- Syntax: V := E
- Semantics: value of V in final state is value of E in initial state
- Example: X:=X+1 (adds one to the value of the variable X)

The Assignment Axiom

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

Where V is any variable, E is any expression, Q is any statement.

- Instances of the assignment axiom are
  - $\bullet \hspace{0.1in} \vdash \hspace{0.1in} \{E=x\} \hspace{0.1in} V:=E \hspace{0.1in} \{V=x\}$
  - $\bullet \hspace{0.1in} \vdash \hspace{0.1in} \left\{ Y=2 \right\} \hspace{0.1in} X:=2 \hspace{0.1in} \left\{ Y=X \right\}$
  - $\bullet \ \vdash \ \big\{ X+1=n+1 \big\} \ X:=X+1 \ \big\{ X=n+1 \big\}$
  - $\vdash \{E = E\} \ X := E \ \{X = E\}$  (if X does not occur in E)

• Recall that

$$\frac{\vdash S_1, \ \dots, \ \vdash \ S_n}{\vdash \ S}$$

means  $\vdash S$  can be deduced from  $\vdash S_1, \ldots, \vdash S_n$ 

• Using this notation, the rule of precondition strengthening is

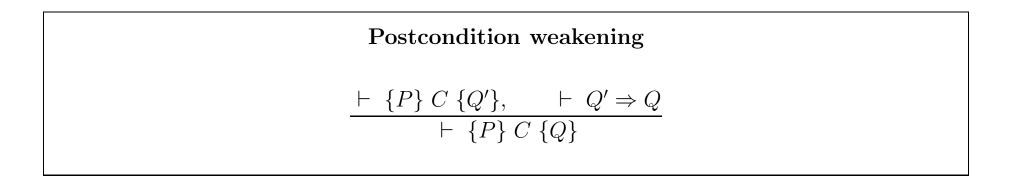
Precondition strengthening

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

• Note the two hypotheses are different kinds of judgements

**Postcondition weakening** 

• Just as the previous rule allows the precondition of a partial correctness specification to be strengthened, the following one allows us to weaken the postcondition



# An Example Formal Proof

• Here is a little formal proof

- 1.  $\vdash$  {R=X  $\land$  0=0} Q:=0 {R=X  $\land$  Q=0} By the assignment axiom
- 2.  $\vdash$  R=X  $\Rightarrow$  R=X  $\land$  0=0 By pure logic
- 3.  $\vdash$  {R=X} Q:=0 {R=X  $\land$  Q=0} By precondition strengthening
- 4.  $\vdash R=X \land Q=0 \implies R=X+(Y \times Q)$  By laws of arithmetic
- 5.  $\vdash$  {R=X} Q:=0 {R=X+(Y × Q)} By postcondition weakening
- The rules precondition strengthening and postcondition weakening are sometimes called the *rules* of consequence

# The sequencing rule

- Syntax:  $C_1$ ;  $\cdots$ ;  $C_n$
- Semantics: the commands  $C_1, \dots, C_n$  are executed in that order
- Example: R:=X; X:=Y; Y:=R
  - the values of X and Y are swapped using R as a temporary variable
  - note side effect: value of R changed to the old value of X

The sequencing rule  

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

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

# **Example Proof**

**Example:** By the assignment axiom:

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

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

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

Hence by (i), (ii) and the sequencing rule

(iv) 
$$\vdash$$
 {X=x $\land$ Y=y} R:=X; X:=Y {R=x $\land$ X=y}

Hence by (iv) and (iii) and the sequencing rule

 $(v) \vdash \{X=x \land Y=y\} R:=X; X:=Y; Y:=R \{Y=x \land X=y\}$ 

# 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

$$\vdash \{P \land S\} C_1 \{Q\}, \qquad \vdash \{P \land \neg S\} C_2 \{Q\}$$
$$\vdash \{P\} \text{ IF } S \text{ THEN } C_1 \text{ ELSE } C_2 \{Q\}$$

• From Assignment Axiom + Precondition Strengthening and

$$\vdash (X \ge Y \implies X = \max(X,Y)) \land (\neg(X \ge Y) \implies Y = \max(X,Y))$$

it follows that

$$\vdash \{T \land X \ge Y\} MAX := X \{MAX = max(X,Y)\}$$

and

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

• 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 <code>WHILE-command</code> 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
  - $\bullet\,$  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  $\vdash \{P \land S\} \subset \{P\}$
- 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



 $\begin{array}{c|c} \vdash & \{P \land S\} \ C \ \{P\} \\ \hline \vdash & \{P\} \ \texttt{WHILE} \ S \ \texttt{DO} \ C \ \{P \land \neg S\} \end{array}$ 

- It is easy to show
  - $\vdash \{X=R+(Y\times Q)\wedge Y\leq R\} R:=R-Y; Q:=Q+1 \{X=R+(Y\times Q)\}$
- Hence by the WHILE-rule with  $P = 'X=R+(Y \times Q)'$  and  $S = 'Y \leq R'$

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

## Example

• From the previous slide

```
 \vdash \{X=R+(Y\times Q)\} \\ \text{WHILE } Y \leq R \text{ DO} \\ (R:=R-Y; Q:=Q+1) \\ \{X=R+(Y\times Q) \land \neg (Y \leq R)\}
```

• It is easy to deduce that

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

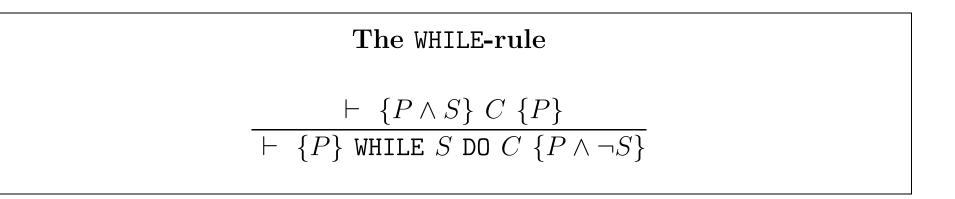
• Hence by the sequencing rule and postcondition weakening

```
⊢ {T}
R:=X;
Q:=0;
WHILE Y≤R D0
(R:=R-Y; Q:=Q+1)
{R<Y ∧ X=R+(Y×Q)}</pre>
```

#### 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?



- 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 \land Y=1 \} \\ \text{WHILE } X \neq 0 \text{ DO} \\ (Y:=Y \times X; X:=X-1) \\ \{ X=0 \land Y=n! \}
```

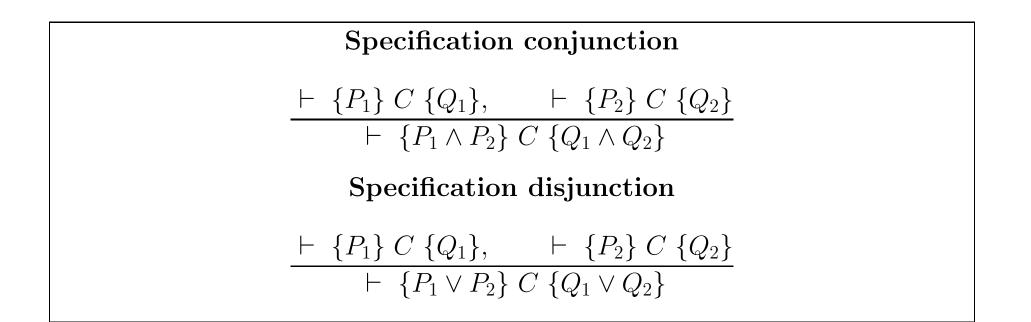
- Look at the facts
  - initially X=n and Y=1
  - finally X=0 and Y=n!
  - on each loop Y is increased and, X is decreased
- Think how the loop works
  - Y holds the result so far
  - X! is what remains to be computed
  - n! is the desired result
- The invariant is  $X! \times Y = n!$ 
  - 'stuff to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

## **Related example**

$$\begin{array}{l} {X=0 \land Y=1} \\ {WHILE X < N DO (X:=X+1; Y:=Y \times X)} \\ {Y=N!} \end{array}$$

- 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:  $Y = X! \land X \leq N$
- At end  $X \leq N \land \neg(X < N) \Rightarrow X=N$
- Often need to strenthen invariants to get them to work
  - typical to add stuff to 'carry along' like  $\mathtt{X}{\leq} \mathtt{N}$

**Conjunction and Disjunction** 



- These rules are useful for splitting a proof into independent bits
  - they enable  $\vdash$  {P} C {Q<sub>1</sub>  $\land$  Q<sub>2</sub>} to be proved by proving separately that both  $\vdash$  {P} C {Q<sub>1</sub>} and also that  $\vdash$  {P} C {Q<sub>2</sub>}
- Any proof with these rules could be done without using them
  - i.e. they are theoretically redundant (proof omitted)
  - however, useful in practice

Derived rules for finding proofs

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

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

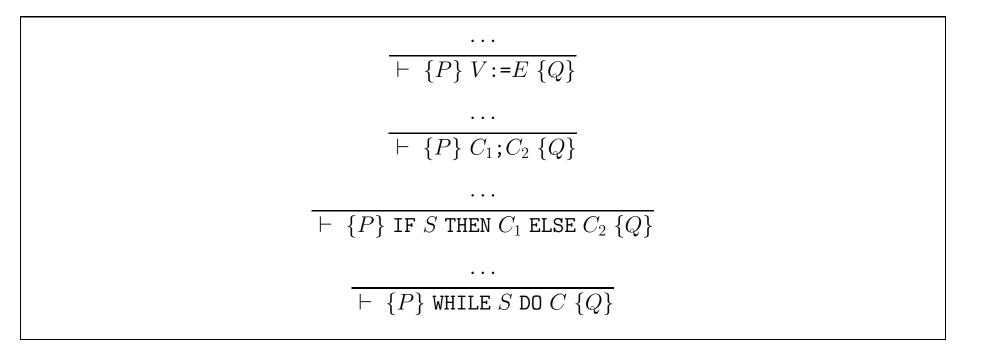
then we could instantiate

 $P \mapsto Precondition, C \mapsto Command, Q \mapsto Postcondition$ to get instances of  $H_1, \dots, H_n$  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



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

# The Derived Assignment Rule

## • An example proof

- 1.  $\vdash$  {R=X  $\land$  0=0} Q:=0 {R=X  $\land$  Q=0} By the assignment axiom.
- 2.  $\vdash$  R=X  $\Rightarrow$  R=X  $\land$  0=0 By pure logic.
- 3.  $\vdash \{R=X\} Q := 0 \{R=X \land Q=0\}$  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**  $\vdash P \Rightarrow Q[E/V]$ 

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

- 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.  $\vdash \mathbf{R}=\mathbf{X} \Rightarrow \mathbf{R}=\mathbf{X} \land \mathbf{0}=\mathbf{0}$  By pure logic.
- 2.  $\vdash$  {R=X} Q:=0 {R=X \land 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:

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

• Hence by the sequenced assignment rule

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

The Derived Sequencing Rule

• The rule below follows from the sequencing and consequence rules

The Derived Sequencing Rule

$$\vdash P \Rightarrow P_{1}$$

$$\vdash \{P_{1}\} C_{1} \{Q_{1}\} \vdash Q_{1} \Rightarrow P_{2}$$

$$\vdash \{P_{2}\} C_{2} \{Q_{2}\} \vdash Q_{2} \Rightarrow P_{3}$$

$$\cdot \qquad \cdot$$

$$\cdot \qquad \cdot$$

$$\vdash P_{2} C_{2} \{Q_{2}\} \vdash Q_{2} \Rightarrow P_{3}$$

$$\cdot \qquad \cdot$$

$$\cdot \qquad \cdot$$

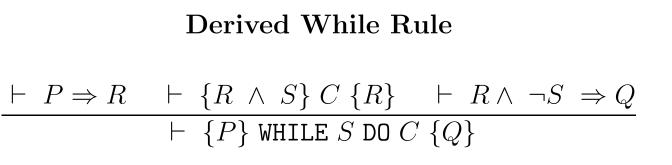
$$\cdot \qquad \cdot$$

$$\vdash P_{2} C_{2} \{Q_{2}\} \vdash Q_{2} \Rightarrow P_{3}$$

$$\cdot \qquad \cdot$$

$$\cdot$$

• Exercise: why no derived conditional rule?



- This follows from the While Rule and the rules of consequence
- Example: it is easy to show
  - $\vdash \quad \texttt{R=X} \ \land \ \texttt{Q=0} \ \Rightarrow \ \texttt{X=R+(Y\times Q)}$
  - $\vdash \{X=R+(Y\times Q)\wedge Y\leq R\} R:=R-Y; Q:=Q+1 \{X=R+(Y\times Q)\}$
  - $\vdash X=R+(Y\times Q)\wedge\neg(Y\leq R) \Rightarrow X=R+(Y\times Q)\wedge\neg(Y\leq R)$
- Then, by the derived While rule

## Forwards and backwards proof

- Previously it was shown how to prove  $\{P\}C\{Q\}$  by
  - $\bullet$  proving properties of the components of C
  - and then putting these together, with the appropriate proof rule, to get the desired property of  ${\cal C}$
- 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