Programming Languages and Compilers (CS 421)

Sasa Misailovic

4110 SC, UIUC

https://courses.engr.illinois.edu/cs421/fa2017/CS421A

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

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 kind 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 kind
- 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 <expression> then <command>
else <command> fi
| while <expression> do <command> od

- 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

———————————
{P [e/x] } x := e {P}

Example:

———————————
{ ? } x := y {x = 2}

The Assignment Rule

———————————
{P [e/x] } x := e {P}

Example:

———————————
{ _ = 2 } x := y { x = 2}

The Assignment Rule

———————————
{P [e/x] } x := e {P}

Example:

———————————
{ y = 2 } x := y { x = 2}

The Assignment Rule

———————————
{P [e/x] } x := e {P}

Examples:

———————————
{y = 2} x := y {x = 2}

———————————
{y = 2} x := 2 {y = x}

———————————
{x + 1 = n + 1} x := x + 1 {x = n + 1}

———————————
{2 = 2} x := 2 {x = 2}

The Assignment Rule – Your Turn

- What is a valid precondition of
  x := x + y {x + y = w – x}?

{ ? }
x := x + y
{x + y = w – x}

The Assignment Rule – Your Turn

- What is a valid 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 ➔ P') and we can show that {P'} C {Q}, then we know that {P} C {Q}
- P is *stronger* than P' means P ➔ P'

Precondition Strengthening

- Examples:

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

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

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

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:

$$\frac{\{z = z \ \&\ z = z\}\ x := z\ \{x = z \ \&\ z = z\} \qquad \{x = z \ \&\ z = z\}\ y := z\ \{x = z \ \&\ y = z\}}{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = z \ \&\ y = z\}}$$

Sequencing

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

- Example:

$$\frac{\{z = z \ \&\ z = z\}\ x := z\ \{x = z \ \&\ z = z\} \qquad \{x = z \ \&\ z = z\}\ y := z\ \{x = z \ \&\ y = z\}}{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = z \ \&\ y = z\}}$$

Postcondition Weakening

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

Example:

$$\frac{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \{x = z \ \&\ y = z\} \qquad \mathbf{(x = z \ \&\ y = z) \rightarrow (x = y)}}{\{z = z \ \&\ z = z\}\ x := z;\ y := z\ \mathbf{\{x = y\}}}$$

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

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

- Example: Want

{y=a}

if x < 0 then y:= y-x else y:= y+x fi

{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|}

(3) (y=a & x<0) → y-x=a+|x|

(2) {y-x=a+|x|} y:=y-x {y=a+|x|}

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

(1) Reduces to (2) and (3) by ***Precondition Strengthening***

(2) Follows from ***assignment*** axiom

(3) Because from algebra: x<0 → |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

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

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

{y=a}

if x < 0 then y:= y-x else y:= y+x

{y=a+|x|}

By the **IfThenElse 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:

{ ? } C { ? }

{ ? } **while** B **do** C **od** { 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:

{ ? } C { ? }

{ P } **while** B **do** C **od** { 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:

{ P and B} C { P }

{ P } **while** B **do** C **od** { 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:

{ P and B } C { P }

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

While

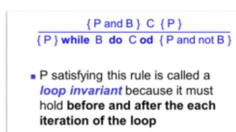
$$\frac{\{\,P \text{ and } B\,\}\ C\ \{\,P\,\}}{\{\,P\,\}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{\,P \text{ and not } B\,\}}$$

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

While

- **While** rule generally needs to be <u>used together</u> 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

Counting up to n


{ P and B } C { P }
{ P } **while** B **do** C **od** { 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**

```
x := 0;
while (x < n) {
   x := x + 1
}
```

P ≡ $x \leq n \land 0 \leq n$

```
Want to show: x >= n && n >= 0
```


Sum of numbers 1 to n

```
x := 0
y := 0

while y < n {
    y := y + 1;
    x := x + y
}
```

P ≡ $x = 1 + \ldots + y \land y \leq n \land 0 \leq n$

```
Want to show: x = 1 + ... + n
```

Fibonacci

```
x = 0; y = 1;
z = 1;

while (z < n) {
   y := x + y;
   x := y - x;
   z := z + 1
}
```

P ≡ $y = \text{fib } z \land x = \text{fib } (z-1) \land z \leq n \land 1 \leq n$

```
Want to show: y = fib n
```

List Length

```
x = lst; y = 0

while (x ≠ []) {
    x := tl x;
    y := y + 1
}
```

P ≡ $y + \text{len } x = \text{len lst}$

```
Want to show: y = len lst
```

Example (Use of Loop Invariant in Full Proof)

- Let us prove

```
{x>= 0 and x = a}

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

{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!)

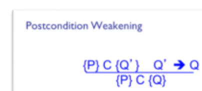
Example

- First attempt:

P = {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) od
```

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

And

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

Problem!! (Dead End)

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, let us combine the two:

P ≡ a! = fact * (x!) and x >=0

We need to show:

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

```
fact := 1;
```

{P}

```
while x > 0 do (fact := fact * x; x := x -1) od
```

{P and not x > 0}

And

2. P and not x > 0 ➔ fact = a!

Example

```
{x>= 0 and x = a} (*this was part 1 to prove*)
   fact := 1;
   while x > 0 do (fact := fact * x; x := x -1) od
{a! = fact * (x!) and x >=0 and not (x>0)}
```

- For Part 1, by 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) od
   ```
   {a! = fact * (x!) and x >=0 and not (x > 0)}

Example

- (Part 3 – Assignment) Suffices to show that
  a! = fact * (x!) and x >= 0
  holds before the while loop is entered

- (Part 4 – While Loop) 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 (part 4)

Example

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

Precondition Strengthening

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

(Part 3) 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)

It holds because x = a ➔ x! = a! .

- So, we have that a! = fact * (x!) and x >= 0 **holds at the start of the while loop!**

Example

To prove (Part 4):

{a! = fact * (x!) and x >=0}
```
while x > 0 do
  (fact := fact * x; x := x -1)
od
```
{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**

- We will use **assignment** rule, **sequencing** rule and **precondition strengthening** rule

Example

Sequencing

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

- We look into the loop body:
  - `(fact := fact * x; x := x – 1)`
- By the sequencing rule, we need to show 2 things:
  - By the ***assignment rule***, show
    {(a! = fact * (x!)) and x >= 0 and x > 0}
    fact = fact * x
    {Q}
  - By the ***assignment rule***, show
    {Q}
    x := x – 1
    {(a! = fact * (**x**!)) and **x** >= 0}

Example

The Assignment Rule

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

- We look into the loop body:
  - `(fact := fact * x; x := x – 1)`
- By the sequencing rule, we need to show 2 things:
  - By the ***assignment rule***, show
    {(a! = fact * (x!)) and x >= 0 and x > 0}
    fact = fact * x
    {Q}
  - From the ***assignment rule***, we know:
    {(a! = fact * (**(x-1)**!)) and **x – 1** >= 0}
    x := x – 1
    {(a! = fact * (**x**!)) and **x** >= 0}

Example

- We look into the loop body:
  - `(fact := fact * x; x := x - 1)`
- By the sequencing rule, we need to show 2 things:
  - By the ***assignment rule***, show

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

  - From the ***assignment rule***, we know:

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

Example

Precondition Strengthening

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

- 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**)

From algebra we know that 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

Second try, let us combine the two:
P ≡ a! = fact * (x!) and x >=0
We need to show:

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

```
fact := 1;
```

{P}

```
while x > 0 do (fact := fact * x; x := x -1) od
```

{P and not x > 0}

And

2. P and not x > 0 ➔ fact = a!

Example

- For Part 2, we need

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

Since we know (x >=0 and not (x > 0)) ➔ (x = 0) so
fact * (x!) = fact * (0!)
And since from algebra we know that 0! = 1,
fact * (0)! = fact * 1 = fact

- Therefore, we can prove:

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

Example

- We proved that (**a! = fact * (x!)**) and x >= 0 is the loop invariant
- We proved the sequence rule for the assignment and wile statements
- We applied postcondition weakening to prove the final predicate

This finishes the proof!

```
{x >= 0 and x = a}
  fact := 1;
  while x > 0 do (fact := fact * x; x := x - 1) od
{fact = a!}
```