4 Verification

4.1 Verification Conditions

Architecture of a Verifier

<table>
<thead>
<tr>
<th>Specification to be proved</th>
<th>human expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotated specification</td>
<td>VC generator</td>
</tr>
<tr>
<td>Set of logic statements (VC’s)</td>
<td>theorem prover</td>
</tr>
<tr>
<td>Simplified set of VC’s</td>
<td>human expert</td>
</tr>
<tr>
<td>End of proof</td>
<td></td>
</tr>
</tbody>
</table>

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:

  \[
  \{T\}
  \]
  \[
  \text{BEGIN}\]
  \[
  R := X; \quad Q := 0; \quad \text{WHILE } Y \leq R \text{ DO}
  \]
  \[
  \text{BEGIN } R := R - Y; \quad Q := Q + 1 \text{ END}\]
  \[
  \text{END}\]
  \[
  \{X = R + Y \times Q \land R < Y\}\]

- Step 1 is to insert annotations \(P_1\) and \(P_2\):

  \[
  \{T\}
  \]
  \[
  \text{BEGIN}\]
  \[
  R := X; \quad Q := 0; \quad \{R = X \land Q = 0\} \leftarrow P_1
  \]
  \[
  \text{WHILE } Y \leq R \text{ DO } \{X = R + Y \times Q\} \leftarrow P_2
  \]
  \[
  \text{BEGIN } R := R - Y; \quad Q := Q + 1 \text{ END}\]
  \[
  \text{END}\]
  \[
  \{X = R + Y \times Q \land R < Y\}\]

- 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:

  \[
  T \Rightarrow (X = X \land 0 = 0)
  \]
  \[
  (R = X \land Q = 0) \Rightarrow (X = R + (Y \times Q))
  \]
  \[
  (X = R + (Y \times Q)) \land Y \leq R \Rightarrow (X = R - Y + (Y \times (Q + 1)))
  \]
  \[
  (X = R + (Y \times Q)) \land \neg(Y \leq R) \Rightarrow (X = R + (Y \times Q) \land R < Y)
  \]
- 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 $R$:

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

- 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 $C_1; C_2; \ldots; C_n$ 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 $\text{[1]}$ and $\text{[2]}$ of the specification below:

  $$\{X = n\}$$

  BEGIN
  $\quad Y := 1; \leftarrow \text{[1]}$
  $\quad$ WHILE $X \neq 0$ DO $\leftarrow \text{[2]}$
  $\quad$ BEGIN $Y := Y \times X; X := X - 1$ END
  END
  $$\{X = 0 \land Y = n!\}$$

- Suitable statements would be:
  - at $\text{[1]}$ $\vdash \{Y = 1 \land X = n\}$
  - at $\text{[2]}$ $\vdash Y \times X! = n!$

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
    - $vc_1 \ldots vc_n$
    - 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:

\( \{P \land S\} C_1 \{Q\} \)

2. The verification conditions generated by:

\( \{P \land \neg S\} C_2 \{Q\} \)
Example

- The verification conditions for:

\[
\{ T \}
\]

\[
\text{IF X} \geq \text{Y THEN MAX := X ELSE MAX := Y}
\]

\[
\{ \text{MAX} = \max(X, Y) \}
\]

are:

1. the VC for:

\[
\{ T \land X \geq Y \} \text{ MAX := X } \{ \text{MAX} = \max(X, Y) \}
\]

which is:

\[
T \land X \geq Y \Rightarrow X = \max(X, Y)
\]

2. the VC for:

\[
\{ T \land \neg(X \geq Y) \} \text{ MAX := Y } \{ \text{MAX} = \max(X, Y) \}
\]

which is:

\[
T \land \neg(X \geq Y) \Rightarrow Y = \max(X, Y)
\]

Annotated Sequences

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

\[
C_1; \ldots; C_{n-1}; \{ R \} C_n
\]

or:

\[
C_1; \ldots; C_{n-1}; V := E
\]

where:

\[
C_1; \ldots; C_{n-1}
\]

is a properly annotated command

VCs for Sequences

<table>
<thead>
<tr>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The verification conditions generated by:</td>
</tr>
</tbody>
</table>
| \[
\{ P \} C_1; \ldots; C_{n-1}; \{ R \} C_n \{ Q \}
\] |
| (where \( C_n \) is not an assignment) are: |
| (a) The verification conditions generated by: |
| \[
\{ P \} C_1; \ldots; C_{n-1} \{ R \}
\] |
| (b) The verification conditions generated by: |
| \[
\{ R \} C_n \{ Q \}
\] |
| 2. The verification conditions generated by: |
| \[
\{ P \} C_1; \ldots; C_{n-1}; V := E \{ Q \}
\] |
| are the verification conditions generated by: |
| \[
\{ P \} C_1; \ldots; C_{n-1} \{ Q[E/V] \}
\] |
Example

- The verification conditions generated from:
  \{X = x \land Y = y\} \quad R := X; X := Y; Y := R \quad \{X = y \land Y = x\}

- Are those generated by:
  \{X = x \land Y = y\} \quad R := X; X := Y \quad \{(X = y \land Y = x)[R/Y]\}

- This simplifies to:
  \{X = x \land Y = y\} \quad R := X; X := Y \quad \{X = y \land R = x\}

- The verification conditions generated by this are those generated by:
  \{X = x \land Y = y\} \quad R := X \quad \{(X = y \land R = x)[Y/X]\}

- Which simplifies to:
  \{X = x \land Y = y\} \quad R := X \quad \{Y = y \land R = x\}

Example

- The only verification condition generated by:
  \{X = x \land Y = y\} \quad R := X \quad \{Y = y \land R = x\}
  is:
  \quad \quad X = x \land Y = y \Rightarrow (Y = y \land R = x)[X/R]

- Which simplifies to:
  \quad \quad X = x \land Y = y \Rightarrow Y = y \land X = x

- Thus the single verification condition from:
  \{X = x \land Y = y\} \quad R := X; X := Y; Y := R \quad \{X = y \land Y = x\}
  is:
  \quad \quad X = x \land Y = y \Rightarrow Y = y \land X = x

VCs for Blocks

<table>
<thead>
<tr>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>The verification conditions generated by:</td>
</tr>
<tr>
<td>{P} BEGIN VAR \textit{V}_1; \ldots; VAR \textit{V}_n; \textit{C} END {Q}</td>
</tr>
<tr>
<td>are:</td>
</tr>
<tr>
<td>1. The verification conditions generated by {P} \ C \ {Q}</td>
</tr>
<tr>
<td>2. The syntactic condition that none of \textit{V}_1 \ldots \textit{V}_n occur in either \textit{P} or \textit{Q}</td>
</tr>
</tbody>
</table>

- 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 \land Y = y\}
BEGIN VAR R; R := X; X := Y; Y := R END
\{X = y \land Y = x\}

• Are those generated by:

\{X = x \land Y = y\} R := X; X := Y; Y := R \{X = y \land Y = x\}
Since \(R\) does not occur in \(\{X = x \land Y = y\}\) or \(\{X = y \land Y = x\}\)

• 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\} WHILE S DO \{R\} C \{Q\}

• The annotation \(R\) is called an invariant

<table>
<thead>
<tr>
<th>WHILE Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>The verification conditions generated by:</td>
</tr>
<tr>
<td>{P} WHILE S DO {R} C {Q}</td>
</tr>
<tr>
<td>are:</td>
</tr>
<tr>
<td>1. (P \Rightarrow R)</td>
</tr>
<tr>
<td>2. (R \land \neg S \Rightarrow Q)</td>
</tr>
<tr>
<td>3. The verification conditions generated by ({R \land S} C {R})</td>
</tr>
</tbody>
</table>

Example

• The verification conditions for:

\{R = X \land Q = 0\}
WHILE \(Y \leq R\) DO \(\{X = R + Y \times Q\}\)
BEGIN R := R \(\rightarrow Y\); Q := Q + 1 END
\{X = R + (Y \times Q) \land R < Y\}

are:

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

Finding Invariants

The WHILE Rule

\[
\frac{\vdash \{P \land S\} \ C \ \{P\}}{\vdash \{P\} \ \text{WHILE} \ S \ \text{DO} \ C \ \{P \land \neg S\}}
\]

- 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
  \[
  \begin{align*}
  &\{X = n \land Y = 1\} \\
  &\text{WHILE } X \neq 0 \ \text{DO} \\
  &\quad \text{BEGIN } Y := Y \times X; \ X := X - 1 \ \text{END} \\
  &\{X = 0 \land Y = n!\}
  \end{align*}
  \]

- 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 \land Y = 1\}
WHILE X < N DO
    BEGIN X := X + 1; Y := Y \times 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: \(Y = X! \land X \leq N\)
• At end \(X \leq N \land \neg(X < N) \Rightarrow X = N\)
• 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
\[
\frac{\vdash \{P_1\} C \{Q_1\}, \quad \vdash \{P_2\} C \{Q_2\}}{\vdash \{P_1 \land P_2\} C \{Q_1 \land Q_2\}}
\]
\[
\frac{\vdash \{P_1\} C \{Q_1\}, \quad \vdash \{P_2\} C \{Q_2\}}{\vdash \{P_1 \lor P_2\} C \{Q_1 \lor Q_2\}}
\]
• These rules are useful for splitting a proof into independent bits
  - they enable \(\vdash \{P\} C \{Q_1 \land Q_2\}\) 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:
\[ \vdash \{ T \} \ R := X \{ R = X \} \]
\[ = \quad \{ \text{precondition strengthening, assignment axiom} \} \]
\[ \text{True} \]
Rather than:
\[ \vdash \{ T \} \ R := X \{ R = X \} \]
\[ = \quad \{ \text{precondition strengthening, } \vdash T \Rightarrow X = X \} \]
\[ \vdash \{ X = X \} \ R := X \{ R = X \} \]
\[ = \quad \{ \text{assignment axiom} \} \]
\[ \text{True} \]

4.6 Derived Assignment

Alternative Rule For Assignment

• Here is a rule that combines the two steps:

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

• 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:
\[ \vdash \{ R = X \} \ Q := 0 \{ R = X \land Q = 0 \} \]
\[ = \quad \{ \text{precondition strengthening, } \vdash R = X \Rightarrow R = X \land 0 = 0 \} \]
\[ \vdash \{ R = X \land 0 = 0 \} \ Q := 0 \{ R = X \land Q = 0 \} \]
\[ = \quad \{ \text{assignment axiom} \} \]
\[ \text{True.} \]
• We can generalise this proof to a proof schema:
\[ \vdash \{ P \} \ V := E \{ Q \} \]
\[ = \quad \{ \text{precondition strengthening, } \vdash P \Rightarrow Q[E/V] \} \]
\[ \vdash \{ Q[E/V] \} \ V := E \{ Q \} \]
\[ = \quad \{ \text{assignment axiom} \} \]
\[ \text{True} \]
The Derived Assignment Rule

- This proof schema justifies:

\[
\begin{align*}
\vdash P & \Rightarrow Q[E/V] \\
\vdash \{P\} V := E \{Q\}
\end{align*}
\]

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

\[
\begin{align*}
\vdash \{R = X\} Q := 0 \{R = X \land Q = 0\} \\
= & \quad \{ \text{derived assignment, } \vdash R = X \Rightarrow R = X \land 0 = 0 \} \\
\end{align*}
\]

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:

\[
\begin{align*}
\vdash \ldots \\
\vdash \{P\} C \{Q\}
\end{align*}
\]

- 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 \text{SKIP} Rule

\[
\begin{align*}
\vdash P & \Rightarrow \text{SKIP} \{Q\} \\
\end{align*}
\]

- Justifying proof schema:

\[
\begin{align*}
\vdash \{P\} \text{SKIP} \{Q\} \\
= & \quad \{ \text{precondition strengthening, } \vdash P \Rightarrow Q \text{ (by assumption)} \} \\
\vdash \{Q\} \text{SKIP} \{Q\} \\
= & \quad \{ \text{SKIP axiom.} \} \\
\text{True}
\end{align*}
\]
The Derived WHILE Rule

\[
\begin{array}{c}
\vdash P \Rightarrow R, \quad \vdash \{ R \land S \} C \{ R \}, \quad \vdash R \land \neg S \Rightarrow Q \\
\vdash \{ P \} \text{WHILE } S \text{ DO } C \{ Q \}
\end{array}
\]

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

Example

\[
\vdash \{ R = X \land Q = 0 \}
\]

WHILE \( Y \leq R \) DO

\[
R := R - Y; \quad Q := Q + 1\\
\{ X = R + (Y \times Q) \land \neg (Y \leq R) \}
\]

= \{ derived WHILE rule, \}

\[
\vdash R = X \land Q = 0 \Rightarrow X = R + (Y \times Q), \\
\vdash X = R + (Y \times Q) \land \neg (Y \leq R) \Rightarrow X = R + (Y \times Q) \land \neg (Y \leq R)
\}

\[
\vdash \{ X = R + (Y \times Q) \land Y \leq R \}
\]

\[
R := R - Y; \quad Q := Q + 1\\
\{ X = R + (Y \times Q) \}
\]

= \{ sequencing rule, derived assignment \}

True

The Derived Sequencing Law

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

\[
\begin{array}{c}
\vdash P \Rightarrow P_1 \\
\vdash \{ P_1 \} C_1 \{ Q_1 \} \quad \vdash Q_1 \Rightarrow P_2 \\
\vdash \{ P_2 \} C_2 \{ Q_2 \} \quad \vdash Q_2 \Rightarrow P_3 \\
\vdots \\
\vdash \{ P_n \} C_n \{ Q_n \} \quad \vdash Q_n \Rightarrow Q \\
\vdash \{ P \} C_1; \ldots; C_n \{ Q \}
\end{array}
\]

Example

- By the assignment axiom:
  1. \( \{ X = x \land Y = y \} R := X \{ R = x \land Y = y \} \)
  2. \( \{ R = x \land Y = y \} X := Y \{ R = x \land X = y \} \)
  3. \( \{ R = x \land X = y \} Y := R \{ Y = x \land X = y \} \)

- Using the derived sequencing rule, it can be deduced in one step from 1, 2, 3 that:

\[
\vdash \{ X = x \land Y = y \} \\
R := X; \quad X := Y; \quad Y := R \quad \{ Y = x \land X = y \}
\]
The Derived Block Rule

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

\[
\begin{align*}
\vdash P &\Rightarrow P_1 \\
\vdash \{P_1\} C_1 \{Q_1\} &\Rightarrow Q_1 \Rightarrow P_2 \\
\vdash \{P_2\} C_2 \{Q_2\} &\Rightarrow Q_2 \Rightarrow P_3 \\
\vdots &\vdots \\
\vdash \{P_n\} C_n \{Q_n\} &\Rightarrow Q_n \Rightarrow Q \\
\vdash \{P\} \text{ BEGIN VAR } V_1; \ldots; V_m; C_1; \ldots; C_n \text{ END } \{Q\}
\end{align*}
\]

where none of the variables \(V_1 \ldots V_m\) occur in \(P\) or \(Q\).

Example

- By the assignment axiom
  1. \(\vdash \{X = x \land Y = y\}\) \(R := X \{R = x \land Y = y\}\)
  2. \(\vdash \{R = x \land Y = y\}\) \(X := Y \{R = x \land X = y\}\)
  3. \(\vdash \{R = x \land X = y\}\) \(Y := R \{Y = x \land X = y\}\)

- Using the derived block rule, it can be deduced in one step from 1, 2 and 3 that:
  \(\vdash \{X = x \land Y = y\}\)
  
  BEGIN VAR \(R; R := X; X := Y; Y := R\) END
  
  \(\{Y = x \land X = y\}\)

Sequenced Assignment

\[
\begin{align*}
\vdash \{P\} C \{Q[E/V]\} \\
\vdash \{P\} C; V := E \{Q\}
\end{align*}
\]

- 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:
  \(\vdash \{X = x \land Y = y\}\) \(R := X \{R = x \land Y = y\}\)

- Hence by the sequenced assignment rule:
  \(\vdash \{X = x \land Y = y\}\) \(R := X; X := Y \{R = x \land X = y\}\)
4.8 Deriving Rules for New Commands

Deriving Rules For New Commands

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

\[
\text{The One-Armed Conditional Rule} \\
\vdash \{ P \land S \} C \{ Q \}, \quad \vdash P \land \neg S \Rightarrow Q \quad \vdash \{ P \} \text{ IF } S \text{ THEN } C \{ Q \}
\]

One-Armed Conditional

- Derivation:
  \[
  \vdash \{ P \} \text{ IF } S \text{ THEN } C \{ Q \} \\
  = \quad \{ \text{definition of one-armed conditional} \} \\
  \vdash \{ P \} \text{ IF } S \text{ THEN } C \text{ ELSE SKIP } \{ Q \} \\
  = \quad \{ \text{conditional rule} \} \\
  \vdash \{ P \land S \} C \{ Q \} \land \vdash \{ P \land \neg S \} \text{ SKIP } \{ Q \} \\
  = \quad \{ \vdash \{ P \land S \} C \{ Q \} \text{ (by assumption)} \} \\
  \vdash \{ P \land \neg S \} \text{ SKIP } \{ Q \} \\
  = \quad \{ \text{derived SKIP rule, } \vdash P \land \neg S \Rightarrow Q \} \\
  \text{True}
\]

- From:
  1. \( \vdash \{ T \land X \geq Y \} \ Y := X \{ Y = \max(X,Y) \} \)
  2. \( \vdash T \land Y > X \Rightarrow \max(X,Y) = Y \)

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

- By the assignment axiom and derived sequenced assignment rule it is sufficient to show the subgoal:
  $\vdash \{X = x \land Y = y\} \ R := X; \ X := Y \ \{R = x \land X = y\}$

- Similarly, this subgoal can be reduced to:
  $\vdash \{X = x \land Y = y\} \ R := X \ \{R = x \land Y = y\}$

- This clearly follows from the assignment axiom.

Backwards Versus Forwards Proof
- Backwards proof just involves using the rules backwards
- Given the rule:
  $\vdash S_1$
  $\vdash S_2$

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

- By the sequencing rule, it is sufficient to prove:
  1. $\vdash \{T\} \ R := X; \ Q := 0 \ \{R = X \land Q = 0\}$
  2. $\vdash \{R = X \land Q = 0\}$
     $\text{WHILE } Y \leq R \text{ DO}$
     $\quad \text{BEGIN } R := R - Y; \ Q := Q + 1 \text{ END}$
     $\quad \{X = R + (Y \times Q) \land R < Y\}$

- To prove 1, by the sequenced assignment axiom, we must prove:
  3. $\vdash \{T\} \ R := X \ \{R = X \land 0 = 0\}$
Example Continued

- To prove 3, by the derived assignment rule, we must prove:
  \[ \vdash T \Rightarrow X = X \land 0 = 0 \]
  This is true by pure logic.

- To prove 2, by the derived \texttt{WHILE} rule, we must prove:
  4. \[ R = X \land Q = 0 \Rightarrow (X = R + (Y \times Q)) \]
  5. \[ X = R + Y \times Q \land \neg (Y \leq R) \Rightarrow \]
  \[ (X = R + (Y \times Q) \land R < Y) \]
  and
  6. \{X = R + (Y \times Q) \land (Y \leq R)}
  BEGIN \texttt{R := R} \texttt{- 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) \land (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) \land (Y \leq R)}
  \[ R := R - Y \]
  \{X = R + (Y \times (Q + 1))}\}

- To prove 8, by the derived assignment rule, we must prove:
  9. \[ X = R + (Y \times Q) \land Y \leq R \Rightarrow \]
  \[ (X = (R - Y) + (Y \times (Q + 1))) \]
  This is true by arithmetic

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