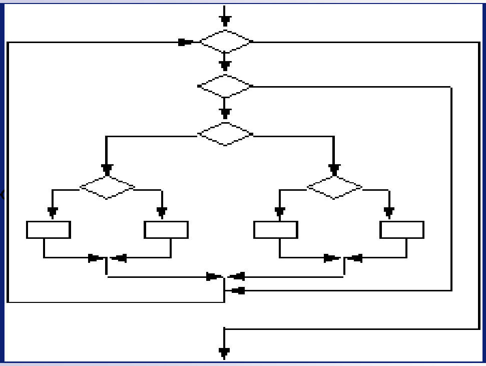
Spring 2008



Program correctness

Axiomatic semantics

Marcello Bonsangue

Leiden Institute of Advanced Computer Science
Research & Education

Axiomatic Semantics

- We have introduced
  - a syntax for sequential programs
  - An operational semantics (transition system) for "running" those programs from a starting state. A computation may terminate in a state or run forever.

- We would also like to have a semantics for reasoning about program correctness

Axiomatic semantics

- We need
  - A logical language for making assertions about programs
    - The program terminates
    - If x = 0 then y = z+1 throughout the rest of the execution of the program
    - If the program terminates, then x = y + z
  - A proof system for establishing those assertions

Why axiomatic semantics

- Documentation of programs and interfaces (Meyer's Design by Contract)
- Guidance in language design and coding
- Proving the correctness of algorithms
- Extended static checking
  - checking array bounds
- Proof-carrying code

- Why not testing?
  - Dijkstra: *Program testing can be used to show the presence of bugs, but never to show their absence!*

The idea

"*Compute a number y whose square is less than the input x*"

↓

We have to write a program P such that

$$y*y < x$$

But what if x = -4?

There is no program computing y!!

The idea (continued)

"*If the input x is a positive number then compute a number y whose square is less than the input x*"

We need to talk about the states before and after the execution of the program P

$$\{ x>0 \}\ P\ \{ y*y < x \}$$

The idea (continued)

- Hoare triple for partial correctness

precondition — postcondition — command

$$\models_{par} \{\phi\}\ c\ \{\psi\}$$

If the command c terminates when it is executed in a state that satisfies ϕ, then the resulting state will satisfy ψ

program termination is not required

Examples

- $\models_{par}$ { $y \leq x$ } z := x; z := z +1 { $y < z$ } is valid
- $\models_{par}$ { true } <u>while</u> true <u>do</u> skip <u>od</u> { false } is valid
- Let Fact = y := 1; z := 0;

```
          while z ≠ x do
                z := z + 1;
                y := y*z
          od
```

Is $\models_{par}$ { $x \geq 0$ } Fact { $y = x!$ } valid?

Total correctness

- Hoare triple for total correctness

precondition

postcondition

$$\models_{tot} \{\phi\}\ c\ \{\psi\}$$

command

If the command c is executed in a state that satisfies ϕ then c is guaranteed to terminate and the resulting state will satisfy ψ

program termination is required

Example

- $\models_{tot}$ { y $\leq$ x } z := x; z := z +1 { y < z } is valid
- $\models_{tot}$ { true } while true do skip od { false } is not valid
- $\models_{tot}$ { false } while true do skip od { true } is valid
- Let Fact = y := 1; z := 0;

```
                  while z ≠ x do
                        z := z + 1;
                        y := y*z
                  od
```

Is $\models_{tot}$ { x $\geq$ 0 } Fact { y = x! } valid?

Partial and total correctness: meaning

- Hoare triple for partial correctness $\vDash_{par} \{\phi\}\ c\ \{\psi\}$

  If ϕ holds in a state σ and $\langle c,\sigma\rangle \to \sigma'$ then
  ψ holds in σ'

- Hoare triple for total correctness $\vDash_{tot} \{\phi\}\ c\ \{\psi\}$

  If ϕ holds in a state σ then
  there exists a σ' such that $\langle c,\sigma\rangle \to \sigma'$ and ψ holds in σ'

- To be more precise, we need to:
  - Formalize the language of assertions for ϕ and ψ
  - Say when an assertion holds in a state.
  - Give rules for deriving Hoare triples

The assertion language

- Extended arithmetic expressions

$$a ::= n \mid x \mid i \mid (a+a) \mid (a-a) \mid (a*a)$$

$n \in N$, $x \in Var$, $i \in LVar$

- Assertions (or extended Boolean expressions)

$$\phi ::= true \mid \neg\phi \mid \phi \wedge \phi \mid a < a \mid \forall i.\phi$$

$i \in LVar$

Program variables

- We need program variables Var in our assertion language
  - To express properties of a state of a program as basic assertion such as

$$x = n \qquad \text{i.e. "}\textit{The value of x is } \text{n"}$$

that can be used in more complex formulas such as

$$x = n \Rightarrow y+1 = x*(y-x) \qquad \text{i.e. "}\textit{If the value of x is } \text{n } \textit{then that of y + 1 is x times y - x}\text{"}$$

Logical variables

- We need a set of logical variables LVar
  - To express mathematical properties such as

    $\exists i.\ n = i * m$ i.e. "*an integer* n *is multiple of another* m"

  - To remember the value of a program variable destroyed by a computation

    Fact2 ≡ y := 1;
    <u>while</u> x ≠ 0 <u>do</u>
    y := y*x;
    x := x – 1
    <u>od</u>

    $\models_{par}\{ x \geq 0 \}$ Fact2 $\{ y = x! \}$ is not valid but

    $\models_{par}\{ x = x_0 \wedge x \geq 0 \}$ Fact2 $\{ y = x_0! \}$ is.

Meaning of assertions

- Next we assign meaning to assertions
  - Problem: "ϕ holds in a state σ" may depends on the value of the logical variables in ϕ
  - Solution: use interpretations of logical variables
  - Examples
    - $z < x$ holds in a state $\sigma : Var \rightarrow N$ with $\sigma(x) = 3$ for all interpretations $I : LVar \rightarrow N$ of the logical variables such that $I(i) < 3$
    - $i < i+1$ holds in a state for all interpretations

Meaning of expressions

- Given a state σ:Var $\to$ N and an interpretation I:LVar$\to$N we define the meaning of an expression e as [[e]]Iσ, inductively given by
  - [[n]]Iσ = n
  - [[x]]Iσ = σ(x)
  - [[i]]Iσ = I(i)
  - [[a_1+a_2]]Iσ = [[a_1]]Iσ +[[a_2]]Iσ
  - [[a_1-a_2]]Iσ = [[a_1]]Iσ - [[a_2]]Iσ
  - [[a_1*a_2]]Iσ = [[a_1]]Iσ *[[a_2]]Iσ

Meaning of assertions

- Given a state $\sigma : Var \to N$ and an interpretation $I : LVar \to N$ we define

$$\sigma, I \models \phi$$

inductively by

  - $\sigma, I \models true$
  - $\sigma, I \models \neg\phi$ iff not $\sigma, I \models \phi$
  - $\sigma, I \models \phi \wedge \psi$ iff $\sigma, I \models \phi$ and $\sigma, I \models \psi$
  - $\sigma, I \models a_1 < a_2$ iff $[[a_1]]I\sigma < [[a_2]]I\sigma$
  - $\sigma, I \models \forall i.\phi$ iff $\sigma, I[n/i] \models \phi$ for all $n \in N$

Partial and total correctness

- Partial correctness: $I \models_{par} \{\phi\}\ c\ \{\psi\}$

$$\forall \sigma\ (\sigma, I \models \phi \text{ and } <c, \sigma> \rightarrow \sigma'\ ) \Rightarrow \sigma', I \models \psi$$

- Total correctness: $I \models_{tot} \{\phi\}\ c\ \{\psi\}$

$$\forall \sigma.\ \sigma, I \models \phi \Rightarrow \exists\ \sigma'.(<c, \sigma> \rightarrow \sigma' \text{ and } \sigma', I \models \psi)$$

where ϕ and ψ are assertions and c is a command

Validity

- To give an absolute meaning to

$$\{i < x\}\ x := x+3\ \{i < x\}$$

  we have to quantify over all interpretations I

- Partial correctness:

$$\models_{par} \{\phi\}\ c\ \{\psi\} \quad \equiv \quad \forall I.\ I \models_{par} \{\phi\}\ c\ \{\psi\}$$

- Total correctness:

$$\models_{tot} \{\phi\}\ c\ \{\psi\} \quad \equiv \quad \forall I.\ I \models_{tot} \{\phi\}\ c\ \{\psi\}$$

Deriving assertions

- We have the meaning of both

$$\models_{par} \{\phi\}\ c\ \{\psi\} \quad \text{and} \quad \models_{tot} \{\phi\}\ c\ \{\psi\}$$

  but it depends on the operational semantics and it cannot be effectively used
- Thus we want to define a proof system to derive symbolically valid assertions from valid assertions.
  - $\vdash_{par} \{\phi\}c\{\psi\}$ means that the Hoare triple $\{\phi\}c\{\psi\}$ can be derived by some axioms and rules
  - Similarly for $\vdash_{tot} \{\phi\}c\{\psi\}$

Free and bound variables

- A logical variable is bound in an assertion if it occurs in the scope of a quantifier

$$\exists i.\ n = i * m$$

- A logical variable is free if it is not bound

$$i + 100 < 77 \wedge \forall i.\ j+i = 3$$

free

bound

Substitution (I)

- For an assertion ϕ, logical variable i and arithmetic expression e we define

  $$\phi[e/i]$$

  as the assertion resulting by substituting in ϕ the free occurrence of i by e.

- Definition for extended arithmetic expressions

n[e/i] = n	$(a_1+a_2)[e/i]=(a_1[e/i]+a_2[e/i])$
x[e/i] = x	$(a_1-a_2)[e/i]=(a_1[e/i]-a_2[e/i])$
i[e/i] = e	$(a_1*a_2)[e/i]=(a_1[e/i]*a_2[e/i])$
j[e/i] = j	

Substitution (II)

- Definition for assertions

$\text{true}[e/i]$	$= \text{true}$	
$(\neg\phi)[e/i]$	$= \neg(\phi[e/i])$	
$(\phi_1 \wedge \phi_2)[e/i]$	$= (\phi_1[e/i] \wedge \phi_2[e/i])$	
$(a_1 < a_2)[e/i]$	$= (a_1[e/i] < a_2[e/i])$	
$(\forall i.\phi)[e/i]$	$= \forall i.\phi$	
$(\forall j.\phi)[e/i]$	$= \forall j.\phi[e/i]$	$j \neq i$

- Pictorially, if ϕ = ---i--i--i- with i free, then

$\phi[e/i]$ = ---e--e--e-

Proof rules partial correctness (I)

- There is one derivation rule for each command in the language.
  - $\{\phi\}$ skip $\{\phi\}$ skip
  - $\{\phi[a/x]\}$ x := a $\{\phi\}$ ass
  - $$\frac{\{\phi\}\ c_1\ \{\psi\} \qquad \{\psi\}\ c_2\ \{\varphi\}}{\{\phi\}\ c_1;\ c_2\ \{\varphi\}}\ \text{seq}$$

Proof rules partial correctness (II)

- $$\frac{\{\phi \wedge b\}\ c_1\ \{\psi\} \qquad \{\phi \wedge \neg b\}\ c_2\ \{\psi\}}{\{\phi\}\ \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2\ \underline{\text{fi}}\ \{\psi\}} \qquad \text{if}$$

- $$\frac{\{\phi \wedge b\}\ c\ \{\phi\}}{\{\phi\}\ \underline{\text{while}}\ b\ \underline{\text{do}}\ c\ \underline{\text{od}}\ \{\phi \wedge \neg b\}} \qquad \text{while}$$

- $$\frac{\vdash \phi \Rightarrow \phi' \qquad \{\phi'\}\ c\ \{\psi'\} \qquad \vdash \psi' \Rightarrow \psi}{\{\phi\}\ c\ \{\psi\}} \qquad \text{cons}$$

A first example: assignment

- Let's prove that

$$\vdash_{par} \{true\}\ x:=1\ \{x=1\}$$

$$\dfrac{\vdash true \Rightarrow 1=1 \qquad \dfrac{}{\{1=1\}\ x:=1\ \{x=1\}}\ \text{ass}}{\{true\}\ x:=1\ \{x=1\}}\ \text{cons}$$

Another example: assignment

- Prove that {true} x:= e {x=e} when x does not appear in e

1. Because x does ot appear in e we have

$$(x=e)[e/x] \equiv (x[e/x]=e[e/x]) \equiv (e=e)$$

2. Use assignment + consequence to obtain the proof

$$\dfrac{\vdash true \Rightarrow e=e \qquad \dfrac{}{\{e=e\}\ x:=e\ \{x=e\}}\ ass}{\{true\}\ x:=e\ \{x=e\}}\ cons$$

Another example: conditional

- Prove $\vdash_{par}$ {true} <u>if</u> y$\leq$1 <u>then</u> x:=1 <u>else</u> x:=y <u>fi</u> {x>0}

```
                                  ---------------------- ass                                  ass ---------------------
⊢true ∧ y≤1⇒1>0   {1>0} x:=1 {x>0}          ⊢true ∧ y>1 ⇒y>0    {y>0} x:=y {x>0}
------------------------------------------------ cons ------------------------------------------------
              {true ∧ y ≤1} x:=1 {x>0}          {true ∧ y >} x:=y {x>0}
        ---------------------------------------------------------------------------- if
                  {true} if y≤1 then x:=1 else x:=y fi {x>0}
```

An example: while

- Prove $\vdash_{par}$ {0 ≤ x} <u>while</u> x>0 <u>do</u> x:=x-1 <u>od</u> {x=0}

  We take as invariant 0 ≤ x in the while-rule

$$\dfrac{\dfrac{\vdash 0 \leq x \wedge x>0 \Rightarrow 0 \leq x-1 \qquad \dfrac{}{\{0 \leq x-1\}\ x:=x-1\ \{0 \leq x\}}\ \text{ass}}{\dfrac{\{0 \leq x \wedge x>0\}\ x:=x-1\ \{0 \leq x\}}{\{0 \leq x\}\ \underline{\text{while}}\ x>0\ \underline{\text{do}}\ x:=x-1\ \underline{\text{od}}\ \{0 \leq x \wedge x \leq 0\}}\ \text{while}}\ \text{cons} \qquad \vdash 0 \leq x \wedge x \leq 0 \Rightarrow x=0}{\{x \leq 0\}\ x>0\ \underline{\text{do}}\ x:=x-1\ \underline{\text{od}}\ \{x=0\}}\ \text{cons}$$

An example: while, again

Prove that $\{x \leq 0\}$ <u>while</u> $x \leq 5$ <u>do</u> $x:=x+1$ <u>od</u> $\{x=6\}$

1. We start with the invariant $x \leq 6$ in the while-rule

$$\dfrac{\vdash x \leq 6 \wedge x \leq 5 \Rightarrow x+1 \leq 6 \qquad \dfrac{}{\{x+1 \leq 6\}x:=x+1\ \{x \leq 6\}}\ \text{ass}}{\dfrac{\{x \leq 6 \wedge x \leq 5\}\ x:=x+1\ \{x \leq 6\}}{\{x \leq 6\}\ \underline{\text{while}}\ x \leq 5\ \underline{\text{do}}\ x:=x+1\ \underline{\text{od}}\ \{x \leq 6 \wedge x>5\}}\ \text{while}}\ \text{cons}$$

2. We finish with the consequence rule

$$\dfrac{\vdash x\leq 0 \Rightarrow x\leq 6 \qquad \{x \leq 6\}\ \underline{\text{while}}\ x \leq 5\ \underline{\text{do}}\ x:=x+1\ \underline{\text{od}}\ \{x \leq 6 \wedge x>5\} \qquad \vdash x\leq 6 \wedge x>5 \Rightarrow x=6}{\{x \leq 0\}\ \underline{\text{while}}\ x \leq 5\ \underline{\text{do}}\ x:=x+1\ \underline{\text{od}}\ \{x=6\}}$$

Auxiliary rules

- They can be derived from the previous ones
  - $\{\phi\}\ c\ \{\phi\}$ if the program variables in ϕ do not appear in c
  - $\{\phi\}\ x := a\ \{\exists x_0.(\phi[x_0/x] \wedge x = a[x_0/x])\}$
  - $$\frac{\{\phi_1\}\ c_1\ \{\psi\} \qquad \{\phi_2\}\ c_2\ \{\psi\}}{\{(b \Rightarrow \phi_1) \wedge (\neg b \Rightarrow \phi_2)\}\ \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2\ \underline{\text{fi}}\ \{\psi\}}$$
  - $$\frac{\{\phi_1\}\ c\ \{\psi\} \qquad \{\phi_2\}\ c\ \{\psi\}}{\{\phi_1 \vee \phi_2\}\ c\ \{\psi\}}$$
  - $$\frac{\{\phi_1\}\ c\ \{\psi_1\} \qquad \{\phi_2\}\ c\ \{\psi_2\}}{\{\phi_1 \wedge \phi_2\}\ c\ \{\psi_1 \wedge \psi_2\}}$$

Comments on Hoare logic

- The rules are syntax directed
  - Three problems:
    - When to apply the consequence rule
    - How to prove the implication in the consequence rule
    - What invariant to use in the while rule
- The last is the real hard one
  - Should it be given by the programmer?

An extensive example: a program

DIV $\triangleq$

```
q := 0;
r := x;
while r ≥ y do
        r := r-y;
        q := q+1
od
```

We wish to prove

$\{x \geq 0 \wedge y > 0\}$ DIV $\{ q*y+r=x \wedge 0 \leq r<y \}$