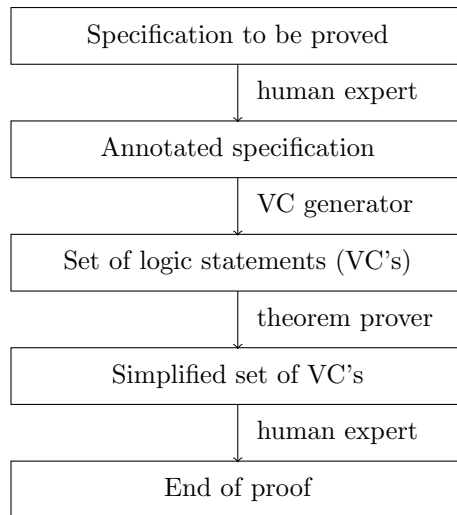


## 4 Verification

### 4.1 Verification Conditions

#### Architecture of a Verifier



#### Verification Flow

- Input: a partial correctness specification annotated with mathematical statements
  - these annotations describe relationships between variables
- The system generates a set of purely mathematical statements called *verification conditions* (or VC's)
- If the verification conditions are provable, then the original specification can be deduced from the axioms and rules of Floyd-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 into it statements (called *assertions*) expressing conditions that are meant to hold at various intermediate points
    - this step is tricky and needs intelligence and a good understanding of how the program works
    - automating it is an artificial intelligence problem
  2. A set of logic statements called *verification conditions* (VC's for short) 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)
- Step 2 converts a verification problem into a conventional mathematical problem

**Example**

- The process will be illustrated with:

```

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

```

- Step 1 is to insert annotations  $P_1$  and  $P_2$ :

```

{T}
BEGIN
  R := X;
  Q := 0; {R = X ∧ Q = 0} ← P1
  WHILE Y ≤ R DO {X = R + Y × Q} ← P2
    BEGIN R := R - Y; Q := Q + 1 END
  END
  {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)**

- 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  $X = R + Y \times Q$  holds, even though the values of R and Q vary
  - $P_2$  is an *invariant* of the WHILE command
- Step 2 will generate the following four verification conditions:

```

T ⇒ (X = X ∧ 0 = 0)
(R = X ∧ Q = 0) ⇒ (X = R + (Y × Q))
(X = R + (Y × Q)) ∧ Y ≤ R ⇒ (X = (R - Y) + (Y × (Q + 1)))
(X = R + (Y × Q)) ∧ ¬(Y ≤ R) ⇒ (X = R + (Y × Q) ∧ 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

## 4.2 Annotations

### Annotation of Commands

- The sequencing rule introduces a new statement  $R$ :

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

- To apply this rule, one needs to come up with a suitable statement  $R$
- If the second command is an assignment, the sequenced assignment rule can be used
- Similarly, to use the derived WHILE rule, we must invent an invariant.
- A command is *properly annotated* if statements (*assertions*) have been inserted at the following places:
  1. before each command  $C_i$  (where  $i > 1$ ) in a sequence  $C_1; C_2; \dots; C_n$  which is not an assignment command
  2. 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

### 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  $\boxed{1}$  and  $\boxed{2}$  of the specification below:

```
{X = n}
BEGIN
  Y := 1; ←  $\boxed{1}$ 
  WHILE X ≠ 0 DO ←  $\boxed{2}$ 
    BEGIN Y := Y × X; X := X - 1 END
  END
{X = 0 ∧ Y = n!}
```

- Suitable statements would be:

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

## 4.3 Verification Conditions for Commands

### Verification Condition Generation

- The verification conditions generated from an annotated specification  $\{P\} C \{Q\}$  are described 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
  - $vc_1 \dots vc_n$
  - together with the VCs for  $C_1$  and those for  $C_2$
- Each VC rule will correspond to either a primitive or derived rule of the logic

**VC for SKIP**

**The SKIP Command**

The single verification condition generated by:

$$\{P\} \text{ SKIP } \{Q\}$$

is:

$$P \Rightarrow Q$$

- Example: The verification condition for:

$$\{X = 0\} \text{ SKIP } \{X = 0\}$$

is:

$$X = 0 \Rightarrow X = 0$$

(which is clearly true)

**VC for Assignments**

**Assignment Commands**

The single verification condition generated by:

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

is:

$$P \Rightarrow Q[E/V]$$

- Example: The verification condition for:

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

is:

$$X = 0 \Rightarrow (X + 1) = 1$$

(which is clearly true)

**VCs for Conditional**

**Conditional**

The verification conditions generated from:

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

are:

1. The verification conditions generated by:
 
$$\{P \wedge S\} C_1 \{Q\}$$
2. The verification conditions generated by:
 
$$\{P \wedge \neg S\} C_2 \{Q\}$$

**Example**

- The verification conditions for:

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

are:

1. the VC for:  
 $\{T \wedge X \geq Y\} MAX := X \{MAX = \max(X, Y)\}$   
 which is:  
 $T \wedge X \geq Y \Rightarrow X = \max(X, Y)$
2. the VC for:  
 $\{T \wedge \neg(X \geq Y)\} MAX := Y \{MAX = \max(X, Y)\}$   
 which is:  
 $T \wedge \neg(X \geq Y) \Rightarrow Y = \max(X, Y)$

**Annotated Sequences**

- If  $C_1; \dots; C_n$  is properly annotated, then it must be of one of the two forms:

$C_1; \dots; C_{n-1}; \{R\} C_n$

or:

$C_1; \dots; C_{n-1}; V := E$

where:

$C_1; \dots; C_{n-1}$

is a properly annotated command

**VCs for Sequences****Sequences**

1. The verification conditions generated by:

$$\{P\} C_1; \dots; C_{n-1}; \{R\} C_n \{Q\}$$

(where  $C_n$  is not an assignment) are:

- (a) The verification conditions generated by:

$$\{P\} C_1; \dots; C_{n-1} \{R\}$$

- (b) The verification conditions generated by:

$$\{R\} C_n \{Q\}$$

2. The verification conditions generated by:

$$\{P\} C_1; \dots; C_{n-1}; V := E \{Q\}$$

are the verification conditions generated by:

$$\{P\} C_1; \dots; C_{n-1} \{Q[E/V]\}$$

**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**

- 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:  
 $X = x \wedge Y = y \Rightarrow Y = y \wedge X = x$

**VCs for Blocks**

Blocks

The verification conditions generated by:

$$\{P\} \text{ BEGIN VAR } V_1; \dots; \text{ VAR } V_n; C \text{ END } \{Q\}$$

are:

1. The verification conditions generated by  $\{P\} C \{Q\}$
2. The syntactic condition that none of  $V_1 \dots V_n$  occur in either  $P$  or  $Q$

- Generating verification conditions from blocks involves checking a syntactic condition that the local variables of the block do not occur in the precondition or postcondition

**Example**

- The verification conditions for:  
 $\{X = x \wedge Y = y\}$   
 BEGIN VAR R; R := X; X := Y; Y := R END  
 $\{X = y \wedge Y = x\}$
- Are those generated by:  
 $\{X = x \wedge Y = y\} R := X; X := Y; Y := R \{X = y \wedge Y = x\}$   
 Since R does not occur in  $\{X = x \wedge Y = y\}$  or  $\{X = y \wedge Y = x\}$
- See previous example for verification conditions generated by this

**VCs for WHILE Commands**

- A correctly annotated specification of a WHILE command has the form:  
 $\{P\} \text{ WHILE } S \text{ DO } \{R\} C \{Q\}$
- The annotation  $R$  is called an *invariant*

WHILE Commands

The verification conditions generated by:

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

are:

1.  $P \Rightarrow R$
2.  $R \wedge \neg S \Rightarrow Q$
3. 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  $\{X = R + Y \times Q\}$   
 BEGIN  $R := R - Y; Q := Q + 1$  END  
 $\{X = R + (Y \times Q) \wedge R < Y\}$   
 are:
- 1.  $R = X \wedge Q = 0 \Rightarrow (X = R + (Y \times Q))$
- 2.  $X = R + Y \times Q \wedge \neg(Y \leq R) \Rightarrow (X = R + (Y \times Q) \wedge R < Y)$   
 Together with the verification condition for:  
 $\{X = R + Y \times Q \wedge (Y \leq R)\}$   
 BEGIN  $R := R - Y; Q := Q + 1$  END  
 $\{X = R + (Y \times Q)\}$   
 Which consists of the single condition:
- 3.  $X = R + Y \times Q \wedge (Y \leq R) \Rightarrow X = (R - Y) + (Y \times (Q + 1))$

## 4.4 Invariants

### Finding Invariants

#### The WHILE Rule

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

- Look at the facts:
  - it must hold initially
  - with the negated test it must establish the result
  - the body must leave it 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*
  - gives the *desired result*

### Example

- Consider a factorial program
 

```
{X = n ∧ Y = 1}
WHILE X ≠ 0 DO
  BEGIN Y := Y × X; X := X - 1 END
{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  $X! \times Y = n!$ 
  - decrease in  $X$  combines with increase in  $Y$  to make invariant

**Related Example**

```

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

```

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

**4.5 Combining Steps****Conjunction and Disjunction**

<p><b>Specification Conjunction</b></p> $\frac{\vdash \{P_1\} C \{Q_1\}, \quad \vdash \{P_2\} C \{Q_2\}}{\vdash \{P_1 \wedge P_2\} C \{Q_1 \wedge Q_2\}}$ <p><b>Specification Disjunction</b></p> $\frac{\vdash \{P_1\} C \{Q_1\}, \quad \vdash \{P_2\} C \{Q_2\}}{\vdash \{P_1 \vee P_2\} C \{Q_1 \vee Q_2\}}$
---

- These rules are useful for splitting a proof into independent bits
  - they enable  $\vdash \{P\} C \{Q_1 \wedge Q_2\}$  to be proved by proving separately that both  $\vdash \{P\} C \{Q_1\}$  and also that  $\vdash \{P\} C \{Q_2\}$
- Any proof with these rules could be done without them

**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:

$$\begin{aligned} & \vdash \{T\} R := X \{R = X\} \\ = & \quad \{ \text{precondition strengthening, assignment axiom} \} \\ & \text{True} \end{aligned}$$

Rather than:

$$\begin{aligned} & \vdash \{T\} R := X \{R = X\} \\ = & \quad \{ \text{precondition strengthening, } \vdash T \Rightarrow X = X \} \\ & \vdash \{X = X\} R := X \{R = X\} \\ = & \quad \{ \text{assignment axiom} \} \\ & \text{True} \end{aligned}$$

## 4.6 Derived Assignment

### Alternative Rule For Assignment

- Here is a rule that combines the two steps:

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

- How do we know this is consistent with the assignment axiom?
- Is it more powerful (i.e. proves more) than the assignment axiom?
- Rather than add the rule as a new primitive, we can derive it
- Start with a small set of simple primitive rules
- Then derive other more complex rules from the primitives
- Rules for defined commands derived in a similar way

### The Derived Assignment Rule

- Here is another example proof similar to the earlier one:

$$\begin{aligned} & \vdash \{R = X\} Q := 0 \{R = X \wedge Q = 0\} \\ = & \quad \{ \text{precondition strengthening,} \\ & \quad \vdash R = X \Rightarrow R = X \wedge 0 = 0 \} \\ & \vdash \{R = X \wedge 0 = 0\} Q := 0 \{R = X \wedge Q = 0\} \\ = & \quad \{ \text{assignment axiom} \} \\ & \text{True.} \end{aligned}$$

- We can generalise this proof to a proof schema:

$$\begin{aligned} & \vdash \{P\} V := E \{Q\} \\ = & \quad \{ \text{precondition strengthening, } \vdash P \Rightarrow Q[E/V] \} \\ & \vdash \{Q[E/V]\} V := E \{Q\} \\ = & \quad \{ \text{assignment axiom.} \} \\ & \text{True} \end{aligned}$$

## The Derived Assignment Rule

- This proof schema justifies:

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

- The previous proof can now be done in one less step:

$$\begin{aligned} & \vdash \{R = X\} Q := 0 \{R = X \wedge Q = 0\} \\ = & \quad \{ \text{derived assignment, } \vdash R = X \Rightarrow R = X \wedge 0 = 0 \} \\ & \text{True} \end{aligned}$$

## 4.7 Other Derived Rules

### Rules of Consequence

- As in the assignment example, the desired precondition and postcondition are rarely in the form required by the primitive rules
- Ideally, for each command we want a rule of the form:

$$\frac{\dots}{\vdash \{P\} C \{Q\}}$$

- where  $P$  and  $Q$  are distinct meta-variables.
- Some of the rules are already in this form e.g. the sequencing rule
- We can derive rules of this form for the other commands using the rules of consequence

### The Derived SKIP Rule

$$\frac{\vdash P \Rightarrow Q}{\vdash \{P\} \text{SKIP} \{Q\}}$$

- Justifying proof schema:

$$\begin{aligned} & \vdash \{P\} \text{SKIP} \{Q\} \\ = & \quad \{ \text{precondition strengthening,} \\ & \quad \vdash P \Rightarrow Q \text{ (by assumption)} \} \\ & \vdash \{Q\} \text{SKIP} \{Q\} \\ = & \quad \{ \text{SKIP axiom.} \} \\ & \text{True} \end{aligned}$$

### The Derived WHILE Rule

$$\begin{array}{c}
 \textbf{The Derived WHILE Rule} \\
 \hline
 \frac{\vdash P \Rightarrow R, \quad \vdash \{R \wedge S\} C \{R\}, \quad \vdash R \wedge \neg S \Rightarrow Q}{\vdash \{P\} \text{ WHILE } S \text{ DO } C \{Q\}}
 \end{array}$$

- This follows from the WHILE rule and the laws of consequence

### Example

$$\begin{array}{l}
 \vdash \{R = X \wedge Q = 0\} \\
 \quad \text{WHILE } Y \leq R \text{ DO} \\
 \quad \quad R := R - Y; Q := Q + 1 \\
 \quad \{X = R + (Y \times Q) \wedge \neg(Y \leq R)\} \\
 = \\
 \quad \{ \text{derived WHILE rule,} \\
 \quad \quad \vdash R = X \wedge Q = 0 \Rightarrow X = R + (Y \times Q), \\
 \quad \quad \vdash X = R + (Y \times Q) \wedge \neg(Y \leq R) \Rightarrow \\
 \quad \quad \quad X = R + (Y \times Q) \wedge \neg(Y \leq R) \} \\
 \vdash \{X = R + (Y \times Q) \wedge Y \leq R\} \\
 \quad R := R - Y; Q := Q + 1 \\
 \quad \{X = R + (Y \times Q)\} \\
 = \\
 \quad \{ \text{sequencing rule, derived assignment} \} \\
 \text{True}
 \end{array}$$

### The Derived Sequencing Law

- The following rule is derivable from the sequencing and consequence rules:

$$\begin{array}{c}
 \textbf{The Derived Sequencing Rule} \\
 \hline
 \begin{array}{l}
 \vdash \{P_1\} C_1 \{Q_1\} \quad \vdash P \Rightarrow P_1 \\
 \vdash \{P_2\} C_2 \{Q_2\} \quad \vdash Q_1 \Rightarrow P_2 \\
 \vdots \quad \vdots \\
 \vdash \{P_n\} C_n \{Q_n\} \quad \vdash Q_{n-1} \Rightarrow P_n \\
 \vdash \{P_n\} C_n \{Q_n\} \quad \vdash Q_n \Rightarrow Q
 \end{array} \\
 \hline
 \vdash \{P\} C_1; \dots; C_n \{Q\}
 \end{array}$$

### Example

- By the assignment axiom:
  1.  $\{X = x \wedge Y = y\} R := X \{R = x \wedge Y = y\}$
  2.  $\{R = x \wedge Y = y\} X := Y \{R = x \wedge X = y\}$
  3.  $\{R = x \wedge X = y\} Y := R \{Y = x \wedge X = y\}$
- Using the derived sequencing rule, it can be deduced in one step from 1, 2, 3 that:
 
$$\vdash \{X = x \wedge Y = y\} \quad R := X; X := Y; Y := R \quad \{Y = x \wedge X = y\}$$

### The Derived Block Rule

- From the derived sequencing rule and the block rule the following rule for blocks can be derived:

<b>The Derived Block Rule</b>	
$\vdash \{P_1\} C_1 \{Q_1\}$	$\vdash P \Rightarrow P_1$
$\vdash \{P_2\} C_2 \{Q_2\}$	$\vdash Q_1 \Rightarrow P_2$
$\vdots$	$\vdots$
$\vdash \{P_n\} C_n \{Q_n\}$	$\vdash Q_n \Rightarrow Q$
$\vdash \{P\} \text{ BEGIN VAR } V_1; \dots; \text{ VAR } V_m; C_1; \dots; C_n \text{ END } \{Q\}$	
where none of the variables $V_1 \dots V_m$ occur in $P$ or $Q$ .	

### Example

- By the assignment axiom
  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.  $\vdash \{R = x \wedge X = y\} Y := R \{Y = x \wedge X = y\}$
- Using the derived block rule, it can be deduced in one step from 1, 2 and 3 that:
 
$$\vdash \{X = x \wedge Y = y\}$$

$$\text{ BEGIN VAR } R; R := X; X := Y; Y := R \text{ END}$$

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

### Sequenced Assignment

<b>The Derived Sequenced Assignment Rule</b>
$\vdash \{P\} C \{Q[E/V]\}$
$\vdash \{P\} C; V := E \{Q\}$

- Exercise: give a proof schema to justify this
- Intuitively work backwards:
  - in rule conclusion push  $Q$  ‘through’  $V := E$
  - changing it to  $Q[E/V]$
- Example: By the assignment axiom:
 
$$\vdash \{X = x \wedge Y = y\} R := X \{R = x \wedge Y = y\}$$
- Hence by the sequenced assignment rule:
 
$$\vdash \{X = x \wedge Y = y\} R := X; X := Y \{R = x \wedge X = y\}$$

## 4.8 Deriving Rules for New Commands

### Deriving Rules For New Commands

- Suppose we define a one-armed conditional by:  
IF  $S$  THEN  $C \equiv$  IF  $S$  THEN  $C$  ELSE SKIP
- We can derive the following rule:

**The One-Armed Conditional Rule**

$$\frac{\vdash \{P \wedge S\} C \{Q\}, \quad \vdash P \wedge \neg S \Rightarrow Q}{\vdash \{P\} \text{ IF } S \text{ THEN } C \{Q\}}$$

### One-Armed Conditional

- Derivation:
  - $\vdash \{P\} \text{ IF } S \text{ THEN } C \{Q\}$
  - $=$  { definition of one-armed conditional }
  - $\vdash \{P\} \text{ IF } S \text{ THEN } C \text{ ELSE SKIP } \{Q\}$
  - $=$  { conditional rule }
  - $\vdash \{P \wedge S\} C \{Q\} \wedge \vdash \{P \wedge \neg S\} \text{ SKIP } \{Q\}$
  - $=$  {  $\vdash \{P \wedge S\} C \{Q\}$  (by assumption) }
  - $\vdash \{P \wedge \neg S\} \text{ SKIP } \{Q\}$
  - $=$  { derived SKIP rule,  $\vdash P \wedge \neg S \Rightarrow Q$  }
  - True
- From:
  1.  $\vdash \{T \wedge X \geq Y\} Y := X \{Y = \max(X, Y)\}$
  2.  $\vdash T \wedge Y > X \Rightarrow \max(X, Y) = Y$
- Then by the derived one-armed conditional rule it follows that:  
 $\vdash \{T\} \text{ IF } X \geq Y \text{ THEN } Y := X \{Y = \max(X, Y)\}$

## 4.9 Forwards and Backwards Proof

### Forwards and Backwards Proof

- $\vdash \{P\} C \{Q\}$  can be proved 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 the 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  $\vdash \{P\} C \{Q\}$
  - generate subgoals until problem solved

### Example

- Suppose one wants to show:
 
$$\vdash \{X = x \wedge Y = y\}$$

$$R := X; X := Y; Y := R$$

$$\{Y = x \wedge X = y\}$$
- By the assignment axiom and derived sequenced assignment rule it is sufficient to show the subgoal:
 
$$\vdash \{X = x \wedge Y = y\} R := X; X := Y \{R = x \wedge X = y\}$$
- Similarly, this subgoal can be reduced to:
 
$$\vdash \{X = x \wedge Y = y\} R := X \{R = x \wedge Y = y\}$$
- This clearly follows from the assignment axiom.

### Backwards Versus Forwards Proof

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

$$\frac{\vdash S_1}{\vdash S_2}$$

- Forwards proof says: if we have proved  $\vdash S_1$  we can deduce  $\vdash S_2$
- Backwards proof says: to prove  $\vdash S_2$  it is sufficient to prove  $\vdash S_1$
- Having proved a theorem by backwards proof, it is simple to extract a forwards proof

### Example Backwards Proof

- To prove:
 
$$\vdash \{T\}$$

$$R := X;$$

$$Q := 0;$$

$$\text{WHILE } Y \leq R \text{ DO}$$

$$\quad \text{BEGIN } R := R - Y; Q := Q + 1 \text{ END}$$

$$\{X = R + (Y \times Q) \wedge R < Y\}$$
- By the sequencing rule, it is sufficient to prove:
  1.  $\vdash \{T\} R := X; Q := 0 \{R = X \wedge Q = 0\}$
  2.  $\vdash \{R = X \wedge Q = 0\}$ 

$$\text{WHILE } Y \leq R \text{ DO}$$

$$\quad \text{BEGIN } R := R - Y; Q := Q + 1 \text{ END}$$

$$\{X = R + (Y \times Q) \wedge R < Y\}$$
- To prove 1, by the sequenced assignment axiom, we must prove:
  3.  $\vdash \{T\} R := X \{R = X \wedge 0 = 0\}$

**Example Continued**

- To prove 3, by the derived assignment rule, we must prove:

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

- This is true by pure logic.

- To prove 2, by the derived WHILE rule, we must prove:

$$4. R = X \wedge Q = 0 \Rightarrow (X = R + (Y \times Q))$$

$$5. X = R + Y \times Q \wedge \neg(Y \leq R) \Rightarrow$$

$$(X = R + (Y \times Q) \wedge R < Y)$$

and

$$6. \{X = R + (Y \times Q) \wedge (Y \leq R)\}$$

BEGIN R := R - Y; Q := Q + 1 END

$$\{X = R + (Y \times Q)\}$$

- 4 and 5 are proved by arithmetic

**Example Continued**

- To prove 6, by the block rule, we must prove:

$$7. \{X = R + (Y \times Q) \wedge (Y \leq R)\}$$

R := R - Y; Q := Q + 1

$$\{X = R + (Y \times Q)\}$$

- To prove 7, by the sequenced assignment rule, we must prove:

$$8. \{X = R + (Y \times Q) \wedge (Y \leq R)\}$$

R := R - Y

$$\{X = R + (Y \times (Q + 1))\}$$

- To prove 8, by the derived assignment rule, we must prove:

$$9. X = R + (Y \times Q) \wedge Y \leq R \Rightarrow$$

$$(X = (R - Y) + (Y \times (Q + 1)))$$

- This is true by arithmetic

- Exercise: Construct the forwards proof that corresponds to this backwards proof