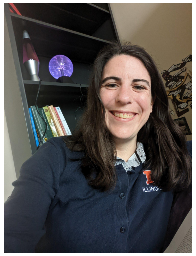
Programming Languages and Compilers (CS 421)

Talia Ringer (they/them)

4218 SC, UIUC



https://courses.grainger.illinois.edu/cs421/fa2023/

Based heavily on slides by Elsa Gunter, which were based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Logistics (Piazza Post)

Objectives for Today

- Last class, we started the final part of semantics, which is the last thing we are covering in this class!
- We began covering **Axiomatic Semantics** specifically, via Floyd-Hoare logic
- Today, we will continue with the **while** rule
- When we are done, we will have some time for **review questions** for the **final**

Objectives for Today

- Last class, we started the final part of semantics, which is the last thing we are covering in this class!
- We began covering **Axiomatic Semantics** specifically, via Floyd-Hoare logic
- Today, we will continue with the **while** rule
- When we are done, we will have some time for **review questions** for the **final**

Questions before we start?

Looping

While (WIP)

$\{P\}\ C\ \{Q\}$

- We need a rule to be able to make assertions about while loops.
  - Inference rule (not axiom) because we can only draw conclusions about the loop if we know something about its **body** **C**
  - Let's start with:

$$\frac{\{\ ??\ \}\ C\ \{\ ??\ \}}{\{\ ??\ \}\ \text{while}\ \ B\ \ \text{do}\ \ C\ \text{od}\ \{\ ??\ \}}\ \textbf{WHILE}$$

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - The loop body C may never execute (if the guard **B** is false), so if we want some **P** to hold **after** the loop, it must hold **before**.
  - Let's start with:

$$\frac{\{\ \mathbf{??}\ \}\ C\ \{\ \mathbf{??}\ \}}{\{\ \mathbf{??}\ \}\ \text{while}\ \ \mathbf{B}\ \ \text{do}\ \ C\ \text{od}\ \{\ \mathbf{??}\ \}}\ \textbf{WHILE}$$

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - The loop body C may never execute (if the guard B is false), so if we want some **P** to hold **after** the loop, it must hold **before**.
  - So let's try:

$$\frac{\{\ \mathbf{??}\ \}\ C\ \{\ \mathbf{??}\ \}}{\{\ \mathbf{P}\ \}\ \text{while}\ \ B\ \ \text{do}\ \ C\ \text{od}\ \{\ \mathbf{P}\ \}}\ \textbf{WHILE}$$

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - If all we know is **P** when we **enter** the while loop, then all we know when we enter the body C is (**P** and **B**)
  - So let's try:

$$\frac{\{\ \mathbf{??}\ \}\ C\ \{\ \mathbf{??}\ \}}{\{\ \mathbf{P}\ \}\ \text{while}\ \ \mathbf{B}\ \ \text{do}\ \ C\ \text{od}\ \{\ \mathbf{P}\ \}}\ \textbf{WHILE}$$

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - If all we know is **P** when we **enter** the while loop, then all we know when we enter the body C is (**P** and **B**)
  - So let's try:

$$\frac{\{\ \mathbf{P}\ \text{and}\ \mathbf{B}\ \}\ C\ \{\ \mathbf{??}\ \}}{\{\ \mathbf{P}\ \}\ \text{while}\ \ \mathbf{B}\ \ \text{do}\ \ C\ \text{od}\ \{\ \mathbf{P}\ \}}\ \textbf{WHILE}$$

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - If we need to know **P** when we **finish** the while loop, we had better know it when we **finish** the loop body C
  - So let's try:

$$\frac{\{\ \mathbf{P} \text{ and } \mathbf{B}\ \}\ C\ \{\ \mathbf{??}\ \}}{\{\ \mathbf{P}\ \}\ \text{while}\ \ \mathbf{B}\ \ \text{do}\ \ C\ \text{od}\ \{\ \mathbf{P}\ \}}\ \textbf{WHILE}$$

While (WIP)

$\{P\}\ C\ \{Q\}$

- We need a rule to be able to make assertions about while loops.
  - If we need to know **P** when we **finish** the while loop, we had better know it when we **finish** the loop body C
  - So let's try:

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

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - Finally, we can strengthen this rule because we also know that when the whole loop is **finished**, not **B** also holds
  - So let's try:

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

While (WIP)

{P} C {Q}

- We need a rule to be able to make assertions about while loops.
  - Finally, we can strengthen this rule because we also know that when the whole loop is **finished**, not **B** also holds
  - So let's try:

$$\frac{\{\ \mathbf{P}\ \text{and}\ \mathbf{B}\ \}\ \mathrm{C}\ \{\ \mathbf{P}\ \}}{\{\ \mathbf{P}\ \}\ \text{while}\ \ \mathbf{B}\ \ \text{do}\ \ \mathrm{C}\ \text{od}\ \{\ \mathbf{P}\ \text{and not}\ \mathbf{B}\ \}}\ \textbf{WHILE}$$

While

{P} C {Q}

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

Looping

While

{P} C {Q}

P satisfying this rule is called a **loop invariant** because it must hold before and after the each iteration of the loop. (Finding these invariants is a major part of the proof process!)

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

While

{P} C {Q}

P satisfying this rule is called a **loop invariant** because it must hold before and after the each iteration of the loop. (Finding these invariants is a major part of the proof process!)

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

While

{P} C {Q}

P satisfying this rule is called a **loop invariant** because it must hold before and after the each iteration of the loop. (Finding these invariants is a major part of the proof process!)

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

So of course it's **undecidable** in general to find **P** for an arbitrary program and specification …

Looping

While IRL

- We can still find loop invariants for specific programs, but doing this often involves **program-specific reasoning** and **intuition**
- Typically **one of the hardest parts** of writing proofs about programs this way
- In addition, the while rule typically needs to be used together with precondition **strengthening** and postcondition **weakening**

While IRL

- We can still find loop invariants for specific programs, but doing this often involves **program-specific reasoning** and **intuition**
- Typically **one of the hardest parts** of writing proofs about programs this way
- In addition, the while rule typically needs to be used together with precondition **strengthening** and postcondition **weakening**

Questions so far?

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{fact = a!}

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{fact = a!}

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

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

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

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

First attempt: **{ a! = fact * (x!) }**

Motivation: Want to compute a!, have computed fact, which is the sequential product of a down through (x + 1). What remains is to compute x!

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

First attempt: **{ a! = fact * (x!) }**

Motivation: Want to compute a!, have computed fact, which is the sequential product of a down through (x + 1). What remains is to compute x!

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

Need: (**a! = fact * (x!)** and not x > 0) → **(fact = a!)**

Motivation: Weakening

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

Need: (**a! = fact * (x!)** and not x > 0) → **(fact = a!)**

Motivation: Weakening

Problem 1: What if x < 0?

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

Need: (**a! = fact * (x!)** and not x > 0) → **(fact = a!)**

Motivation: Weakening

Problem 1: What if **x < 0**? Impossible, but our loop invariant doesn't tell us that, so we can't show the implication.

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

Need: (**a! = fact * (x!)** and not x > 0) → **(fact = a!)**

Motivation: Weakening

Problem 2: We need that x = 0 when loop is done.

Example

We want to show that:

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

fact := 1;

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

{**fact = a!**}

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

Motivation: Same as before, but add **x >= 0**

Example

We want to show that:

{x >= 0 and x = a}

fact := 1;

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

{**fact = a!**}

Need:

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

Motivation: Weakening

Example

Pure logic fragment

??

(a! = fact * (x!) and x >=0 and not x > 0) $\rightarrow$
(fact = a!)

Looping

Example

Pure logic fragment

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

Looping

Example

Pure logic fragment

(**x >= 0** and **not (x > 0)**) → **x = 0**

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

rewrite to (a! = fact) → (fact = !a)

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

Looping

Example

Pure logic fragment

(**x >= 0** and **not (x > 0)**) → **x = 0**

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

rewrite to (a! = fact) → (fact = !a)

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

Looping

Example

Pure logic fragment

(**x >= 0** and **not (x > 0)**) → **x = 0**

---

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

---

rewrite to (a! = fact) → (fact = !a)

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

Looping

Example

By weakening, remains to show:

??

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

Looping

Example

Sequence rule applies

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

{**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)}

SEQ

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

Example

??

Sequence rule applies

??

{x >= 0 and x = a}

fact := 1

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

{**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)}

SEQ

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

Example

Move to new slide

??

{**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)}

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

SEQ

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

Example

Move to this slide

??

{x >= 0 and x = a}

fact := 1

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

Looping

Example

Assignment rule gets us this from a different precondition

ASSIGN

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

??

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

Example

So we need to get from this precondition to the one we want

ASSIGN

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

??

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

Example

We can do this by strengthening

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

ASSIGN

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

STR

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

Example

And this in the pure logic fragment

$$\frac{\dfrac{x = a \rightarrow x! = a!}{(x >= 0 \text{ and } x = a) \rightarrow (a! = 1 * (x!) \text{ and } x >= 0)} \quad \dfrac{}{\{a! = 1 * (x!) \text{ and } x >= 0\}\ fact := 1\ \{a! = fact * (x!) \text{ and } x >= 0\}}\ \text{ASSIGN}}{\{x >= 0 \text{ and } x = a\}\ fact := 1\ \{a! = fact * (x!) \text{ and } x >= 0\}}\ \text{STR}$$

Example

x = a → x! = a!

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

ASSIGN

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

STR

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

Example

This means our loop invariant is <u>strong enough</u>. But is it actually a loop invariant?

??

{**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)}

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

SEQ

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

Example

Move to new slide

{**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)}

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

SEQ

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

Example

Move to this slide

{**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)}

Looping

Example

This is a while loop

$$\frac{\{a! = fact * (x!) \text{ and } x >= 0 \text{ and } x > 0\}\ (fact = fact * x;\ x := x - 1)\ \{a! = fact * (x!) \text{ and } x >= 0\}}{\{a! = fact * (x!) \text{ and } x >=0\}\ \text{while } x > 0 \text{ do } (fact := fact * x;\ x := x - 1) \text{ od}\ \{a! = fact * (x!) \text{ and } x >= 0 \text{ and not } (x > 0)\}}\ \text{WHILE}$$

Looping

Example

You show later: assignment, sequencing, strengthening …

??

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

WHILE

{**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)}

Looping

Example

By this and the first weakening we did

{**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)}

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

SEQ

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

Example

We get that:

{x >= 0 and x = a}

fact := 1;

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

{fact = a!}

Questions?

Final Review: Ask Away

The End

- Great job!!!
- **WA11 due Tomorrow**
- **Final** is December 12th, 8:00 AM - 11:00 AM
- All deadlines can be found on **course website**
- Use **office hours** and **class forums** for help