Methodologies for Software Processes

Lecture 6- Hoare Logic

(The lecture slides and the notes are taken from Prof. Mike Gordon from Cambridge University)

Conditionals

- Syntax: `IF` S `THEN` C_1 `ELSE` C_2
- Semantics:
  - if the statement S is true in the current state, then C_1 is executed
  - if S is false, then C_2 is executed
- Example: `IF X<Y THEN MAX:=Y ELSE MAX:=X`
  - the value of the variable `MAX` it set to the maximum of the values of `X` and `Y`

The Conditional Rule

The conditional rule

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

- **From Assignment Axiom + Precondition Strengthening and**

  $\vdash$ (X$\geq$Y $\Rightarrow$ X = max(X,Y)) $\wedge$ ($\neg$(X$\geq$Y) $\Rightarrow$ Y = max(X,Y))

  it follows that

  $\vdash$ {T $\wedge$ X$\geq$Y} MAX:=X {MAX=max(X,Y)}

  and

  $\vdash$ {T $\wedge$ $\neg$(X$\geq$Y)} MAX:=Y {MAX=max(X,Y)}

- **Then by the conditional rule it follows that**

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

WHILE-commands

- Syntax: `WHILE` S `DO` C
- Semantics:
  - 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 nothing is done
  - thus C is repeatedly executed until the value of S becomes false
  - if S never becomes false, then the execution of the command never terminates
- Example: `WHILE` $\neg$`(X=0) DO X:= X-2`
  - if the value of `X` is non-zero, then its value is decreased by `2` and then the process is repeated
- This `WHILE`-command will terminate (with `X` having value `0`) if the value of `X` is an even non-negative number
  - in all other states it will not terminate

Invariants

- Suppose $\vdash \{P \wedge S\}\ C\ \{P\}$
- P is said to be an *invariant* of C whenever S holds
- The WHILE-rule says that
  - if P is an invariant of the body of a WHILE-command whenever the test condition holds
  - then P is an invariant of the whole WHILE-command
- In other words
  - if executing C *once* preserves the truth of P
  - then executing C *any number of times* also preserves the truth of P
- The WHILE-rule also expresses the fact that after a WHILE-command has terminated, the test must be false
  - otherwise, it wouldn't have terminated

The WHILE-Rule

The WHILE-rule

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

- It is easy to show

  $\vdash$ {X=R+(Y×Q)∧Y≤R} R:=R-Y; Q:=Q+1 {X=R+(Y×Q)}

- Hence by the WHILE-rule with P = 'X=R+(Y×Q)' and S = 'Y≤R'

  $\vdash$ {X=R+(Y×Q)}
  WHILE Y≤R DO
  (R:=R-Y; Q:=Q+1)
  {X=R+(Y×Q) ∧ ¬(Y≤R)}

Example

- From the previous slide

$$\vdash \{X=R+(Y\times Q)\}$$
$$\texttt{WHILE } Y\leq R \texttt{ DO}$$
$$\texttt{(R:=R-Y; Q:=Q+1)}$$
$$\{X=R+(Y\times Q) \wedge \neg(Y\leq R)\}$$

- It is easy to deduce that

$$\vdash \{T\}\ \texttt{R:=X; Q:=0}\ \{X=R+(Y\times Q)\}$$

- Hence by the sequencing rule and postcondition weakening

$$\vdash \{T\}$$
$$\texttt{R:=X;}$$
$$\texttt{Q:=0;}$$
$$\texttt{WHILE } Y\leq R \texttt{ DO}$$
$$\texttt{(R:=R-Y; Q:=Q+1)}$$
$$\{R<Y \wedge X=R+(Y\times Q)\}$$

Summary

- We have given:
  - a notation for specifying what a program does
  - a way of proving that it meets its specification
- Now we look at ways of finding proofs and organising them:
  - finding invariants
  - derived rules
  - backwards proofs
  - annotating programs prior to proof
- Then we see how to automate program verification
  - the automation mechanises some of these ideas

How does one find an invariant?

The WHILE-rule

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

- Look at the facts:
  - invariant P must hold initially
  - with the negated test $\neg S$ the invariant P must establish the result
  - when the test S holds, the body must leave the invariant P unchanged
- Think about how the loop works – the invariant should say that:
  - what has been done so far together with what remains to be done
  - holds at each iteration of the loop
  - and gives the desired result when the loop terminates

Example

- Consider a factorial program

```
{X=n ∧ Y=1}
 WHILE X≠0 DO
   (Y:=Y×X; X:=X-1)
{X=0 ∧ Y=n!}
```

- Look at the facts
  - initially `X=n` and `Y=1`
  - finally `X=0` and `Y=n!`
  - on each loop `Y` is increased and, `X` is decreased
- Think how the loop works
  - `Y` holds the result so far
  - `X!` is what remains to be computed
  - `n!` is the desired result
- The invariant is `X!×Y = n!`
  - 'stuff to be done' × 'result so far' = 'desired result'
  - decrease in `X` combines with increase in `Y` to make invariant

Related example

$\{X=0 \wedge Y=1\}$
WHILE $X<N$ DO $(X:=X+1;\ Y:=Y\times X)$
$\{Y=N!\}$

- Look at the Facts
  - initially $X=0$ and $Y=1$
  - finally $X=N$ and $Y=N!$
  - on each iteration both X an Y increase: X by 1 and Y by X
- An invariant is $Y = X!$
- At end need $Y = N!$, but WHILE-rule only gives $\neg(X<N)$
- Ah Ha! Invariant needed: $\boxed{Y = X! \wedge X\leq N}$
- At end $X \leq N \wedge \neg(X<N) \Rightarrow X=N$
- Often need to strenthen invariants to get them to work
  - typical to add stuff to 'carry along' like $X\leq N$

Conjunction and Disjunction

Specification conjunction

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

Specification disjunction

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

- These rules are useful for splitting a proof into independent bits
  - they enable $\vdash \{\mathtt{P}\}\ \mathtt{C}\ \{\mathtt{Q}_1 \wedge \mathtt{Q}_2\}$ to be proved by proving separately that both $\vdash \{\mathtt{P}\}\ \mathtt{C}\ \{\mathtt{Q}_1\}$ and also that $\vdash \{\mathtt{P}\}\ \mathtt{C}\ \{\mathtt{Q}_2\}$
- Any proof with these rules could be done without using them
  - i.e. they are theoretically redundant (proof omitted)
  - however, useful in practice

Combining Multiple Steps

- Proofs involve lots of tedious fiddly small steps
  - similar sequences are used over and over again
- It is tempting to take short cuts and apply several rules at once
  - this increases the chance of making mistakes
- Example:
  - by assignment axiom & precondition strengthening
    - $\vdash \{\mathtt{T}\}\ \mathtt{R} := \mathtt{X}\ \{\mathtt{R} = \mathtt{X}\}$
- Rather than:
  - by the assignment axiom
    - $\vdash \{\mathtt{X} = \mathtt{X}\}\ \mathtt{R} := \mathtt{X}\ \{\mathtt{R} = \mathtt{X}\}$
  - by precondition strengthening with $\vdash\ \mathtt{T} \Rightarrow \mathtt{X}{=}\mathtt{X}$
    - $\vdash \{\mathtt{T}\}\ \mathtt{R} := \mathtt{X}\ \{\mathtt{R} = \mathtt{X}\}$

Derived rules for finding proofs

- Suppose the goal is to prove $\{Precondition\}\ Command\ \{Postcondition\}$
- If there were a rule of the form

$$\frac{\vdash\ H_1,\ \cdots,\ \vdash\ H_n}{\vdash\ \{P\}\ C\ \{Q\}}$$

  then we could instantiate

  $P \mapsto Precondition$, $C \mapsto Command$, $Q \mapsto Postcondition$

  to get instances of $H_1,\ \cdots, H_n$ as subgoals
- Some of the rules are already in this form e.g. the sequencing rule
- We will derive rules of this form for all commands
- Then we use these derived rules for mechanising Hoare Logic proofs

Derived Rules

- We will establish derived rules for all commands

$$\frac{\ldots}{\vdash \{P\}\ V := E\ \{Q\}}$$

$$\frac{\ldots}{\vdash \{P\}\ C_1 ; C_2\ \{Q\}}$$

$$\frac{\ldots}{\vdash \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

$$\frac{\ldots}{\vdash \{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \{Q\}}$$

- These support 'backwards proof' starting from a goal $\{P\}\ C\ \{Q\}$

The Derived Assignment Rule

- **An example proof**

1. $\vdash$ $\{\texttt{R=X} \wedge \texttt{0=0}\}$ $\texttt{Q:=0}$ $\{\texttt{R=X} \wedge \texttt{Q=0}\}$ By the assignment axiom.
2. $\vdash$ $\texttt{R=X} \Rightarrow \texttt{R=X} \wedge \texttt{0=0}$ By pure logic.
3. $\vdash$ $\{\texttt{R=X}\}$ $\texttt{Q:=0}$ $\{\texttt{R=X} \wedge \texttt{Q=0}\}$ By precondition strengthening.

- **Can generalise this proof to a proof schema:**

1. $\vdash$ $\{Q[E/V]\}$ $V:=E$ $\{Q\}$ By the assignment axiom.
2. $\vdash$ $P \Rightarrow Q[E/V]$ By assumption.
3. $\vdash$ $\{P\}$ $V:=E$ $\{Q\}$ By precondition strengthening.

- **This proof schema justifies:**

Derived Assignment Rule

$$\frac{\vdash P \Rightarrow Q[E/V]}{\vdash \{P\}\ V:=E\ \{Q\}}$$

- **Note:** $Q[E/V]$ **is the weakest liberal precondition** $wlp(V:=E, Q)$

- **Example proof above can now be done in one less step**

1. $\vdash$ $\texttt{R=X} \Rightarrow \texttt{R=X} \wedge \texttt{0=0}$ By pure logic.
2. $\vdash$ $\{\texttt{R=X}\}$ $\texttt{Q:=0}$ $\{\texttt{R=X} \wedge \texttt{Q=0}\}$ By derived assignment.

Derived Sequenced Assignment Rule

- The following rule will be useful later

Derived Sequenced Assignment Rule

$$\frac{\vdash \{P\}\ C\ \{Q[E/V]\}}{\vdash \{P\}\ C;V:=E\ \{Q\}}$$

- Intuitively work backwards:
  - push Q 'through' $V:=E$, changing it to $Q[E/V]$
- Example: By the assignment axiom:

  $\vdash$ {X=x∧Y=y} R:=X {R=x∧Y=y}
- Hence by the sequenced assignment rule

  $\vdash$ {X=x∧Y=y} R:=X; X:=Y {R=x∧X=y}

Backward Hoare & forward Floyd assignment axioms

- **Recall Hoare (backward) and Floyd (forward) assignment axioms**

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

  Floyd axiom: $\vdash \{P\}\ V := E\ \{\exists v.\ V = E[v/V]\ \wedge\ P[v/V]\}$

The Derived While Rule

Derived While Rule

$$\frac{\vdash P \Rightarrow R \quad \vdash \{R \wedge S\}\, C\, \{R\} \quad \vdash R \wedge \neg S \Rightarrow Q}{\vdash \{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \{Q\}}$$

- This follows from the While Rule and the rules of consequence
- Example: it is easy to show

$\vdash$ R=X $\wedge$ Q=0 $\Rightarrow$ X=R+(Y$\times$Q)

$\vdash$ {X=R+(Y$\times$Q)$\wedge$Y$\leq$R} R:=R-Y; Q:=Q+1 {X=R+(Y$\times$Q)}

$\vdash$ X=R+(Y$\times$Q)$\wedge\neg$(Y$\leq$R) $\Rightarrow$ X=R+(Y$\times$Q)$\wedge\neg$(Y$\leq$R)

- Then, by the derived While rule

```
⊢ {R=X ∧ Q=0}
   WHILE Y≤R DO
     (R:=R-Y; Q:=Q+1)
 {X=R+(Y×Q) ∧ ¬(Y≤R)}
```

The Derived Sequencing Rule

- The rule below follows from the sequencing and consequence rules

The Derived Sequencing Rule

$$\frac{\begin{array}{ll} & \vdash P \Rightarrow P_1 \\ \vdash \{P_1\}\ C_1\ \{Q_1\} & \vdash Q_1 \Rightarrow P_2 \\ \vdash \{P_2\}\ C_2\ \{Q_2\} & \vdash Q_2 \Rightarrow P_3 \\ \vdots & \vdots \\ \vdash \{P_n\}\ C_n\ \{Q_n\} & \vdash Q_n \Rightarrow Q \end{array}}{\vdash \{P\}\ C_1;\ \ldots\ ;\ C_n\ \{Q\}}$$

Example

- By the assignment axiom

(i) $\vdash$ {X=x∧Y=y} R:=X {R=x∧Y=y}

(ii) $\vdash$ {R=x∧Y=y} X:=Y {R=x∧X=y}

(iii) $\vdash$ {R=x∧X=y} Y:=R {Y=x∧X=y}

- Using the derived sequencing rule, it can be deduced *in one step* from (i), (ii), (iii) and the fact that for any P: $\vdash P \Rightarrow P$

$\vdash$ {X=x ∧ Y=y} R:=X; X:=Y; Y:=R {Y=x ∧ X=y}

Forwards and backwards proof

- Previously it was shown how to prove $\{P\}C\{Q\}$ by
  - proving properties of the components of C
  - and then putting these together, with the appropriate proof rule, to get the desired property of C
- For example, to prove $\vdash \{P\}C_1;C_2\{Q\}$
- First prove $\vdash \{P\}C_1\{R\}$ and $\vdash \{R\}C_2\{Q\}$
- then deduce $\vdash \{P\}C_1;C_2\{Q\}$ by sequencing rule
- This method is called *forward proof*
  - move forward from axioms via rules to conclusion
- The problem with forwards proof is that it is not always easy to see what you need to prove to get where you want to be
- It is more natural to work backwards
  - starting from the goal of showing $\{P\}C\{Q\}$
  - generate subgoals until problem solved

Example

- Suppose one wants to show

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

- By the assignment axiom and derived sequenced assignment rule it is sufficient to show the subgoal

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

- Similarly this subgoal can be reduced to

  $\{X{=}x \wedge Y{=}y\}\ R{:=}X\ \{R{=}x \wedge Y{=}y\}$

- This clearly follows from the assignment axiom

Backwards versus Forwards Proof

- Backwards proof just involves using the rules backwards
- Given the rule

$$\frac{\vdash S_1 \quad \ldots \quad \vdash S_n}{\vdash S}$$

- Forwards proof says:
  - **if we have proved** $\vdash S_1 \ \ldots \vdash S_n$ **we can deduce** $\vdash S$
- Backwards proof says:
  - **to prove** $\vdash S$ **it is sufficient to prove** $\vdash S_1 \ \ldots \vdash S_n$
- Having proved a theorem by backwards proof, it is simple to extract a forwards proof

Example Backwards Proof

- To prove

$\vdash$ {T}
R:=X;
Q:=0;
WHILE Y$\leq$R DO
(R:=R-Y; Q:=Q+1)
{X=R+(Y$\times$Q) $\wedge$ R<Y}

- By the sequencing rule, it is sufficient to prove

(i) $\vdash$ {T} R:=X; Q:=0 {R=X $\wedge$ Q=0}

(ii) $\vdash$ {R=X $\wedge$ Q=0}
WHILE Y$\leq$R DO
(R:=R-Y; Q:=Q+1)
{X=R+(Y$\times$Q) $\wedge$ R<Y}

- Where does {R=X $\wedge$ Q=0} come from? (Answer later)

Example Continued (1)

- From previous slide:

  (i) $\vdash$ {T} R:=X; Q:=0 {R=X $\wedge$ Q=0}

- To prove (i), by the sequenced assignment axiom, we must prove:

  (iii) $\vdash$ {T} R:=X {R=X $\wedge$ 0=0}

- To prove (iii), by the derived assignment rule, we must prove:

  $\vdash$ T $\Rightarrow$ X=X $\wedge$ 0=0

- This is true by pure logic

Example continued (2)

- From an earlier slide:

(ii) $\vdash$ {R=X $\wedge$ Q=0}
WHILE Y$\leq$R DO
(R:=R-Y; Q:=Q+1)
{X=R+(Y$\times$Q) $\wedge$ R<Y}

- To prove (ii), by the derived while rule, we must prove:

(iv) R=X $\wedge$ Q=0 $\Rightarrow$ (X = R+(Y$\times$Q))

(v) X = R+Y$\times$Q $\wedge$ $\neg$(Y$\leq$R) $\Rightarrow$ (X = R+(Y$\times$Q) $\wedge$ R<Y)

and

(vi) {X = R+(Y$\times$Q) $\wedge$ (Y$\leq$R)}
(R:=R-Y; Q:=Q+1)
{X=R+(Y$\times$Q)}

- (iv) and (v) are proved by pure arithmetic

Example Continued (3)

- To prove (vi), we must prove

$$\begin{array}{ll} & \{\texttt{X = R+(Y}\times\texttt{Q)} \wedge \texttt{(Y}\leq\texttt{R)}\} \\ \text{(vii)} & \texttt{(R:=R-Y; Q:=Q+1)} \\ & \{\texttt{X=R+(Y}\times\texttt{Q)}\} \end{array}$$

- To prove (vii), by the sequenced assignment rule, we must prove

$$\begin{array}{ll} & \{\texttt{X=R+(Y}\times\texttt{Q)} \wedge \texttt{(Y}\leq\texttt{R)}\} \\ \text{(viii)} & \texttt{R:=R-Y} \\ & \{\texttt{X=R+(Y}\times\texttt{(Q+1))}\} \end{array}$$

- To prove (viii), by the derived assignment rule, we must prove

(ix) $\texttt{X=R+(Y}\times\texttt{Q)} \wedge \texttt{Y}\leq\texttt{R} \Rightarrow \texttt{(X = (R-Y)+(Y}\times\texttt{(Q+1)))}$

- This is true by arithmetic

Annotations

- The sequencing rule introduces a new statement R

$$\frac{\vdash \{P\}\ C_1\ \{R\} \qquad \vdash \{R\}\ C_2\ \{Q\}}{\vdash \{P\}\ C_1;C_2\ \{Q\}}$$

- To apply this backwards, one needs to find a suitable statement R
- If C_2 is V:=E then sequenced assignment gives $Q[E/V]$ for R
- If C_2 isn't an assignment then need some other way to choose R
- Similarly, to use the derived While rule, must invent an invariant

Annotate First

- It is helpful to think up these statements before you start the proof and then annotate the program with them
  - the information is then available when you need it in the proof
  - this can help avoid you being bogged down in details
  - the annotation should be true whenever control reaches that point
- Example, the following program could be annotated at the points P_1 and P_2 indicated by the arrows

```
{T}
   R:=X;
   Q:=0; {R=X ∧ Q=0} ⟵P1
   WHILE Y≤R DO {X = R+Y×Q} ⟵P2
      (R:=R-Y; Q:=Q+1)
{X = R+Y×Q ∧ R<Y}
```

Summary

- We have looked at three ways of organizing proofs that make it easier for humans to apply them:
  - deriving "bigger step" rules
  - backwards proof
  - annotating programs
- Next we see how these techniques can be used to mechanize program verification

NEW TOPIC: Mechanizing Program Verification

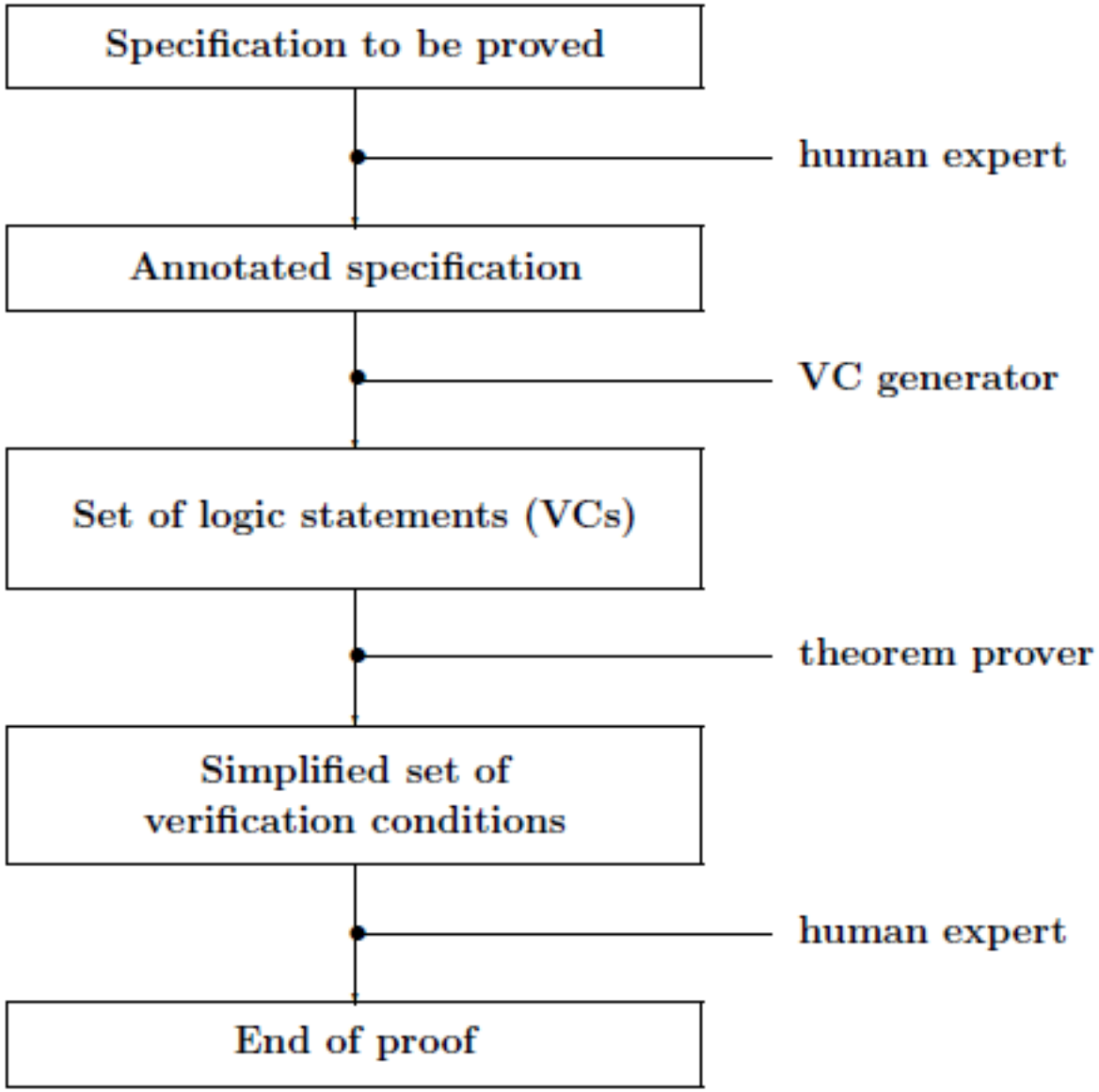
- The architecture of a simple program verifier will be described
- Justified with respect to the rules of Floyd-Hoare logic
- It is clear that
  - proofs are long and boring, even if the program being verified is quite simple
  - lots of fiddly little details to get right, many of which are trivial, e.g.

$$\vdash \ (\mathtt{R}{=}\mathtt{X} \ \wedge \ \mathtt{Q}{=}0) \ \Rightarrow \ (\mathtt{X} = \mathtt{R} + \mathtt{Y}{\times}\mathtt{Q})$$

Mechanization

- Goal: automate the routine bits of proofs in Floyd-Hoare logic
- Unfortunately, logicians have shown that it is impossible in principle to design a *decision procedure* to decide automatically the truth or falsehood of an arbitrary mathematical statement
- This does not mean that one cannot have procedures that will prove many *useful* theorems
  - the non-existence of a general decision procedure merely shows that one cannot hope to prove *everything* automatically
  - in practice, it is quite possible to build a system that will mechanize the boring and routine aspects of verification

Architecture of a Verifier



Commentary

- Input: a Hoare triple annotated with mathematical statements
  - these annotations describe relationships between variables
- The system generates a set of purely mathematical statements called *verification conditions* (or VCs)
- If the verification conditions are provable, then the original specification can be deduced from the axioms and rules of Hoare logic
- The verification conditions are passed to a *theorem prover* program which attempts to prove them automatically
  - if it fails, advice is sought from the user

Verification conditions

- The three steps in proving $\{P\}C\{Q\}$ with a verifier
- 1 The program C is *annotated* by inserting statements (*assertions*) expressing conditions that are meant to hold at intermediate points
  - tricky: needs intelligence and good understanding of how the program works
  - automating it is an artificial intelligence problem
- 2 A set of logic statements called *verification conditions* (VCs) is then generated from the annotated specification
  - this is purely mechanical and easily done by a program
- 3 The verification conditions are proved
  - needs automated theorem proving (i.e. more artificial intelligence)
- To improve automated verification one can try to
  - reduce the number and complexity of the annotations required
  - increase the power of the theorem prover
  - still a research area

Validity of Verification Conditions

- It will be shown that
  - **if one can prove all the verification conditions generated from $\{P\}C\{Q\}$**
  - **then $\vdash \{P\}C\{Q\}$**
- Step $\boxed{2}$ converts a verification problem into a conventional mathematical problem
- The process will be illustrated with:

```
{T}
   R:=X;
   Q:=0;
   WHILE Y≤R DO
      (R:=R-Y; Q:=Q+1)
{X = R+Y×Q ∧ R<Y}
```

Example

- Step 1 is to insert annotations P_1 and P_2

```
{T}
   R:=X;
   Q:=0; {R=X ∧ Q=0} ⟵P1
   WHILE Y≤R DO {X = R+Y×Q} ⟵P2
      (R:=R-Y; Q:=Q+1)
{X = R+Y×Q ∧ R<Y}
```

- The annotations P_1 and P_2 state conditions which are intended to hold *whenever* control reaches them

Example Continued

```
{T}
   R:=X;
   Q:=0; {R=X ∧ Q=0} ⟵P1
   WHILE Y≤R DO {X = R+Y×Q} ⟵P2
      (R:=R-Y; Q:=Q+1)
{X = R+Y×Q ∧ R<Y}
```

- Control only reaches the point at which P_1 is placed once
- It reaches P_2 each time the WHILE body is executed
  - whenever this happens X=R+Y×Q holds, even though the values of R and Q vary
  - P_2 is an *invariant* of the WHILE-command

Generating and Proving Verification Conditions

- Step 2 will generate the following four verification conditions

(i) $T \Rightarrow (X=X \wedge 0=0)$

(ii) $(R=X \wedge Q=0) \Rightarrow (X = R+(Y \times Q))$

(iii) $(X = R+(Y \times Q)) \wedge Y \leq R) \Rightarrow (X = (R-Y)+(Y \times (Q+1)))$

(iv) $(X = R+(Y \times Q)) \wedge \neg(Y \leq R) \Rightarrow (X = R+(Y \times Q) \wedge R<Y)$

- Notice that these are statements of arithmetic
  - the constructs of our programming language have been 'compiled away'
- Step 3 consists in proving the four verification conditions
  - easy with modern automatic theorem provers

Annotation of Commands

- An annotated command is a command with statements (*assertions*) embedded within it

- A command is *properly annotated* if statements have been inserted at the following places

  (i) before C_2 in $C_1;C_2$ if C_2 is *not* an assignment command

  (ii) after the word `DO` in `WHILE` commands

- The inserted assertions should express the conditions one expects to hold *whenever* control reaches the point at which the assertion occurs

- Can reduce number of annotations using weakest preconditions (see later)

Annotation of Specifications

- A properly annotated specification is a specification $\{P\}C\{Q\}$ where C is a properly annotated command

- Example: To be properly annotated, assertions should be at points ① and ② of the specification below

```
{X=n}
   Y:=1; ⟵①
   WHILE X≠0 DO ⟵②
     (Y:=Y×X; X:=X-1)
{X=0 ∧ Y=n!}
```

- Suitable statements would be

at ①: $\{Y = 1 \wedge X = n\}$
at ②: $\{Y \times X! = n!\}$

Verification Condition Generation

- The VCs generated from an annotated specification $\{P\}C\{Q\}$ are obtained by considering the various possibilities for C
- We will describe it command by command using rules of the form:
- The VCs for $C(C_1, C_2)$ are
  - vc_1, ... , vc_n
  - together with the VCs for C_1 and those for C_2
- Each VC rule corresponds to either a primitive or derived rule

A VC Generation Program

- The algorithm for generating verification conditions is *recursive* on the structure of commands
- The rule just given corresponds to the recursive program clause:

  $$\text{VC } (C(C_1, C_2)) = [vc_1, \ \ldots \ , vc_n] @ (\text{VC } C_1) @ (\text{VC } C_2)$$

- The rules are chosen so that only one VC rule applies in each case
  - applying them is then purely mechanical
  - the choice is based on the syntax
  - only one rule applies in each case so VC generation is deterministic

Justification of VCs

- This process will be justified by showing that $\vdash \{P\}C\{Q\}$ if all the verification conditions can be proved
- We will prove that for any `C`
  - assuming the VCs of $\{P\}C\{Q\}$ are provable
  - then $\vdash \{P\}C\{Q\}$ is a theorem of the logic

Justification of Verification Conditions

- The argument that the verification conditions are sufficient will be by *induction* on the structure of C
- Such inductive arguments have two parts
  - show the result holds for atomic commands, i.e. assignments
  - show that when C is not an atomic command, then if the result holds for the constituent commands of C (this is called the *induction hypothesis*), then it holds also for C
- The first of these parts is called the *basis* of the induction
- The second is called the *step*
- The basis and step entail that the result holds for all commands

VC for Assignments

Assignment commands

The single verification condition generated by

$$\{P\}\ V\texttt{:=}E\ \{Q\}$$

is

$$P \Rightarrow Q\texttt{[}E/V\texttt{]}$$

- Example: The verification condition for

  $\{\texttt{X=0}\}\ \texttt{X:=X+1}\ \{\texttt{X=1}\}$

  is

  $\texttt{X=0} \Rightarrow \texttt{(X+1)=1}$

  (which is clearly true)

- Note: $Q\texttt{[}E/V\texttt{]} = \texttt{wlp("}V\texttt{:=}E\texttt{",}Q\texttt{)}$

Justification of Assignment VC

- We must show that if the VCs of $\{P\}\ V := E\ \{Q\}$ are provable then $\vdash\ \{P\}\ V := E\ \{Q\}$

- Proof:
  - Assume $\vdash\ P\ \Rightarrow\ Q[E/V]$ as it is the VC
  - From derived assignment rule it follows that $\vdash\ \{P\}\ V := E\ \{Q\}$

VCs for Conditionals

Conditionals

The verification conditions generated from

$$\{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}$$

are

(i) the verification conditions generated by

$$\{P\ \wedge\ S\}\ C_1\ \{Q\}$$

(ii) the verifications generated by

$$\{P\ \wedge\ \neg S\}\ C_2\ \{Q\}$$

- Example: The verification conditions for

  {T} IF X≥Y THEN MAX:=X ELSE MAX:=Y {MAX=max(X,Y)}

  are

  (i) the VCs for {T ∧ X≥Y} MAX:=X {MAX=max(X,Y)}

  (ii) the VCs for {T ∧ ¬(X≥Y)} MAX:=Y {MAX=max(X,Y)}

Justification for the Conditional VCs (1)

- Must show that if VCs of
  $\{P\}$ `IF` S `THEN` C_1 `ELSE` C_2 $\{Q\}$
  are provable, then
  $\vdash$ $\{P\}$ `IF` S `THEN` C_1 `ELSE` C_2 $\{Q\}$

- Proof:
  - Assume the VCs $\{P \wedge S\}\ C_1\ \{Q\}$ and $\{P \wedge \neg S\}\ C_2\ \{Q\}$
  - The inductive hypotheses tell us that if these VCs are provable then the corresponding Hoare Logic theorems are provable
  - i.e. by induction $\vdash\ \{P \wedge S\}\ C_1\ \{Q\}$ and $\vdash\ \{P \wedge \neg S\}\ C_2\ \{Q\}$
  - Hence by the conditional rule $\vdash\ \{P\}$ `IF` S `THEN` C_1 `ELSE` C_2 $\{Q\}$

Review of Annotated Sequences

- If $C_1;C_2$ is properly annotated, then either

  Case 1: it is of the form $C_1;\{R\}C_2$ and C_2 is not an assignment

  Case 2: it is of the form $C;V := E$

- And C, C_1 and C_2 are properly annotated

VCs for Sequences

Sequences

1. The verification conditions generated by

$$\{P\}\ C_1\ \{R\}\ C_2\ \{Q\}$$

(where C_2 is not an assignment) are the union of:

(a) the verification conditions generated by $\{P\}\ C_1\ \{R\}$

(b) the verifications generated by $\{R\}\ C_2\ \{Q\}$

2. The verification conditions generated by

$$\{P\}\ C; V := E\ \{Q\}$$

are the verification conditions generated by $\{P\}\ C\ \{Q[E/V]\}$

Example

- The verification conditions generated from

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

- Are those generated by

  $\{X=x \wedge Y=y\}\ R:=X;\ X:=Y\ \{(X=y \wedge Y=x)[R/Y]\}$

- This simplifies to

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

- The verification conditions generated by this are those generated by

  $\{X=x \wedge Y=y\}\ R:=X\ \{(X=y \wedge R=x)[Y/X]\}$

- Which simplifies to

  $\{X=x \wedge Y=y\}\ R:=X\ \{Y=y \wedge R=x\}$

Example Continued

- The only verification condition generated by

  $\{X{=}x \wedge Y{=}y\}\ R{:=}X\ \{Y{=}y \wedge R{=}x\}$

  is

  $X{=}x \wedge Y{=}y \Rightarrow (Y{=}y \wedge R{=}x)[X/R]$

- Which simplifies to

  $X{=}x \wedge Y{=}y \Rightarrow Y{=}y \wedge X{=}x$

- Thus the single verification condition from

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

  is

  $X{=}x \wedge Y{=}y \Rightarrow Y{=}y \wedge X{=}x$

Justification of VCs for Sequences (1)

- Case 1: If the verification conditions for

  $\{P\}\ C_1\ ;\ \{R\}\ C_2\ \{Q\}$

  are provable

- Then the verification conditions for

  $\{P\}\ C_1\ \{R\}$

  and

  $\{R\}\ C_2\ \{Q\}$

  must both be provable

- Hence by induction

  $\vdash\ \{P\}\ C_1\ \{R\}$ and $\vdash\ \{R\}\ C_2\ \{Q\}$

- Hence by the sequencing rule

  $\vdash\ \{P\}\ C_1;C_2\ \{Q\}$

Justification of VCs for Sequences (2)

- Case 2: If the verification conditions for

  $\{P\}\ C; V := E\ \{Q\}$

  are provable, then the verification conditions for

  $\{P\}\ C\ \{Q[E/V]\}$

  are also provable

- Hence by induction

  $\vdash\ \{P\}\ C\ \{Q[E/V]\}$

- Hence by the derived sequenced assignment rule

  $\vdash\ \{P\}\ C; V := E\ \{Q\}$

VCs for WHILE-Commands

- A correctly annotated specification of a WHILE-command has the form

  $\{P\}$ WHILE S DO $\{R\}$ C $\{Q\}$

- The annotation R is called an invariant

WHILE-commands

The verification conditions generated from

$$\{P\} \text{ WHILE } S \text{ DO } \{R\}\ C\ \{Q\}$$

are

(i) $P \Rightarrow R$

(ii) $R \wedge \neg S \Rightarrow Q$

(iii) the verification conditions generated by $\{R \wedge S\}\ C\{R\}$

Example

- The verification conditions for

$\{R=X \wedge Q=0\}$
WHILE $Y \leq R$ DO $\{X=R+Y \times Q\}$
(R:=R-Y; Q:=Q+1)
$\{X = R+(Y \times Q) \wedge R<Y\}$
are:

(i) $R=X \wedge Q=0 \Rightarrow (X = R+(Y \times Q))$

(ii) $X = R+Y \times Q \wedge \neg(Y \leq R) \Rightarrow (X = R+(Y \times Q) \wedge R<Y)$

together with the verification condition for

$\{X = R+(Y \times Q) \wedge (Y \leq R)\}$
(R:=R-Y; Q:=Q+1)
$\{X=R+(Y \times Q)\}$
which consists of the single condition

(iii) $X = R+(Y \times Q) \wedge (Y \leq R) \Rightarrow X = (R-Y)+(Y \times (Q+1))$

Example Summarised

- By previous transparency

$$\vdash \{R{=}X \wedge Q{=}0\}$$
$$\texttt{WHILE } Y \leq R \texttt{ DO}$$
$$\texttt{(R:=R-Y; Q:=Q+1)}$$
$$\{X = R+(Y \times Q) \wedge R<Y\}$$

if

$$\vdash R{=}X \wedge Q{=}0 \Rightarrow (X = R+(Y \times Q))$$

and

$$\vdash X = R+(Y \times Q) \wedge \neg(Y \leq R) \Rightarrow (X = R+(Y \times Q) \wedge R<Y)$$

and

$$\vdash X = R+(Y \times Q) \wedge (Y \leq R) \Rightarrow X = (R-Y)+(Y \times (Q+1))$$

Justification of WHILE VCs

- If the verification conditions for

  $\{P\}$ WHILE S DO $\{R\}$ C $\{Q\}$

  are provable, then

  $\vdash\ P \Rightarrow R$

  $\vdash\ (R\ \wedge\ \neg S)\ \Rightarrow\ Q$

  and the verification conditions for

  $\{R\ \wedge\ S\}\ C\ \{R\}$

  are provable

- By induction

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

- Hence by the derived WHILE-rule

  $\vdash\ \{P\}$ WHILE S DO C $\{Q\}$

Summary

- Have outlined the design of an automated program verifier
- Annotated specifications compiled to mathematical statements
  - if the statements (VCs) can be proved, the program is verified
- Human help is required to give the annotations and prove the VCs
- The algorithm was justified by an inductive proof
  - it appeals to the derived rules
- All the techniques introduced earlier are used
  - backwards proof
  - derived rules
  - annotation