Class 24 – 4/22

Proving properties of imperative programs – "Hoare logic"

- Judgments, a.k.a. "Hoare formulas"
- Axioms
- Rules of inference

“Invariants” in programming

Invariants are relationships among the variables of a program that always hold.

- Within a class, invariants represent consistency among fields, e.g. “the count field is always the same as the number of non-zero entries in the values array,” or “if the visited bit of this node is set, and this node is not the entry point of the graph, then at least one predecessor’s visited bit is set.”

“Invariants” in programming

- In a loop, invariants represent relationships that hold *no matter how often the body of the loop is executed*.

```
x = x0; y = 1;
while ( x>0 )
      {y := y*x; x := x-1;}


a = a0; b = 0;
while (a != []) { b = b + hd a;
                  a = tl a; }
```

Hoare logic

- Hoare logic, introduced by C.A.R. Hoare, is an effort to formalize the proof of correctness of *imperative* programs.
- It is a proof system in which properties of programs are proved from properties of their component parts.
- It includes a formalization of the notion of *loop invariant*, which forms the hard part of most proofs.

<u>Correctness of imperative programs</u>

- "Hoare formula" says that if the variables in a program satisfy some properties, then after executing a given program, they satisfy some different properties.
- Examples:

x>0 { while (x>0)
{y := y*x; x := x-1;} } ***y = y * x!***

x=x0 & y=y0 { t := x; x := y; y := t }
x=y0 & y=x0

true { if (x<0) x := -x; } ***x = |x|***

true { n := length(a); b := [hd a];

a := tl a;

while (a != []) {

b = (hd a + hd b) :: b;

a = tl a; }

} $\boldsymbol{b_i = \sum_{k=0}^{n-i-1} a_k}$ (where b_i = hd (tl^i b),

and similarly for a_k)

Inference rules of Hoare logic

Judgments: P {S} Q

P, Q assertions about variables in the program

S a statement in this language:

```
Stmt -> Var :- Expr | Stmt;Stmt
        | if (Expr) then Stmt else Stmt
        | while (Expr) Stmt
```

Inference rules of Hoare logic

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

$$\frac{P \mathbin{\&} b\ \{S\}\ P}{P\ \{\text{while } (b)\ S\}\ P \mathbin{\&} \neg b}$$

$$\frac{P\ \{S_1\}\ Q \quad Q\ \{S_2\}\ R}{P\ \{S_1;\ S_2\}\ R}$$

$$\frac{P \Rightarrow P' \quad P'\ \{S\}\ Q' \quad Q' \Rightarrow Q}{P\ \{S\}\ Q}$$

$$\frac{P \mathbin{\&} b\ \{S_1\}\ Q \quad P \mathbin{\&} \neg b\ \{S_2\}\ Q}{P\ \{\text{if } (b) \text{ then } S_1 \text{ else } S_2\}\ Q}$$

Rule of assignment

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

Examples

$y=2 \{ x:=y \} \ x=2$

$y=2 \{ x:=2 \} \ y=x$

$x+1=n+1 \ \{ x:=x+1 \} \ x=n+1$

$x+1=n \ \{ x:=x+1 \} \ x=n$

$true \ \{ x:=2 \} \ x=2$

Rule of consequence

$$\frac{P \Rightarrow P' \quad P' \{S\}\ Q' \quad Q' \Rightarrow Q}{P\ \{S\}\ Q}$$

Sequence rule

$$\frac{P\ \{S_1\}\ Q \qquad Q\ \{S_2\}\ R}{P\ \{S_1;\ S_2\}\ R}$$

If rule

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

While rule

$$\frac{P \ \& \ b \ \{S\} \ P}{P \ \{\text{while } (b) \ S\} \ P \ \& \ \neg b}$$

Comments on Hoare logic

- Proofs in Hoare logic are *almost* syntax-directed, i.e. almost have the same shape as the program being proved. The only exceptions are the uses of the rule of consequence.

- Applying Hoare rules is largely mechanical – given A and Q, most of the proof (including P) can be generated automatically. Creativity is required mainly in determining the invariant in a while loop, because Q may not have the form "P & $\neg$b", and so a formula of that form needs to be found (after which the rule of consequence can be used, proving P & $\neg$b $\Rightarrow$ Q).

Example: gcd algorithm

a > 0 & b > 0 & a=a₀ & b=b₀ {

while ($a \neq b$)

if ($a > b$) then $a := a - b$;

else $b := b - a$;

} ***a = gcd(a₀, b₀)***