

# Converting Imperative Programs to Formulas

Viktor Kunčák

# Verifying Imperative Programs

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

# Verification-Condition Generation for Imperative Non-Deterministic Programs

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

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

Specification:  $old(x) > 0 \rightarrow x > 0 \wedge y > 0$

Adhering to specification is relation subset:

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

or validity of the following implication:

$$\begin{aligned} & x' = x + 2 \wedge y' = x + 12 \\ \rightarrow & (x > 0 \rightarrow (x' > 0 \wedge y' > 0)) \end{aligned}$$

## Simple Imperative Programs

$F$  - formulas,  $t$  - terms (with only pure mathematical operations)

Fixed number of mutable variables  $V = \{x_1, \dots, x_n\}$

Imperative statements:

- ▶  **$x = t$** : change  $x \in V$  to have value given by  $t$ ; leave vars in  $V \setminus \{x\}$  unchanged
- ▶ **if( $F$ ) $c_1$  else  $c_2$** : if  $F$  holds, execute  $c_1$  else execute  $c_2$
- ▶  **$c_1; c_2$** : 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
- ▶ **if( $*$ )  $c_1$  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  $x_1, \dots, x_n$  and final values of variables  $x'_1, \dots, x'_n$

# 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

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

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

## 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.

Examples:

$$R(x = x + 2) = x' = x + 2 \wedge y' = y$$

$$R(y = x + 10) = x' = x \wedge y' = x + 10$$

## Formula for if-else

*if*(*b*) *c*<sub>1</sub> *else* *c*<sub>2</sub>



## Formula for if-else

*if*( $b$ )  $c_1$  *else*  $c_2$

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

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

## Command semicolon

$c_1; c_2$

## Command semicolon

$c_1; c_2$

Corresponds to relation composition:

$$r_1 \circ r_2 = \{(\bar{x}, \bar{x}') \mid \exists \bar{x}'' . (\bar{x}, \bar{x}'') \in r_1 \wedge (\bar{x}'', \bar{x}') \in r_2\}$$

## Command semicolon

$c_1; c_2$

Corresponds to relation composition:

$$r_1 \circ r_2 = \{(\bar{x}, \bar{x}') \mid \exists \bar{x}'' . (\bar{x}, \bar{x}'') \in r_1 \wedge (\bar{x}'', \bar{x}') \in r_2\}$$

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

## Command semicolon

$c_1; c_2$

Corresponds to relation composition:

$$r_1 \circ r_2 = \{(\bar{x}, \bar{x}') \mid \exists \bar{x}'' . (\bar{x}, \bar{x}'') \in r_1 \wedge (\bar{x}'', \bar{x}') \in r_2\}$$

What are  $R(c_1)$  and  $R(c_2)$  and in terms of which variables they are expressed? Each in terms of  $\bar{x}$  and  $\bar{x}'$ .

Let  $r_1 = \{(\bar{x}, \bar{x}') \mid R(c_1)\}$ ,  $r_2 = \{(\bar{x}, \bar{x}') \mid R(c_2)\}$

Thus,

$$(\bar{x}, \bar{x}'') \in r_1 \iff (\bar{x}, \bar{x}'') \in \{(\bar{x}, \bar{x}') \mid R(c_1)\} \iff R(c_1)[\bar{x}' := \bar{x}'']$$

Similarly,  $(\bar{x}'', \bar{x}') \in r_2 \iff R(c_2)[\bar{x}' := \bar{x}'']$

## Command semicolon

$c_1; c_2$

Corresponds to relation composition:

$$r_1 \circ r_2 = \{(\bar{x}, \bar{x}') \mid \exists \bar{x}'' . (\bar{x}, \bar{x}'') \in r_1 \wedge (\bar{x}'', \bar{x}') \in r_2\}$$

What are  $R(c_1)$  and  $R(c_2)$  and in terms of which variables they are expressed? Each in terms of  $\bar{x}$  and  $\bar{x}'$ .

Let  $r_1 = \{(\bar{x}, \bar{x}') \mid R(c_1)\}$ ,  $r_2 = \{(\bar{x}, \bar{x}') \mid R(c_2)\}$

Thus,

$$(\bar{x}, \bar{x}'') \in r_1 \iff (\bar{x}, \bar{x}'') \in \{(\bar{x}, \bar{x}') \mid R(c_1)\} \iff R(c_1)[\bar{x}' := \bar{x}'']$$

$$\text{Similarly, } (\bar{x}'', \bar{x}') \in r_2 \iff R(c_2)[\bar{x}' := \bar{x}'']$$

$$R(c_1; c_2) \iff (\bar{x}, \bar{x}') \in r_1 \circ r_2 \iff$$

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

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

- ▶ a useful convention:  $\bar{x}''$  refer to position in program source code,  $\bar{x}^i$

## Computing relation for the example from before

$$\begin{aligned}R(x = x + 2; y = x + 10) &= \exists \bar{x}'' . R(c_1)[\bar{x}' := \bar{x}''] \wedge R(c_2)[\bar{x} := \bar{x}''] \\&= \exists x'', y'' . (x' = x + 2 \wedge y' = y)[x' := x'', y' := y''] \wedge \\&\quad (x' = x \wedge y' = x + 10)[x := x'', y := y''] \\&= \exists x'', y'' . (x'' = x + 2 \wedge y'' = y) \wedge \\&\quad (x' = x'' \wedge y' = x'' + 10) \quad (*) \\&\iff (x' = x + 2 \wedge y' = x + 2 + 10) \\&\iff (x' = x + 2 \wedge y' = x + 12)\end{aligned}$$

Where at step (\*) we used (twice) the “one-point rule” of logic with equality:

$$(\exists u.(u = t \wedge F)) \iff F[u := t]$$

if  $u \notin FV(t)$ .

# 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)
```

ends up dividing  $x$  by two!

Translation,  $R(\text{havoc}(x))$ :



# 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)
```

ends up dividing  $x$  by two!

Translation,  $R(\text{havoc}(x))$ :

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

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

## Non-deterministic choice

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

## Non-deterministic choice

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

$R(\textit{if}(\ast) c_1 \textit{ else } c_2)$ :

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

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

assume

*assume(F)*

assume

*assume*( $F$ )

$R(\text{assume}(F))$ :

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

assume

*assume*( $F$ )

$R(\text{assume}(F)):$

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

- ▶ This command does not change any state.

assume

*assume*( $F$ )

$R(\text{assume}(F))$ :

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

- ▶ This command does not change any state.
- ▶ If  $F$  does not hold, it stops with “instantaneous success”.

## Example of Translation

```
0
  (if (b) x = x + 1 else y = x + 2);
1
x = x + 5;
2
  (if (*) y = y + 1 else x = y)
3
```

becomes

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

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

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



## Justifying the name for $\text{assume}(F)$

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

1.  $R(\text{assume}(F); c)$

## Justifying the name for $\text{assume}(F)$

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

1.  $R(\text{assume}(F); c) = F \wedge R(c)$
2.  $R(c; \text{assume}(F))$

## Justifying the name for $\text{assume}(F)$

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

1.  $R(\text{assume}(F); c) = F \wedge R(c)$

2.  $R(c; \text{assume}(F)) = R(c) \wedge F[\bar{x} := \bar{x}']$

where  $F[\bar{x} := \bar{x}']$  denotes  $F$  with all variables replaced with primed versions

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

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

```
if (b) c1 else c2
```

|||

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

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

Expressing assignment through havoc and assume

# Expressing assignment through havoc and assume

$x = e$

|||

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

Under what conditions this holds?

## Expressing assignment through havoc and assume

$x = e$

|||

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

Under what conditions this holds?

$x \notin FV(e)$

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



## Expressing assignment through havoc and assume

$x = e$

|||

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

Under what conditions this holds?

$x \notin FV(e)$

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

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

## Loop-Free Programs as Relations: Summary

command $c$	$R(c)$	$\rho(c)$
$(x = t)$	$x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$	
$c_1 ; c_2$	$\exists \bar{z}. R(c_1)[\bar{x}' := \bar{z}] \wedge R(c_2)[\bar{x} := \bar{z}]$	$\rho(c_1) \circ \rho(c_2)$
<b>if</b> (*) $c_1$ <b>else</b> $c_2$	$R(c_1) \vee R(c_2)$	$\rho(c_1) \cup \rho(c_2)$
<b>assume</b> ( $F$ )	$F \wedge \bigwedge_{v \in V} v' = v$	$\Delta_{S(F)}$

$\rho(x_i = t) = \{((x_1, \dots, x_i, \dots, x_n), (x_1, \dots, x'_i, \dots, x_n) \mid x'_i = t)\}$

$S(F) = \{\bar{x} \mid F\}$ ,  $\Delta_A = \{(\bar{x}, \bar{x}) \mid \bar{x} \in A\}$  (diagonal relation on  $A$ )

$\Delta$  (without subscript) is identity on entire set of states (no-op)

We always have:  $\rho(c) = \{(\bar{x}, \bar{x}') \mid R(c)\}$

Shorthands:

$$\frac{\mathbf{if}(*)\ c_1\ \mathbf{else}\ c_2}{\mathbf{assume}(F)} \quad \Bigg| \quad \frac{c_1 \sqcap c_2}{[F]}$$

Examples:

$$\mathbf{if}(F)\ c_1\ \mathbf{else}\ c_2 \equiv [F]; c_1 \sqcap [\neg F]; c_2$$

$$\mathbf{if}(F)\ c \equiv [F]; c \sqcap [\neg F]$$

# Program Paths

## Loop-Free Programs

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

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

<pre>(if (x &gt; 0)   x = x - 1 else   x = 0 ); (if (y &gt; 0)   y = y - 1 else   y = x + 1 )</pre>	<pre>([x &gt; 0]; x = x - 1 ∪ [¬(x&gt;0)]; x = 0) ); ([y &gt; 0]; y = y - 1 ∪ [¬(y&gt;0)]; y = x+1 )</pre>	<pre>(<math>\Delta_{S(x&gt;0)} \circ \rho(x = x - 1)</math> ∪ <math>\Delta_{S(\neg(x&gt;0))} \circ \rho(x = 0)</math> ) <math>\circ</math> (<math>\Delta_{S(y&gt;0)} \circ \rho(y = y - 1)</math> ∪ <math>\Delta_{S(\neg(y&gt;0))} \circ \rho(y = x + 1)</math> )</pre>
---	--	---

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:

$$\begin{aligned} & \left( \begin{array}{l} \Delta_1 \circ r_1 \\ \cup \\ \Delta_2 \circ r_2 \end{array} \right) \circ \left( \begin{array}{l} \Delta_3 \circ r_3 \\ \cup \\ \Delta_4 \circ r_4 \end{array} \right) \\ & \equiv \begin{array}{l} \Delta_1 \circ r_1 \circ \Delta_3 \circ r_3 \cup \\ \Delta_1 \circ r_1 \circ \Delta_4 \circ r_4 \cup \\ \Delta_2 \circ r_2 \circ \Delta_3 \circ r_3 \cup \\ \Delta_2 \circ r_2 \circ \Delta_4 \circ r_4 \end{array} \end{aligned}$$

Sequential composition of basic statements is called basic path.

Loop-free code describes finitely many (exponentially many) paths.

# 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”  $\rightsquigarrow$  smallest set:  $\emptyset$
- ▶ weakest condition: “true”  $\rightsquigarrow$  biggest set: set of all tuples

# Hoare Triples



## About Hoare Logic

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

Example proof:

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

# Hoare Triple and Friends

Sir Charles Antony Richard Hoare



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

Born 11 January 1934

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

**Hoare Triple:**

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

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

**Strongest postcondition:**

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

**Weakest precondition:**

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

# Postconditions and Their Strength

What is the relationship between these postconditions?

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

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

## Postconditions and Their Strength

What is the relationship between these postconditions?

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

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

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

that satisfy a given property.

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

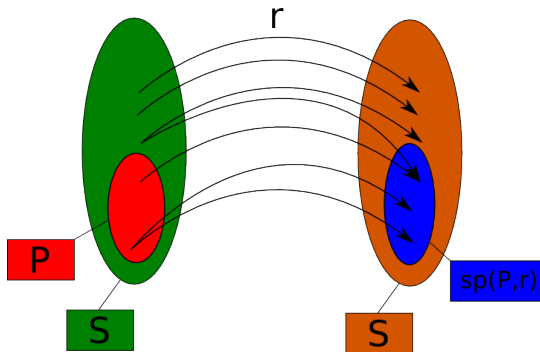
Strongest Postconditions

## Strongest Postcondition

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

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

This is simply the relation image of a set.



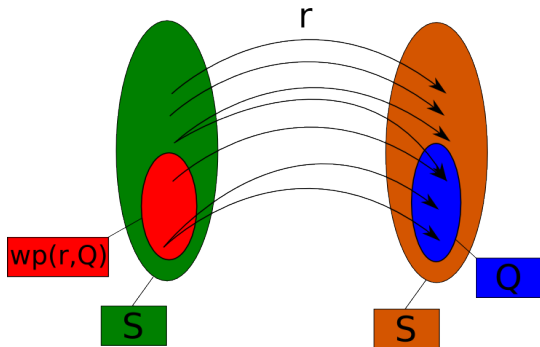
# Weakest Preconditions

## Weakest Precondition

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

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

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





# Three Forms of Hoare Triple

Lemma: the following three conditions are equivalent:

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

# Three Forms of Hoare Triple

Lemma: the following three conditions are equivalent:

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

Proof. The three conditions expand into the following three formulas

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

which are easy to show equivalent using basic first-order logic properties, such as  $(P \wedge Q \rightarrow R) \longleftrightarrow (P \rightarrow (Q \rightarrow R))$ ,  $(\forall u.(A \rightarrow B)) \longleftrightarrow (A \rightarrow \forall u.B)$  when  $u \notin FV(A)$ , and  $(\forall u.(A \rightarrow B)) \longleftrightarrow ((\exists u.A) \rightarrow B)$  when  $u \notin FV(B)$ .