Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
http://courses.engr.illinois.edu/cs421

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Axiomatic Semantics

- Also called Floyd-Hoare Logic
- Based on formal logic (first order predicate calculus)
- Axiomatic Semantics is a logical system built from *axioms* and *inference rules*
- Mainly suited to simple imperative programming languages

Axiomatic Semantics

- Used to formally prove a property (*post-condition*) of the *state* (the values of the program variables) after the execution of program, assuming another property (*pre-condition*) of the state holds before execution

Axiomatic Semantics

- Goal: Derive statements of form
  {P} C {Q}
  - P , Q logical statements about state, P precondition, Q postcondition, C program
- Example: {x = 1} x := x + 1 {x = 2}

Axiomatic Semantics

- *Approach:* For each type of language statement, give an axiom or inference rule stating how to derive assertions of form
  {P} C {Q}
  where C is a statement of that type
- Compose axioms and inference rules to build proofs for complex programs

Axiomatic Semantics

- An expression {P} C {Q} is a *partial correctness* statement
- For *total correctness* must also prove that C terminates (i.e. doesn't run forever)
  - Written: [P] C [Q]
- Will only consider partial correctness here

Language

- We will give rules for simple imperative language

<command>
 ::= <variable> := <term>
 | <command>; … ;<command>
 | if <statement> then <command> else <command>
 | while <statement> do <command>

- Could add more features, like for-loops

Substitution

- Notation: P[e/v] (sometimes P[v <- e])
- Meaning: Replace every v in P by e
- Example:

  (x + 2) [y-1/x] = ((y – 1) + 2)

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Example:

$$\frac{}{\{\quad ?\quad\}\ x := y\ \{x = 2\}}$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Example:

$$\frac{}{\{\_ = 2\}\ x := y\ \{x = 2\}}$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Example:

$$\frac{}{\{y = 2\}\ x := y\ \{x = 2\}}$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Examples:

$$\frac{}{\{y = 2\}\ x := y\ \{x = 2\}}$$

$$\frac{}{\{y = 2\}\ x := 2\ \{y = x\}}$$

$$\frac{}{\{x + 1 = n + 1\}\ x := x + 1\ \ \{x = n + 1\}}$$

$$\frac{}{\{2 = 2\}\ x := 2\ \{x = 2\}}$$

The Assignment Rule – Your Turn

- What is the weakest precondition of

  x := x + y {x + y = w – x}?

{ ? }

x := x + y

{x + y = w – x}

The Assignment Rule – Your Turn

- What is the weakest precondition of

  x := x + y {x + y = w – x}?

{(x + y) + y = w – (x + y)}

x := x + y

{x + y = w – x}

Precondition Strengthening

$$\frac{P \Rightarrow P' \quad \{P'\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

- Meaning: If we can show that P implies P' (P$\Rightarrow$ P') and we can show that {P'} C {Q}, then we know that {P} C {Q}
- P is *stronger* than P' means P $\Rightarrow$ P'

Precondition Strengthening

- Examples:

$$\frac{x = 3 \Rightarrow x < 7 \quad \{x < 7\}\ x := x + 3\ \{x < 10\}}{\{x = 3\}\ x := x + 3\ \{x < 10\}}$$

$$\frac{True \Rightarrow 2 = 2 \quad \{2 = 2\}\ x := 2\ \{x = 2\}}{\{True\}\ x := 2\ \{x = 2\}}$$

$$\frac{x=n \Rightarrow x+1=n+1 \quad \{x+1=n+1\}\ x:=x+1\ \{x=n+1\}}{\{x=n\}\ x:=x+1\ \{x=n+1\}}$$

Which Inferences Are Correct?

$$\frac{\{x > 0\ \&\ x < 5\}\ x := x * x\ \{x < 25\}}{\{x = 3\}\ x := x * x\ \{x < 25\}}$$

$$\frac{\{x = 3\}\ x := x * x\ \{x < 25\}}{\{x > 0\ \&\ x < 5\}\ x := x * x\ \{x < 25\}}$$

$$\frac{\{x * x < 25\}\ x := x * x\ \{x < 25\}}{\{x > 0\ \&\ x < 5\}\ x := x * x\ \{x < 25\}}$$

Which Inferences Are Correct?

$$\frac{\{x > 0\ \&\ x < 5\}\ x := x * x\ \{x < 25\}}{\{x = 3\}\ x := x * x\ \{x < 25\}}$$ ✓

~~$$\frac{\{x = 3\}\ x := x * x\ \{x < 25\}}{\{x > 0\ \&\ x < 5\}\ x := x * x\ \{x < 25\}}$$~~

$$\frac{\{x * x < 25\}\ x := x * x\ \{x < 25\}}{\{x > 0\ \&\ x < 5\}\ x := x * x\ \{x < 25\}}$$ ✓

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

- Example:

$$\frac{\begin{array}{c}\{z = z \ \&\ z = z\}\ x := z\ \{x = z \ \&\ z = z\} \\ \{x = z \ \&\ z = z\}\ y := z\ \{x = z \ \&\ y = z\}\end{array}}{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = z \ \&\ y = z\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

- Example:

$$\frac{\begin{array}{c}\{z = z \ \&\ z = z\}\ x := z\ \{x = z \ \&\ z = z\} \\ \{x = z \ \&\ z = z\}\ y := z\ \{x = z \ \&\ y = z\}\end{array}}{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = z \ \&\ y = z\}}$$

Postcondition Weakening

$$\frac{\{P\}\ C\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ C\ \{Q\}}$$

Example:

$$\frac{\begin{array}{c}\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = z \ \&\ y = z\} \\ (x = z \ \&\ y = z) \Rightarrow (x = y)\end{array}}{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = y\}}$$

Rule of Consequence

$$\frac{P \Rightarrow P' \quad \{P'\}\ C\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ C\ \{Q\}}$$

- Logically equivalent to the combination of Precondition Strengthening and Postcondition Weakening
- Uses $P \Rightarrow P$ and $Q \Rightarrow Q$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and } (\text{not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \{Q\}}$$

- Example: Want

{y=a}
if x < 0 then y:= y-x else y:= y+x
{y=a+|x|}

Suffices to show:

(1) {y=a&x<0} y:=y-x {y=a+|x|} and
(4) {y=a&not(x<0)} y:=y+x {y=a+|x|}

{y=a&x<0} y:=y-x {y=a+|x|}

$$\begin{array}{ll}(3) & (y=a \& x<0) \Rightarrow y-x=a+|x| \\ (2) & \{y-x=a+|x|\}\ y:=y-x\ \{y=a+|x|\} \\ \hline (1) & \{y=a \& x<0\}\ y:=y-x\ \{y=a+|x|\}\end{array}$$

(1) Reduces to (2) and (3) by Precondition Strengthening
(2) Follows from assignment axiom
(3) Because $x<0 \Rightarrow |x| = -x$

{y=a&not(x<0)} y:=y+x {y=a+|x|}

(6) (y=a&not(x<0))➔(y+x=a+|x|)

(5) {y+x=a+|x|} y:=y+x {y=a+|x}}

(4) {y=a&not(x<0)} y:=y+x {y=a+|x|}

(4) Reduces to (5) and (6) by Precondition Strengthening

(5) Follows from assignment axiom

(6) Because not(x<0) ➔ |x| = x

If then else

$$\frac{(1)\ \{y=a \& x<0\}\ y:=y-x\ \{y=a+|x|\} \qquad (4)\ \{y=a \& not(x<0)\}\ y:=y+x\ \{y=a+|x|\}}{\{y=a\}\ \text{if } x < 0 \text{ then } y:= y-x \text{ else } y:= y+x\ \{y=a+|x|\}}$$

By the if_then_else rule

While

- We need a rule to be able to make assertions about **while** loops.
  - Inference rule because we can only draw conclusions if we know something about the body
  - Let's start with:

$$\frac{\{\ ?\ \}\ C\ \{\ ?\ \}}{\{\ ?\ \}\ \textbf{while}\ B\ \textbf{do}\ C\ \{\ P\ \}}$$

While

- The loop may never be executed, so if we want P to hold after, it had better hold before, so let's try:

$$\frac{\{\ ?\ \}\ C\ \{\ ?\ \}}{\{\ P\ \}\ \textbf{while}\ B\ \textbf{do}\ C\ \{\ P\ \}}$$

While

- If all we know is P when we enter the **while** loop, then we all we know when we enter the body is (P and B)
- If we need to know P when we finish the **while** loop, we had better know it when we finish the loop body:

$$\frac{\{P \text{ and } B\}\ C\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{P\}}$$

While

- We can strengthen the previous rule because we also know that when the loop is finished, **not B** also holds
- Final **while** rule:

$$\frac{\{P \text{ and } B\}\ C\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{P \text{ and not } B\}}$$

While

{ P and B } C { P }
{ P } **while** B **do** C { P and not B }

- P satisfying this rule is called a *loop invariant* because it must hold before and after the each iteration of the loop

While

- **While** rule generally needs to be used together with precondition strengthening and postcondition weakening
- There is NO algorithm for computing the correct P; it requires intuition and an understanding of why the program works

Example

- Let us prove

  {x>= 0 and x = a}

  fact := 1;

  while x > 0 do (fact := fact * x; x := x –1)

  {fact = a!}

Example

- We need to find a condition P that is true both before and after the loop is executed, and such that

(P and not x > 0) ➔ (fact = a!)

Example

- First attempt:

  {a! = fact * (x!)}

- Motivation:
- What we want to compute: **a!**
- What we have computed: **fact**

  which is the sequential product of **a** down through **(x + 1)**

- What we still need to compute: **x!**

Example

By post-condition weakening suffices to show

1. {x>=0 and x = a}

   fact := 1;

   while x > 0 do (fact := fact * x; x := x –1)

   {a! = fact * (x!) and not (x > 0)}

and

2. {a! = fact * (x!) and not (x > 0) } ➔ {fact = a!}

Problem

2. {a! = fact * (x!) and not (x > 0)} ➔ {fact = a!}

- Don't know this if x < 0
- Need to know that x = 0 when loop terminates
- Need a new loop invariant
- Try adding x >= 0
- Then will have x = 0 when loop is done

Example

Second try, combine the two:

P = {a! = fact * (x!) and x >=0}

Again, suffices to show

1. {x>=0 and x = a}
   fact := 1;
   while x > 0 do (fact := fact * x; x := x –1)
   {P and not x > 0}

and

2. {P and not x > 0} ➔ {fact = a!}

Example

- For 2, we need

{a! = fact * (x!) and x >=0 and not (x > 0)} ➔
{fact = a!}

But {x >=0 and not (x > 0)} ➔ {x = 0} so

fact * (x!) = fact * (0!) = fact

Therefore

{a! = fact * (x!) and x >=0 and not (x > 0)} ➔
{fact = a!}

Example

- For 1, by the sequencing rule it suffices to show

3. {x>=0 and x = a}
   fact := 1
   {a! = fact * (x!) and x >=0 }

And

4. {a! = fact * (x!) and x >=0}
   while x > 0 do
   (fact := fact * x; x := x –1)
   {a! = fact * (x!) and x >=0 and not (x > 0)}

Example

- Suffices to show that

{a! = fact * (x!) and x >= 0}

holds before the while loop is entered and that if

{(a! = fact * (x!)) and x >= 0 and x > 0}

holds before we execute the body of the loop, then

{(a! = fact * (x!)) and x >= 0}

holds after we execute the body

Example

By the assignment rule, we have

{a! = 1 * (x!) and x >= 0}
fact := 1
{a! = fact * (x!) and x >= 0}

Therefore, to show (3), by precondition strengthening, it suffices to show

(x>= 0 and x = a) ➔
(a! = 1 * (x!) and x >= 0)

Example

(x>= 0 and x = a) ➔
(a! = 1 * (x!) and x >= 0)
holds because x = a ➔ x! = a!

Have that {a! = fact * (x!) and x >= 0}
holds at the start of the while loop

Example

To show (4):
{a! = fact * (x!) and x >=0}
while x > 0 do
(fact := fact * x; x := x –1)
{a! = fact * (x!) and x >=0 and not (x > 0)}
we need to show that
{(a! = fact * (x!)) and x >= 0}
is a loop invariant

Example

We need to show:
{(a! = fact * (x!)) and x >= 0 and x > 0}
(fact = fact * x; x := x – 1)
{(a! = fact * (x!)) and x >= 0}

We will use assignment rule, sequencing rule and precondition strengthening

Example

By the assignment rule, we have
{(a! = fact * ((x-1)!)) and x – 1 >= 0}
x := x – 1
{(a! = fact * (x!)) and x >= 0}
By the sequencing rule, it suffices to show
{(a! = fact * (x!)) and x >= 0 and x > 0}
fact = fact * x
{(a! = fact * ((x-1)!)) and x – 1 >= 0}

Example

By the assignment rule, we have that
{(a! = (fact * x) * ((x-1)!)) and x – 1 >= 0}
fact = fact * x
{(a! = fact * ((x-1)!)) and x – 1 >= 0}
By Precondition strengthening, it suffices to show that
((a! = fact * (x!)) and x >= 0 and x > 0) ➔
((a! = (fact * x) * ((x-1)!)) and x – 1 >= 0)

Example

However
fact * x * (x – 1)! = fact * x
and (x > 0) ➔ x – 1 >= 0
since x is an integer,so
{(a! = fact * (x!)) and x >= 0 and x > 0} ➔
{(a! = (fact * x) * ((x-1)!)) and x – 1 >= 0}

Example

Therefore, by precondition strengthening

{(a! = fact * (x!)) and x >= 0 and x > 0}

fact = fact * x

{(a! = fact * ((x-1)!)) and x – 1 >= 0}

This finishes the proof