Proving partial correctness of `while`-programs

Lecture #6

Notations used in verification

- Assertions to be proved in program verification:
  - Statements of mathematics:

    $$(x + 1)^2 = x^2 + 2x + 1$$

  - Partial correctness specifications:

    $$\{P\}\ C\ \{Q\}$$

  - Total correctness specifications

    $$[P]\ C\ [Q]$$

Terms from formal logic

- Floyd-Hoare logic (FHL) gives rules for proving the partial and total correctness of programs, i.e. terms $\vdash \{P\}\ C\ \{Q\}$ and $\vdash [P]\ C\ [Q]$
- Predicate calculus gives rules for proving theorems of logic
- Arithmetics gives decision rules for proving statements about integers
- Theorems are statements, which can be proved to be true.
- Axioms are statements which are assumed to be true.
- $\vdash S$ means that *S* can be proved (unconditionally) using proof rules
- $\Gamma \vdash S$ means that S can be deduced from the assumptions Γ (from axioms $\Gamma = \{A_1, A_2, \ldots, A_n\}$).

FHL deduction systems

- *The inference rule –* an instruction on how to make a proof step
- The rules may have differenct form. The rules of FHL are specified with a notation of the form

$$\frac{\overbrace{\vdash S_1, \ldots, \vdash S_n}^{\text{hypothesis}}}{\underbrace{\vdash S}_{\text{conclusion}}}$$

- This means the conclusion $\vdash S$ may be deduced from the hypotheses $\vdash S_1, \ldots, \vdash S_n$
- The hypotheses can either be theorems of FHL or predicate calculus
- Example: (a rule of propositional logic**)**

$$\frac{\vdash p \Rightarrow q \qquad \vdash q \Rightarrow p}{\vdash p \Leftrightarrow q}$$

Elements of proof theory

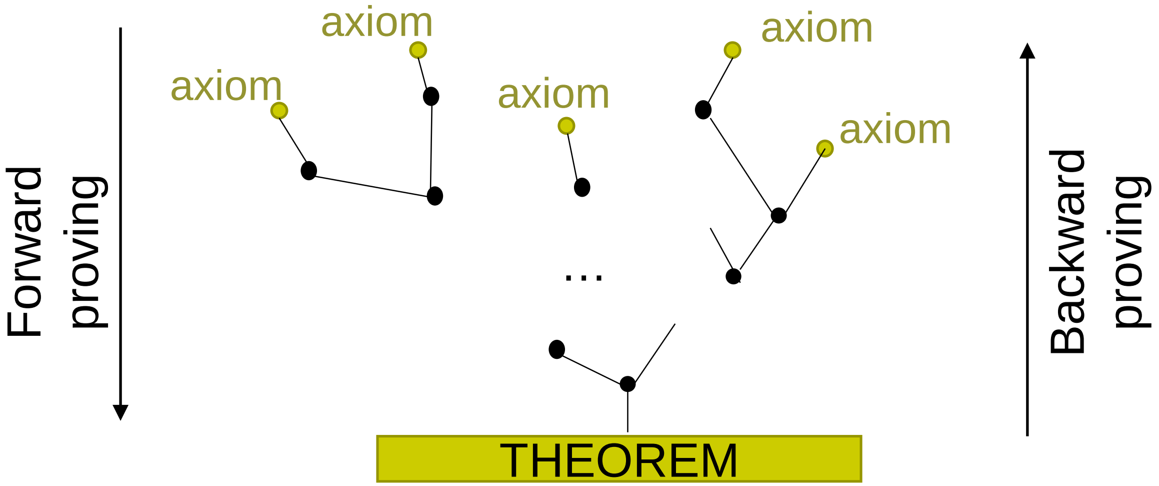
- *Deduction* (*proof*) - sequence (tree) of *statements* where every statement is either
  - an *axiom* *or*
  - deduced from true statements by proof rules
- Properties of the proof rules:
  - *Correctness* (*soundness*) - it is not possible to deduce something that is not correct from correct assumptions.
  - *Completeness* - all statements that are correct are deducible from axioms using the proof rules.
- *Deduction system* ≃ *set of axioms* (*or axiom schemas*) + *set of deduction rules*

Hoare type proof systems

- There is a special axiom or proof rule of each programming language construct
- Instead of concrete axioms FHL consists of axiom schemas that are instantiated by concrete program conditions
- The order of applying inference rules in the proof is determined by the syntactic structure of the program to be verified.
- This makes constructing proofs much easier.

Proof

- Typically the proof has a shape of a tree where
  - theorem is the root of the tree and
  - axioms are leaves.
  - The edges correspond to applications of inference rules



The rules of FHL: sequential composition

- Syntax:

$$\frac{\vdash \{P\}\ C1\ \{Q\}, \qquad\qquad \vdash \{Q\}\ C2\ \{R\},}{\vdash \{P\}\ C1\ ;\ C2\ \{R\}}$$

- Semantics:

  If Hoare triples $\{P\}C1\{Q\}$ and $\{Q\}C2\{R\}$, have proofs then also triple with sequential composition of $C1$ and $C2$, i.e. $\{P\}\ C1;\ C2\ \{R\}$ has a proof.

- Example:

$$\frac{\vdash \{X=1\}\ \texttt{X:=X+1}\ \ \{X=2\}\ \ \vdash \{X=2\}\ \texttt{X:=X+1}\ \ \{X=3\}}{\vdash \{X=1\}\ \texttt{X:=X+1;\ X:=X+1}\ \ \{X=3\}}$$

FHL rules: sequential composition example

Assume we have given tripples 1-3:

1. $\vdash \{X = x \wedge Y = y\}\ R := X\ \{R = x \wedge Y = y\}$
2. $\vdash \{R = x \wedge Y = y\}\ X := Y\ \{R = x \wedge X = y\}$
3. $\vdash \{R = x \wedge X = y\}\ Y := R\ \{Y = x \wedge X = y\}$

by sequential coposition rule (1.) and (2.) provide

4. $\vdash \{X = x \wedge Y = y\}\ R := X;\ \ X := Y\ \ \{R = x \wedge X = y\}$

and (4.), (3.) entail

5. $\vdash \{X = x \wedge Y = y\}\ R := X\ ;\ \ X := Y\ ;\ \ Y := R\ \{Y = x \wedge X = y\}$

FHL rules: SKIP-*axiom*

- Syntax:

$$\vdash \{P\}\ \texttt{SKIP}\ \{P\}$$

- Semantics:
  - Program state does not change when skip is executed
- Explanation:
  - This is axiom scheme where P may be any assertion
- Examples of concrete SKIP-axioms:
  - $\vdash \{Y = 2\}\ \texttt{SKIP}\ \{Y = 2\}$
  - $\vdash \{\mathrm{T}\}\ \texttt{SKIP}\ \{\mathrm{T}\}$
  - $\vdash \{Y = K \times X + R\}\ \texttt{SKIP}\ \{Y = K \times X + R\}$

FHL rules: assignment

- Syntax: $V := E$
- Semantics:
  The state is changed by assigning the value of the term E to the variable V. All other variables preserve their values
- Example: Y:=Y+5

  This adds 5 to the value of the variable Y.

- Variable substitution:
  - $P[E/V]$ denotes the result of replacing all occurrences of V in P by E.

- Example: $(X + 1 > X)\,[Y + Z / X] = ((Y + Z) + 1 > Y + Z)$
- Following property holds: $V[E/V] = E$

FHL: assignment axiom:

$$\vdash \{P[E/V]\}\ \texttt{V:=E}\ \{P\}$$

where

- *V* is variable, *E* is an expression, *P* is any statement
- $P[E/V]$ – denotes the result of substituting the term E for all occurrences of the variable V in statement P.

- Explanation:
  - the value of a variable V after executiong an assignment V:=E
  - equals the value of the expression E in the state before executing it.

- <u>Example</u>:
  - $\vdash \{Y = 2\}$ X:=2 $\{Y = X\}$
  - $\vdash \{Z = X^{**}Y\}$ X:=X**Y $\{Z = X\}$

FHL rules: precondition strenghtening

$$\frac{\vdash P \Rightarrow P' \qquad\qquad \vdash \{P'\}\ C\ \{Q\}}{\vdash \{P\}\ C\ \{Q\}}$$

- Application example
  From implication $\vdash X = n \Rightarrow X + 1 = n + 1$
  and
  assignment axiom $\vdash \{X + 1 = n + 1\}\ X := X + 1\ \{X = n + 1\}$
  we can deduce
  $$\vdash \{X = n\}\ X := X + 1\ \{X = n + 1\},$$

  where n is auxilliary variable that occurs only in pre-and post-conditions .

FHL rules: postcondition weakening

$$\frac{\vdash \{P\}\ C\ \{Q'\} \qquad \vdash Q' \Rightarrow Q}{\vdash \{P\}\ C\ \{Q\}}$$

- Example (application of rules in forward reasoning):

Proof step	Used inference rule
1. $\vdash \{R = X \wedge 0{=}0\}\ Q{:=}\ 0\ \{R{=}X \wedge Q{=}0\}$	(assignment axiom)
2. $\vdash R = X \Rightarrow R = X \wedge 0 = 0$	(logic equality
3. $\vdash \{R = X\}\ Q := 0\ \{R = X \wedge Q = 0\}$	(precondition strengthening)
4. $\vdash R = X \wedge Q = 0 \Rightarrow R = X + (Y \times Q)$	(arithmetic equality)
5. $\vdash \{R = X\}\ Q := 0\ \{R = X + (Y \times Q)\}$	(postcondition weakening)

FHL rules: `BEGIN-END` -blocks

- Syntax:

  `BEGIN VAR` *V*`1; … ;`*V*`n ;` *C* `END`
- Semantics:
  - Variables *V*`1; … ;`*V*`n` are used locally within the block. After *C* is executed the values of *V*`1; … ;`*V*`n` are restored to the values they had befor the block was entered
  - The initial values for *V*`1; … ;`*V*`n` in the block are unspecified.

- Example:

  `BEGIN VAR R; R := X; X:=Y; Y:=R END`
  - Variables `X` and `Y` exchange their values by using an auxiliary variable `R`.

FHL rules: `BEGIN-END` -blocks

Block-rule**:**

$$\frac{\vdash \{P\}\ C\ \{Q\}}{\vdash \{P\}\ \texttt{BEGIN VAR}\ V\texttt{1; … ;}V\texttt{n ;}\ C\ \texttt{END}\ \{Q\}}$$

where none of the variables V`1; … ;`V`n` occur in P or Q.

Explanation:

This restriction of variable occurrence in P and Q is because their value is determined locally only within the block. Their valuation outside the block may be different.

FHL rules: `IF`- command

- Syntax:

  $\{P\}$ `IF` S `THEN` $C1$ `ELSE` $C2$ $\{Q\}$

- Semantics:
  - If the statement S is *true* (in current state), then $C1$ is executed
  - If S *is false* then $C2$ is executed

- Example:
  - `IF X < Y  THEN  MAX:= Y  ELSE MAX:= X`

FHL rules: `IF`- command

- `IF` - rule1 (one branch):

$$\frac{\vdash \{P \wedge S\}\ C\ \{Q\} \qquad \vdash P \wedge \neg S \Rightarrow Q}{\vdash \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C\ \{Q\}}$$

- `IF`- rule 2 (two branches):

$$\frac{\vdash \{P \wedge S\}\ C_1\ \{Q\}, \qquad \vdash \{P \wedge \neg S\}\ C_2\ \{Q\}}{\vdash \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

FHL rules: application example of `IF`- command

- <u>Example</u>:

Given

- $\vdash \{T \wedge \quad X >= Y\}$ `MAX := X` $\{MAX = max(X, Y)\}$
- $\vdash \{T \wedge \neg(X >= Y)\}$ `MAX := Y` $\{MAX = max(X, Y)\}$

By `IF`-rule 2 it follows:

- $\vdash \{T\}$ `IF X>=Y THEN MAX:=X ELSE  MAX:=Y` $\{MAX= max(X, Y)\}$

FHL rules: `WHILE`-command

- <u>Syntax</u>: `WHILE` S `DO` C

- <u>Semantics:</u>
  - If the statement S is true in the current state, then C is executed and the `WHILE` command is repeated.
  - If S is false, then exit the command,
  - Command C is repeatedly executed until the value of S becomes *false*
  - If S never becomes *false*, then the execution never terminates

- <u>Example</u>:

`WHILE ¬(X = 0) DO X := X – 2`

For which values of `X` the command does not terminate?

FHL rules: `WHILE`-command

- Invariants

  Suppose

  $$\vdash \{P \wedge S\}\ C\ \{P\}$$

  then P is an invariant of C whenever S holds.

- Explanation (`WHILE`-rule):
  - if the execution of `WHILE`-command body C preserves truth value of P *once*, then it preserves this truth value for arbitrary number of executions of C.
  - If `WHILE`-command has terminated, then loop condition S must be *false* (because this is `WHILE` termination condition).

FHL rules: `WHILE`-command

- (Simple) WHILE–rule:

$$\frac{\vdash \{P \wedge S\}\ \mathtt{C}\ \{P\}}{\vdash \{P\}\ \mathtt{WHILE}\ S\ \mathtt{DO}\ \mathtt{C}\ \{P \wedge \neg S\}}$$

- Extended WHILE-rule:

$$\frac{\vdash P \Rightarrow R \qquad \vdash \{R \wedge S\}\ \mathtt{C}\ \{R\} \qquad \vdash R \wedge \neg S \Rightarrow Q}{\vdash \{P\}\ \mathtt{WHILE}\ S\ \mathtt{DO}\ \mathtt{C}\ \{Q\}}$$

`WHILE`-command: invariant

How to find an invariant?

- It must hold initially when entering the loop
- With negated test it must establish the result of loop
- The body must leave it unchanged

- Intuition:
  - The invariant says that what *has been done* so far together with what remains *to be done* gives the *desired result*.
  - Analogy with milestone where one face indicates the distance passed and the other the distance to go, their sum is the total distance between the departure point and destination.

WHILE-command: example 1

- Example (factorial program 1):

$$\{X = n \wedge Y = 1\}$$

```
WHILE X≠0 DO
BEGIN Y:=Y×X; X:=X-1  END
```

$$\{X = 0 \wedge Y = n!\}$$

- Analyze the variable values
  - Finally `X = 0` and `Y = n!`
  - Initially `X = n` and `Y = 1`
  - On each loop Y is increased and X is decreased.

`WHILE`-command: example 1

- How the variables keep their values in execution?
  - Y holds the result so far
  - X! is what remains to be computed
  - n! is the desired result

→ The invariant is $X! \times Y = n!$

`WHILE`-command: example 2

- Example (factorial program 2):

$\{X = n \wedge Y = 1\}$

```
WHILE X<N DO
BEGIN X:=X+1; Y:=Y×X; END
```

$\{Y = N!\}$

- Analyze the variable values
  - Finally `X = N` and `Y = N!`
  - Initially `X = 0` and `Y = 1`
  - On each loop both `X` and `Y` increase.

WHILE-command: example 2

```
{X=0 ∧ Y=1}
 WHILE X<N DO
   BEGIN X:=X+1; Y:=Y×X END
{Y=N!}
```

- How the values of variables evolve in execution?
  - At end $Y = N!$
  - and $\neg(X<N) \Rightarrow X=N$
  - Consquently the invariant must be

$$Y = X! \wedge X \leq N$$

FHL rules: conjunction and disjunction of specifications

$$\frac{\vdash \{P_1\}\ C\ \{Q_1\} \qquad\qquad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}}$$

$$\frac{\vdash \{P_1\}\ C\ \{Q_1\} \qquad\qquad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \vee P_2\}\ C\ \{Q_1 \vee Q_2\}}$$

- These rules allow splitting large tripples into simpler ones and prove them separately. To prove

$$\vdash \{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}.$$

- it sufices proving independently

$$\vdash \{P_1\}\ C\ \{Q_1\} \quad \text{and} \quad \vdash \{P_2\}\ C\ \{Q_2\}$$

FHL rules: `FOR`-command

- Syntax:

```
FOR V := E1 UNTIL  E2  DO C
```

  - Restriction: index variable `V` must not occur in `E1` or `E2` or be the left hand side of an assignment in `C`.

- Semantics:
  - If the values of terms *E*1 and *E*2 are positive numbers $e1$ and $e2$, where $e1 \leq e2$, then C is executed $(e2 - e1) + 1$ times
  - with the variable *V* taking values $e1, e1{+}1, e1{+}2, \ldots, e2$.
  - for any other value of *V* the `FOR`-command acts as `skip`.

- Example:

```
FOR  N:=1 UNTIL  M  DO X:=X+N
```

  - expressions `E1` and `E2` are evaluated only once at the entry to `FOR`-command;
  - if `E1` and `E2` do not have positive integer value or `E1>E2`, then `FOR`-command does nothing.

Reduction to WHILE-command

- `FOR`-command

  `FOR V:=E`$_1$ `UNTIL E`$_2$ `DO C`

  is equivalent to `WHILE`-program

```
BEGIN VAR V;
  V:=E1;
  WHILE V≥E1 ∧ V≤E2 DO
   BEGIN
     C;
     V:=V+1
    END
END
```

Annotating the FOR-command

- Having an annotated FOR-command

  $\{P\}$ `FOR V:= E`$_1$ `UNTIL E`$_2$ `DO` $\{R\}$ `C` $\{Q\}$

- we can transform it to equivalent annotated `WHILE` program
- *R* of this `WHILE` program must include condition $V \leq E_2+1$

$\{P\}$

```
BEGIN VAR V;
   V:= E1;
   WHILE V≥E1 ∧ V≤E2 DO {R} BEGIN
        C;
        V:=V+1
   END
END
```

$\{Q\}$

FHL rules: FOR-command

- FOR–axiom:

$$\vdash \{P \land (E2 < E1)\}\ \texttt{FOR}\ V\texttt{:=E1 UNTIL E2 DO C}\ \{P\}$$

- (Primary) FOR–rule:

$$\frac{\vdash \{P \land (E1 \le V) \land (V \le E2)\}\ \texttt{C}\ \{P[V+1/V]\}}{\vdash \{P[E1/V]\} \land (E1 \le E2)\}\ \texttt{FOR}\ V\texttt{:=E1 UNTIL E2 DO C}\ \{P[E2+1/V]\},}$$

- Extended FOR–rule:

$$\frac{\vdash P \Rightarrow R[E1/V] \qquad \vdash R[E2+1/V] \Rightarrow Q \qquad \vdash P \land (E2 < E1) \Rightarrow Q \qquad \vdash \{R \land (E1 \le V) \land (V \le E2)\}\ \texttt{C}\ \{R[V+1/V]\}}{\vdash \{P\}\ \texttt{FOR V:=E1 UNTIL E2 DO}\ \{R\}\ \texttt{C}\ \{Q\}}$$

Summary

- The proof system must be *sound* and *complete,*
- i.e. it is necessary to show that axioms are valid and inference rules entail true conclusions if the hypothesis are true.
- The calculus is complete if all its valid assertions are also provable.
- FHL is relative complete, if for all programs of given programming language the FHL triples expressible in it can be transformed to programming command free logic formuli.
- Ed. Clarke proved that there is not sound and complete FHL for languages that include recursion, static scoping, global variables and parametrized procedure calls.