Automatic Program Verification I: A Logical Basis and its Implementation

Igarashi, London, and Luckham

Verification Condition Generator

- Symbolic execution
- Backwards substitution

Backwards Substitution

Is: a mechanical method of generating lemmas to be proved that guarantee the consistency of a program and its specifications

Needs: the programs to be verified must be written in a language that is axiomatically defined
A set of rules to generate subgoals and ultimately verification conditions (VCs) to be proved using the underlying deductive system(s)

Verification Condition Generator (VCG)

- Reformulate the axiomatic definition to produce a deterministic set of rules
- Rules generate subgoals and verfication conditions (VCs)

- This is a backwards substitution approach
- VCs correspond to Hoare's lemmas [ILL 75]

Rules are of the format

To prove:
the goal to be proved

We need to prove:
possibly empty list of subgoals

And to output:
possible empty list of lemmas

To prove
$P\{x := e\}Q$
Need to prove
$P \rightarrow Q_e^{\mathbf{x}}$

More generally,
To prove
$P\{A; x := e\}Q$
Need to prove
$P\{A\}Q_e^{\mathbf{x}}$

Assignment Proof Rule (V1)

$$\frac{P\{A\}\ Q_e^x}{P\{A;\ x := e\}Q}$$

Consequence Rules (V2)

$$\frac{P\{A\}Q,\ Q \rightarrow R}{P\{A;\ \text{assert } Q\}\ R}$$

$$\frac{P \rightarrow Q}{P\{\}Q}$$

$$\frac{P\{A\}(Q \rightarrow R)}{P\{A;\ Q\text{-if}\}R}$$

The proof rules are just a reformulation of the axioms and rules of inference to make the approach deterministic and thus mechanizable

Backwards Substitution Proof

A proof of P{S}Q is a sequence of sentences, the first of which is P{S}Q, and each sentence is either a lemma to be proved in the underlying logical system or a simpler sentence of the form P'{S'}Q'. Each sentence in the sequence is derived from a previous line by applying a verification rule.

The output of a backwards substitution proof is the generated lemmas (VCs)


```
PROCEDURE TEST (A, B: INTEGER;
                VAR X, Y, Z: INTEGER);
BEGIN
   X := A + B;
   Y := A - B;
   Z := X + Y
END;
```

ENTRY: true

EXIT: X = A + B & Y = A - B & Z = 2A

true{x:=a+b; y:=a-b; z:=x+y} x=a+b & y=a-b & z=2a

Compound Statement Rule (V6)

$$\frac{P\{S1; \ldots ; Sn\}R}{P\{\text{begin } S1; \ldots ; Sn \text{ end}\}R}$$

Iteration Rule (V3)

$$\frac{P\{A\}R,\ R \,\&\, B\{S\}R,\ R \,\&\, \sim B \rightarrow Q}{P\{A; \text{ assert } R; \text{ while } B \text{ do } S\}Q}$$


```
1  PROCEDURE  FACT ( N:INTEGER; VAR Y:INTEGER);
2  VAR X: INTEGER;
3  BEGIN
4     X := 0;
5     Y := 1;
6     ASSERT ( Y = X! & X ≤ N );
7     WHILE X < N DO BEGIN
8          X := X + 1;
9          Y := Y * X
10    END
11 END;

ENTRY:    N ≥ 0
EXIT:     Y = N!
```


Conditional Rules (V4)

$$\frac{P\{A; \text{ B-if; } S1\}R,\ \ P\{A; \sim\text{B-if; } S2\}R}{P\{A; \text{ if B then S1 else S2}\}R}$$

$$\frac{P\{A; \text{ B-if; } S\}R,\ \ P\{A; \sim\text{B-if}\}R}{P\{A; \text{ if B then S}\}R}$$


```
PROCEDURE SAMPLE (X,Y: INTEGER; B:BOOLEAN;
                                VAR Z: INTEGER);
BEGIN

   Z := X * Z;

   IF  B
     THEN  Z:= Z + Y
     ELSE  Z:=Z -Y

END;
```

ENTRY: TRUE

EXIT: (B & Z = Z' * X' + Y' | ~B & Z = Z' * X' - Y') & B = B'

What About Repeat-Until ?

$$\frac{}{P\{A; \text{ repeat S assert Q until } B\}R}$$