y' = y$,
which, after one-pint rule, reduces to true.

- All assignments have true as domain.

Preserving Domain

It is not interesting program development step $P \sqsupseteq P'$ is P' is false, or is false for most inputs.
Example ($Var = \{x, y\}$)

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

Refinement $P \sqsupseteq Q$, ensures $R(Q) \rightarrow R(P)$. A consequence is $(\exists \bar{x}'.R(Q)) \rightarrow (\exists \bar{x}'.R(P))$.
We additionally wish to preserve the *domain* of the relation between $\bar{x}, \bar{x}'$

- if P has some execution from $\bar{x}$ ending in $\bar{x}'$
- then Q should also have some execution, ending in some (possibly different) $\bar{x}'$ (even if it has fewer choices)
$$(\exists \bar{x}'.R(P)) \leftrightarrow (\exists \bar{x}'.R(Q))$$

So, we want relations to be smaller or equal, but domains equal.

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) =$

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') =$

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold?

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold? yes

Now consider the right hand side:

- domain of P is

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold? yes

Now consider the right hand side:

- domain of P is $\exists x', y'.x' + x' = y \wedge y' = y$
- equivalent to:

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold? yes

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y \% 2 = 0$
- domain of P is:

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold? yes

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y\%2 = 0$
- domain of P is: $\exists x', y'. x' = 3 \wedge y' = 6 \wedge y' = y$
- equivalent to:

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold? yes

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y \% 2 = 0$
- domain of P is: $\exists x', y'. x' = 3 \wedge y' = 6 \wedge y' = y$
- equivalent to: $y = 6$

Does domain formula of P' imply the domain formula of P?

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y' \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold? yes

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y \% 2 = 0$
- domain of P is: $\exists x', y'. x' = 3 \wedge y' = 6 \wedge y' = y$
- equivalent to: $y = 6$

Does domain formula of P' imply the domain formula of P? no

Preserving Domain: Exercise

Given P:

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

Find P_1 and P_2 such that

- $P \sqsupseteq P_1 \sqsupseteq P_2$
- no two programs among P, P_1, P_2 are equivalent
- programs P, P_1 and P_2 have equivalent domains
- the relation described by P_2 is a partial function