Loops

Programs section, **Lecture 15**

Pavel Panchekha

CS 6110, U of Utah

25 February 2020

Hoare Logic

Behavior of a statement given by a **triple**:

$$\{P\}\ \mathbf{s}\ \{Q\}$$

If **P** true of a state, and **s** executed, then **Q** true after

P and **Q** are **logical properties** of the state and effects

- Need not (but can) exactly represent the state

Conditionals

In what cases is **{P} if (e) { s } else { t } {Q}** true?

- If **e**, same as **{P} s {Q}**
- If not **e**, same as **{P} t {Q}**

Triples already include the “if precondition” idea:

$$\{P \land e\}\ s\ \{Q\} \lor \{P \land \neg e\}\ t\ \{Q\}$$

Weakest Precondition

$$\texttt{\{P\} x = e \{Q\}} \quad \leftrightarrow (P \rightarrow Q[x := e])$$

$$WP[\texttt{x = e}](Q) = Q[x := e]$$

Common pattern: $P \rightarrow \underline{\text{something}(Q)}$

Weakest precondition of Q

$$\texttt{\{P\} s \{Q\}} \quad \leftrightarrow (P \rightarrow \overline{WP[s](Q)})$$

Class Progress

Logical reasoning	Program logics	Static analysis

Expressions	Statements
Loops	Procedures

Loops

What are the **weakest preconditions of loops**?

Imagining loops as infinitely-long sequences

Loop invariants for verification

What does not change over a loop iteration

Proving termination using decreasing measures

It can’t go below zero!

Verifying Loops

Loop invariants

Some loop examples

Which of these Hoare triples **are true**?

$\{\ n = m \wedge i = 0\ \}$

```
while (n > 0) { i++; n--; }
```

$\{\ i = m\ \}$

$\{\ l \geq 0 \wedge n = 1\ \}$

```
k = 0
while k < l:
  k += 1
  n *= 2
```

$\{\ n = 2^l\ \}$

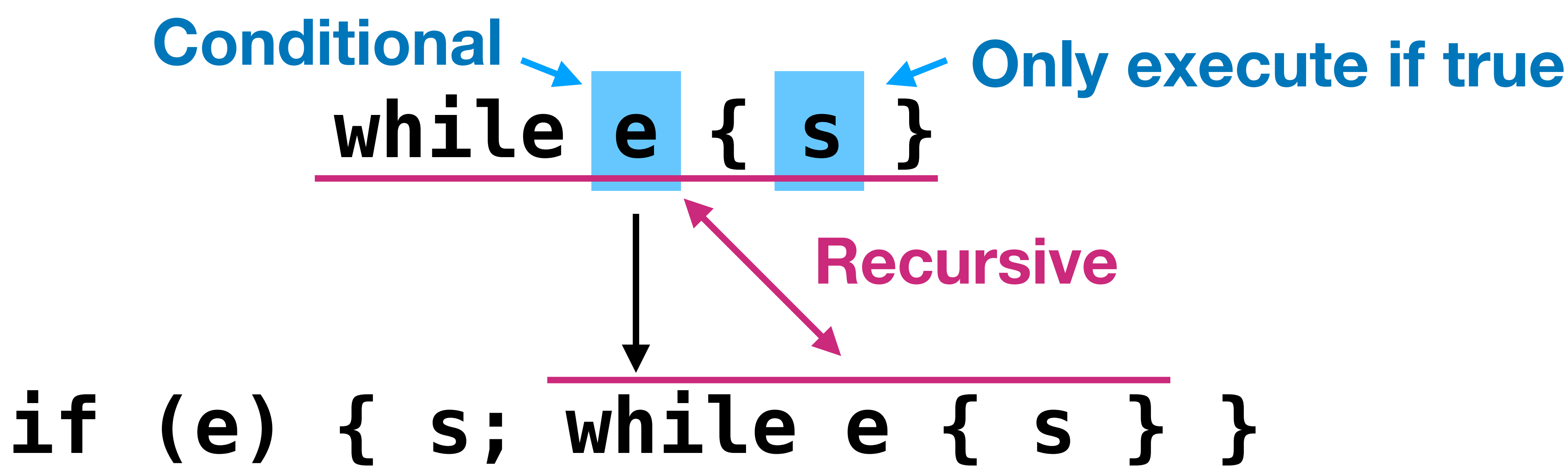
$\{\ l < r\ \}$

```
while rand():
  t = l
  l = r
  r = t
```

$\{\ l < r\ \}$

What loops do

How do we describe the **behavior** of loops?



We can think of while loops as **infinite statements**

```
if (e) { s; if (e) { s; if (e) { … } } }
```

Hoare Triples

```
while e { s } = if (e) { s; while e { s } }
```

Use loop unrolling to **convert Hoare triples** to logic:

```
{P} while e { s } {Q}
```

```
{P} if e { s; while e { s } } {Q}
```

$(P \wedge \neg e \rightarrow Q) \wedge$ `{P ∧ e} s; while e { s } {Q}`

$(P \wedge \neg e \rightarrow Q) \wedge$

`{P ∧ e} s {P’}` $\wedge$

`{ P’} while e { s } {Q}`

Hoare Triples

$(P \wedge \neg e \rightarrow Q) \wedge$

`{P ∧ e} s {P'}` $\wedge$

`{ P'} while e { s } {Q}`

↓

$(P \wedge \neg e \rightarrow Q) \wedge$

`{P ∧ e} s {P'}` $\wedge$

$(P' \wedge \neg e \rightarrow Q) \wedge$

`{P' ∧ e} s {P''}` $\wedge$

$(P'' \wedge \neg e \rightarrow Q) \wedge$

...

Invent infinitely-many conditions $P^{(n)}$

Loop Invariants

$(P \land \neg e \rightarrow Q) \land$

{P ∧ e} s {P'} ∧

$(P' \land \neg e \rightarrow Q) \land$

{P' ∧ e} s {P''} ∧

$(P'' \land \neg e \rightarrow Q) \land$

...

Invent infinitely-many conditions $P^{(n)}$

Simplest case: $P^{(n)}$ is some **fixed condition** I

$P \Rightarrow I \land$

$(I \land \neg e \rightarrow Q) \land$

{I ∧ e} s {I}

Loop Invariants

New syntax for writing loop invariants:

```
{P} while {I} e { s } {Q}
```

$P \rightarrow I \wedge$

$(I \wedge \neg e \rightarrow Q) \wedge$

```
{I ∧ e} s {I}
```

Weakest precondition computed from invariant:

$$WP(Q) = I \wedge (I \wedge \neg e \rightarrow Q) \wedge (I \wedge e \rightarrow WP[s](I))$$

Demo

Count to **n**

Search an array

Binary search

Exercises

Write down **loop invariants** for the following loops:

$\{ \; n = m \wedge n \geq 0 \wedge i = 0 \; \}$

```
while (n > 0) { i++; n--; }
```

$\{ \; i = m \; \}$

$\{ \; l \geq 0 \wedge n = 1 \; \}$

```
k = 0
while k < l:
  k += 1
  n *= 2
```

$\{ \; n = 2^l \; \}$

$\{ \; l < r \; \}$

```
while rand():
  t = l
  l = r
  r = t
```

$\{ \; l \neq r \; \}$

Invariant properties

If I_1 and I_2 are invariants, **so is** $I_1 \wedge I_2$; the reverse if false:

$\{\ 0 < n \wedge 0 < m\ \}$

```
n += m
m += n
```

$\{\ 0 < n \wedge 0 < m\ \}$

You may need an I that is stronger than P

$\{\ n = m \wedge n \geq 0 \wedge i = 0\ \}$

```
while (n > 0) { i++; n--; }
```

$\{\ i = m\ \}$

Course Updates

Milestone Presentations and Assignment 4

Assignment 4

Use Dafny to **verify several sorting algorithms**:

- Bubble sort
- Insertion sort
- Merge sort

Check the textbook

Proofs due **March 5**; do not change the specification!

Dafny available online but **much faster** if installed locally

Milestone I

Milestone presentations on **March 3**; time limit like proposal

- What did you **get done**?
- What did you **learn in the process**?
- Where did you **deviate from the plan**?
- What is your **plan for Milestone II**?

Switch who presents if proposal was solo

Termination

Go away, halting problem

Must We Terminate?

Behavior of a statement given by a **triple**:

$$\{P\}\ \mathbf{s}\ \{Q\}$$

If **P** true of a state, and **s** executed, then **Q** true after

If **s** doesn't terminate, do we treat **Q** as true or false?

Doesn't matter: **loops ought to terminate**

Proving Termination

Which of the following loops **terminate**?

```
k = 0
while k < l:
  k += 1
  n *= 2
```

$\{\ l \leq r\ \}$

```
while r - l > 1:
  if f():
    r = (r + l) // 2
  else:
    l = (r + l) // 2
```

$\{\ m \geq 0\ \}$

```
while m > 1:
  if m % 2 == 0:
    m = m / 2
  else:
    m = 2*m + 1
```

Termination

What does it mean that a loop terminates?

Loop must have a **finite number of iterations**

Idea: **compute number of iterations** for the loop

Problem: **usually hard**, requires inductive reasoning

Better idea: **bound** number of iterations for the loop

Usually easy by hand, often **possible automatically**

Bounding Iterations

Bound must **decrease** every iteration

Function that computes bound called the "measure" M

```
while e:
  { I ∧ e }
  s
  { I }
```

$\{ I \wedge e \}$ / $\{ I \}$

```
while e:
  { M() = n }
  s
  { M() < n }
```

$\{ M() = n \}$ / $\{ M() < n \}$

Can assume I and e to prove measure decreases

Higher Ordinals

Sometimes convenient to have **non-integer measures**

Lexicographic order, tree depth very common

```
while m > 0:
  if n == 0:
    m--
    n = f(m)
  else:
    n--
```

(m, n) always decreases

Bound depends on “f”

Key property: cannot decrease **infinitely many times**

Measures Example

k decreases

```
k = 0
while k < l:
  k += 1
  n *= 2
```

r - l decreases

$\{\ l \leq r\ \}$

```
while r - l > 1:
  if f():
    r = (r + l) // 2
  else:
    l = (r + l) // 2
```

(m, n) decreases

$\{\ n \geq 0\ \}$

```
while m > 0:
  if n == 0:
    m -= 1
    n = m
  else:
    n -= 1
```

Exercises

Find **loop measures** for the following loops:

```
{ m = 0 }
while n > 0:
  n = n / 2
  m++
```

```
{ stack : List<T> }
while stack:
  print(f(stack.pop()))
```

Exercises

Find **loop measures** for the following loops:

```
{ binary_search_tree(node) }
while True:
  if node.value == x:
    return node
  elif node.value < x:
    node = node.right
  else:
    node = node.left
```

Loop Verification

Verifying a loop requires **two inputs:**

Invariant I | Measure M

With two constraints:

$$\{I \land e\}\ s\ \{I\}$$

$$\{I \land e \land M = m\}\ s\ \{M < m\}$$

Automatically discovering I and M is **impossible**

It would solve … were possible!

Though heuristics work quite well

Next class:

Procedures

To do:

- ☐ Course feedback
- ☐ Milestone I presentation
- ☐ Assignment 4

Loops

What are the **weakest preconditions of loops**?

Imagining loops as infinitely-long sequences

Loop invariants for verification

What does not change over a loop iteration

Proving termination using decreasing measures

It can’t go below zero!

PROCEDURES

REUSING

P AND Q

Recursion Measures, Again

SEPARATION

FRAME

PROBLEM

Next class:

Procedures

To do:

- ☐ Course feedback
- ☐ Milestone I presentation
- ☐ Assignment 4