Lecture 25: Examples of Hoare logic

- sum of n
- fibonacci
- list append
- list reverse
- termination

A note about the assignment axiom

Assignment rule fails if aliasing is possible.
Aliasing is when two different expressions refer to the same location.

if i≠j a a[j]=3 {a[i] := 4} a[i]=4 & a[j]=3

// False if: i = j

s.x=~~3~~4{r.x := 3} r.x=~~4~~3 & s.x=~~3~~4

// False if: r, s point to same object

void f(int& x, int& y)
x=4

y=3 {x := 4} x=4 & y=3

// False if: x, y are C++ ref parameters and point to same location

Assignment axiom still valid as long as right-hand side of assignment is not aliasable. Call: int i; f(i,i)

Sum of n

$x = 0 \;\&\; y = 0$

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

$x = 1 + \ldots + n$

Loop invariant:

Inv: $x = 1 + \ldots + y \;\&\; y \leq n$

After loop: $\&\; y \leq n$

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

$\varphi(x, y, n) = n - y$

Inv: $x = 1 + \ldots + y \;\&\; y \le n$

Asgn

Asgn
$$\text{Inv} \;\&\; y < n \;\; \{y := y+1\}^{x = 1+\ldots+(y-1)}_{\&\; y-1 \le n \;\&\; y \le n} \;\; \{x := x+y\}^{x-y = 1+\ldots+(y-1)}_{\&\; y \le n}$$

$$\text{Inv} \;\&\; y < n \;\; \{y := y+1 \; ; \; x := x+y\}^{x-y = 1+\ldots+(y-1)}_{\&\; y \le n} \Rightarrow \text{Inv}$$

Cons

$$\text{Inv} \;\&\; y < n \;\; \{y := y+1; \; x := x+y\} \; \text{Inv}$$

$x = 0 \;\&\; y = 0 \Rightarrow \text{Inv}$ ✓

$\text{Inv} \;\{\text{while}(\ldots)\}\; \text{Inv} \;\&\; y \not< n$ ✓

$\text{Inv} \;\&\; y \not< n \Rightarrow x = 1 + \ldots + n$

Cons

$$x = 0 \;\&\; y = 0 \;\; \{\text{while}(\;)\ldots\} \;\; x = 1 + \ldots + n$$

the rule of consequence Inv: $x = 1 + \ldots + y \;\&\; y \le n$

to prove $x+y+1 = 1 + \ldots + (y+1) \;\&\; y+1 \le n$

$\Rightarrow$ Inv $\&\; y < n$

$$\frac{\dfrac{}{x+y+1 = 1+\ldots+(y+1) \;\&\; y+1 \le n \;\{y := y+1\}\; \text{Inv}}\,\text{Asgn} \qquad \dfrac{}{x+y = 1+\ldots+y \;\&\; y \le n \;\{x := x+y\}\; \text{Inv}}\,\text{Asgn}}{\text{Inv} \;\&\; y < n \;\{y := y+1;\; x := x+y\}\; \text{Inv}}$$

$$\frac{x=0 \;\&\; y=0 \Rightarrow \text{Inv} \qquad \text{Inv} \;\{\text{while}(\ldots)\}\; \text{Inv} \;\&\; y \not< n \qquad \text{Inv} \;\&\; y \not< n \Rightarrow x = 1+\ldots+n}{\{x=0 \;\&\; y=0\}\;\{\text{while}(\;)\ldots\}\;\{x = 1+\ldots+n\}}\,\text{Cons}$$

Fibonacci

$x = 0 \;\&\; y = 1 \;\&\; z = 1 \;\&\; 1 \leq n$

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

$y = fib\ n$

Inv: $y = fib\ z \;\&\; x = fib\ (z-1) \;\&\; z \leq n$

$\varphi(x, y, z, n) = n - z$

Inv: $y = fib\ z \ \&\ x = fib\ (z-1)$
$\&\ z \le n$

$x+y = fib\ (z+1)$ $\&\ z+1 \le n$ $\{y := x+y\}$ $y = fib\ (z+1)$ $\&\ x = fib\ z$ $\&\ z+1 \le n$

$y = fib\ (z+1)$ $\&\ y - x = fib\ z$ $\&\ z+1 \le n$ $\{x := y-x\}$ $y = fib\ (z+1)$ $\&\ x = fib\ z$ $\&\ z+1 \le n$ $\{z := z+1\}$ Inv

---

$Inv\ \&\ z < n \{y := x+y; \dots\}$ Inv

$x = 0 \ \&\ y = 1 \Rightarrow$ Inv

$\dots$

Inv $\{$ while $\dots \}$ Inv $\&\ z \neq n$

Inv $\&\ z \neq n \Rightarrow y = fib\ n$

Cons.

$x = 0 \ \&\ y = 1$ $\&\ z = 1 \ \&\ 1 \le n$ $\{$ while $\dots \}$ $y = fib\ n$

List length

$x = lst \;\&\; y = 0$

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

$y = len\ lst$

Inv : $y + len\ x = len\ lst$

$\varphi(x, y, lst) = len\ x$

List reverse

$(q(x, y, lst) = len\ x)$

x = lst & y = []

{

while (x ≠ []) {

y := hd x :: y;

x := tl x;

}

}

y = rev lst

Inv: $lst = rev\ y \mathbin{@} x$

$lst = [a_1, a_2, \ldots, a_n]$

$y = [a_i, a_{i-1}, \ldots, a_1]$

$x = [a_{i+1}, \ldots, a_n]$

$\Rightarrow$ $y = a_{i+1} :: [a_i, a_{i-1}, \ldots]$

$x = [a_{i+2}, \ldots, a_n]$

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 $\rightarrow$ non-negative integers. Prove: For every iteration of the loop, ϕ(the 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

$\varphi(x, y, lst) = len\ x$

- list reverse

$\varphi(x, y, lst) = len\ x$