CS 421 Lecture 24: Examples of Hoare logic

- Lecture outline
  - sum of *n*
  - Fibonacci
  - list append
  - list reverse
  - termination

Review: nference rules for Hoare logic

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

$$\frac{P \Rightarrow P' \quad P'\{S\}Q' \quad Q' \Rightarrow Q}{P\{S\}Q} \qquad \frac{P \,\&\, b\{S\}P}{P\{\text{while}\,(b)\,S\}P \,\&\, \neg b}$$

$$\frac{P\{S_1\}Q \quad Q\{S_2\}R}{P\{S_1; S_2\}R} \qquad \frac{P \,\&\, b\{S_1\}Q \quad P \,\&\, \neg b\{S_2\}Q}{P\{\text{if}\ (b)\ \text{then}\ S_1\ \text{else}\ S_2\}Q}$$

A Note about the assignment axiom

- Assignment rule fails when *aliasing* is possible.
- Aliasing is when two different expressions refer to the same location.
  - a[j] = 3 { a[i] := 4 } a[i] = 4 & a[j] = 3
  - s.x = 4 { r.x := 3 } r.x = 3 & s.x = 4
  - y = 3 { x := 4 } x = 4 & y = 3
- Assignment axiom still valid as long as right-hand side of assignment is not aliasable.

Sum of n

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

Sum of n

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

After loop:

x = 1 + ... + y & y = n

Sum of n

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

Loop invariant:

$P \equiv x = 1 + \ldots + y \wedge y \leq n$

After loop:

$x = 1 + \ldots + y \;\&\; y = n$

Proof

$$\frac{}{x = 0 \wedge y = 0 \quad \{While \ldots\} \quad x = 1 + \ldots + n}$$

Proof

$x = 0 \wedge y = 0 \rightarrow x = 1 + \ldots + y \wedge y \leq n$

$x = 1 + \ldots + y \wedge y \leq n \wedge \neg(y < n) \rightarrow x = 1 + \ldots + n$

$x = 1 + \ldots + y \wedge y \leq n \wedge y < n \rightarrow$?

$x + y + 1 = 1 + \ldots + y + 1 \wedge y + 1 \leq n$

$\{y := y + 1\} \quad x + y = 1 + \ldots + y \wedge y \leq n$

$\{x := x + y\} \quad x = 1 + \ldots + y \wedge y \leq n$

$? \quad \{y := y + 1; x := x + y\} \quad x = 1 + \ldots + y \wedge y \leq n$

$x = 1 + \ldots + y \wedge y \leq n \wedge y < n \quad \{y := y + 1; x := x + y\} \quad x = 1 + \ldots + y \wedge y \leq n$

$x = 1 + \ldots + y \wedge y \leq n \quad \{\text{While } y < n \ldots\} \quad x = 1 + \ldots + y \wedge y \leq n \wedge \neg(y < n)$

$x = 0 \wedge y = 0 \quad \{\text{While} \ldots\} \quad x = 1 + \ldots + n$

Fibonacci

```
x = 0 & y = 1 & z = 1 & 1 ≤ n
{
  while (z < n)
    y := x + y;
    x := y – x;
    z := z + 1
}
y = fib n
```

Fibonacci

```
x = 0 & y = 1 & z = 1 & 1 ≤ n
{
  while (z < n)
    y := x + y;
    x := y – x;
    z := z + 1
}
y = fib n
```

Invariant:

P ≡ y = fib z ∧ x = fib (z-1)

∧ z ≤ n

Proof

$$x = 0 \wedge y = 1 \wedge z = 0 \wedge 1 \leq n \quad \{While \ldots\} \quad y = fib\ n$$

Proof

$x = 0 \wedge y = 1 \wedge z = 0 \wedge 1 \leq n \rightarrow y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n$

$y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n \wedge \neg(z < n) \rightarrow y = fib\ n$

$y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n \wedge z < n \rightarrow$?

$x+y = fib\ (z+1) \wedge x+y-x = fib\ (z+1-1) \wedge z + 1 \leq n$

$\{y := x + y\}\ \ y = fib\ (z+1) \wedge y-x = fib\ (z+1-1) \wedge z + 1 \leq n$

$\{x := y - x\}\ \ y = fib\ (z+1) \wedge x = fib\ (z+1-1) \wedge z + 1 \leq n$

$\{z := z + 1\}\ \ y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n$

$? \qquad \{y := x + y;\ x := y - x;\ z := z + 1\}\ \ y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n$

$y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n \wedge z < n \quad \{y := x + y;\ x := y - x;\ z := z + 1\} \quad y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n$

$y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n \quad \{\text{While } z < n \ldots\} \quad y = fib\ z \wedge x = fib\ (z-1) \wedge z \leq n \wedge \neg(z < n)$

$x = 0 \wedge y = 1 \wedge z = 0 \wedge 1 \leq n \quad \{\text{While } \ldots\} \quad y = fib\ n$

List length

```
x = lst & y = 0
{
  while (x ≠ [])
    x := tl x;
    y := y + 1
}
y = len lst
```

List length

```
x = lst & y = 0
{
  while (x ≠ [])
    x := tl x;
    y := y + 1
}
y = len lst
```

Invariant:

P ≡ len lst = y + len x

Proof

$$\frac{}{x = lst \wedge y = 0 \quad \{While \ldots\} \quad y = len\ lst}$$

Proof

$x = lst \wedge y = 0 \Rightarrow len\ lst = y + len\ x$

$len\ lst = y + len\ x \wedge \neg(x \neq []) \Rightarrow y = len\ lst$

$len\ lst = y + len\ x \wedge x \neq [] \Rightarrow$?

$len\ lst = y + 1 + len(tl\ x)$

$\{x := tl\ x\}$ $len\ lst = y + 1 + len\ x$

$\{y := y + 1\}$ $len\ lst = y + len\ x$

? $\{x := tl\ x;\ y := y + 1\}$ $len\ lst = y + len\ x$

$len\ lst = y + len\ x \wedge x \neq []$ $\{x := tl\ x;\ y := y + 1\}$ $len\ lst = y + len\ x$

$len\ lst = y + len\ x$ $\{While\ x \neq []\ ...\}$ $len\ lst = y + len\ x \wedge \neg(x \neq [])$

$x = lst \wedge y = 0$ $\{While\ ...\}$ $y = len\ lst$

List reverse

```
x = lst & y = []
{
  while (x ≠ [])
    y := hd x :: y;
    x := tl x
}
y = rev lst
```

List reverse

```
x = lst & y = []
{
  while (x ≠ [])
    y := hd x :: y;
    x := tl x
}
y = rev lst
```

Invariant:

P ≡ lst = rev y @ x

Proof

$$\frac{}{x = lst \wedge y = [] \quad \{While \ldots\} \quad y = rev\ lst}$$

Proof

$x = lst \wedge y = [] \Rightarrow lst = rev\ y\ @\ x$

$lst = rev\ y\ @\ x \wedge \neg(x \neq []) \Rightarrow y = rev\ lst$

$lst = rev\ y\ @\ x \wedge x \neq [] \Rightarrow$?

$lst = rev\ (hd\ x\ @\ y)\ @\ (tl\ x)$

$\{y := hd\ x\ @\ y\}$ $lst = rev\ y\ @\ (tl\ x)$

$\{x := tl\ x\}$ $lst = rev\ y\ @\ x$

? $\{y := hd\ x\ @\ y;\ x := tl\ x\}$ $lst = rev\ y\ @\ x$

$lst = rev\ y\ @\ x \wedge x \neq []$ $\{y := hd\ x\ @\ y;\ x := tl\ x\}$ $lst = rev\ y\ @\ x$

$lst = rev\ y\ @\ x$ $\{While\ x \neq []\ ...\}$ $lst = rev\ y\ @\ x \wedge \neg(x \neq [])$

$x = lst \wedge y = []$ $\{While\ ...\}$ $y = rev\ lst$

Proving termination

- Weird property of the Hoare proof system: it is possible to "prove" non-terminating programs.
- E.g., in "sum" program, change termination condition to "y != n". Now suppose n is negative.
- Judgments in Hoare logic are assertions about *partial correctness*:
  - P {A} Q means "if the state satisfies P, then after executing A, *if A terminates*, the state will satisfy Q."
  - If A doesn't terminate, the judgment is vacuously true.

Proving termination

- *Total correctness* means A will satisfy its specification (*i.e.*, its partial correctness formula) *and* will definitely terminate.

- Total correctness is usually proven in two separate steps:
  1. Prove partial correctness
  2. Prove termination

Proving termination of loops

- Obviously, the only place where non-termination is possible is in loops.

- To prove termination of a loop:
  - Define a function φ : program states → non-negative integers.
  - Prove: for every iteration of the loop, φ(current state) < φ(the previous state).
  - As long as φ is correctly defined as a function whose values are non-negative integers, then the loop cannot go on forever.

Termination proof examples

- Sum of n
  - $\varphi(x, y, n) = n - y$
- Fibonacci
  - $\varphi(x, y, z, n) = n - z$
- List append
  - $\varphi(x, y, len) = len\ x$
- List reverse
  - $\varphi(x, y, len) = len\ x$