Introduction to Artificial Intelligence

Software Verification

Gerd Beuster

UNIVERSITÄT KOBLENZ-LANDAU

Winter Term 2005

Part I

What is formal specifi cation and verifi cation?

Specification

A specification describes the semantics of a (software-) component.

Specification

A specification describes the semantics of a (software-) component.

Informal Specification “Function `max` returns the maximum of two values.”

Specification

A specification describes the semantics of a (software-) component.

Informal Specification "Function `max` returns the maximum of two values."

Formal Specification

$$\{i = x_0 \wedge j = x_1\}$$

```
k := max(i,j)
```

$$\{((x_0 < x_1) \rightarrow k = x_1) \wedge ((x_0 \geq x_1) \rightarrow k = x_0)\}$$

Formal specification gives a precise description of the component's behavior in a formal language.

Informal Specification

Advantages

- Easy to understand, even for non-experts.
- Good tool support.
- Better than no specification at all. . .

Informal Specification

Advantages

- Easy to understand, even for non-experts.
- Good tool support.
- Better than no specification at all. . .

Disadvantages

- Not precise.
- Details, constraints and excepts may be overlooked.
- Inconsistencies may not be detected.

Informal Specification

Advantages

- Easy to understand, even for non-experts.
- Good tool support.
- Better than no specification at all. . .

Disadvantages

- Not precise.
- Details, constraints and excepts may be overlooked.
- Inconsistencies may not be detected.

$\Rightarrow$ Does the implementation really satisfy the specification?

Formal Specification

Advantages

- Precise
- All details have to be specified.
- Mathematical proofs of properties are possible.

Formal Specification

Advantages

- Precise
- All details have to be specified.
- Mathematical proofs of properties are possible.

$\Rightarrow$ Correctness of implementation can be proven

Checking for correctness

If we have an informal specification only, we run tests against the system.

Checking for correctness

If we have an informal specification only, we run tests against the system.

Advantages

- Writing tests is easy
- Good tool support

Checking for correctness

If we have an informal specification only, we run tests against the system.

Advantages

- Writing tests is easy
- Good tool support

Disadvantages

- Not all cases can be tested. . .

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
    k := j
fi
if(j < i) then
    k := i
fi
```

Is this program correct?
Let's test it!

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
   k := j
fi
if(j < i) then
   k := i
fi
```

i	j	k	
5	3	5	✓

Is this program correct?
Let’s test it!

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
   k := j
fi
if(j < i) then
   k := i
fi
```

i	j	k	
5	3	5	✓
4	7	7	✓

Is this program correct?
Let's test it!

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
    k := j
fi
if(j < i) then
    k := i
fi
```

Is this program correct?
Let's test it!

i	j	k	
5	3	5	✓
4	7	7	✓
8	0	8	✓

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
    k := j
fi
if(j < i) then
    k := i
fi
```

i	j	k	
5	3	5	✓
4	7	7	✓
8	0	8	✓
0	2	2	✓

Is this program correct?
Let's test it!

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
   k := j
fi
if(j < i) then
   k := i
fi
```

Is this program correct?
Let's test it!

i	j	k	
5	3	5	✓
4	7	7	✓
8	0	8	✓
0	2	2	✓
9	2	9	✓

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
   k := j
fi
if(j < i) then
   k := i
fi
```

i	j	k	
5	3	5	✓
4	7	7	✓
8	0	8	✓
0	2	2	✓
9	2	9	✓
8	9	9	✓

Is this program correct?
Let's test it!

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
   k := j
fi
if(j < i) then
   k := i
fi
```

Is this program correct?
Let's test it!

i	j	k	
5	3	5	✓
4	7	7	✓
8	0	8	✓
0	2	2	✓
9	2	9	✓
8	9	9	✓
5	2	5	✓

Tests

Example: Calculating the maximum of two numbers

A little program

```
Input:i,j Output:k
if(i < j) then
    k := j
fi
if(j < i) then
    k := i
fi
```

Is this program correct?
Let's test it!

i	j	k	
5	3	5	✓
4	7	7	✓
8	0	8	✓
0	2	2	✓
9	2	9	✓
8	9	9	✓
5	2	5	✓
5	5	?	x

Uups!

What about formal specification?

If we have a formal specification, we can give a mathematical proof of the correctness of the component.

What about formal specification?

If we have a formal specification, we can give a mathematical proof of the correctness of the component.

Advantages

- We know once and for all that the component satisfies the specification.
- Enforces clean and good specifications, implementations, and documentations.

What about formal specification?

If we have a formal specification, we can give a mathematical proof of the correctness of the component.

Advantages

- We know once and for all that the component satisfies the specification.
- Enforces clean and good specifications, implementations, and documentations.

Disadvantages

- Expert knowledge required.
- Expensive

(When) is it worth it?

Formal specifications makes always sense. (Well, at least try to be formal as possible. . .)

- Enforces good documentation
- Guarantees compatibility to other components
- If it still does not work, at least you know whose fault it is.

(When) is it worth it?

Proofs of correctness make sense, when. . .

- . . . errors are expensive (Pentium bug)
- . . . errors are dangerous (automotive electronics)
- . . . processed data is sensible (patient data, security systems)
- . . . quality must be guaranteed (demands by law or by the users)

Who does formal verification?

- Intel, AMD, Infineon
  Verify (components of) chips
- BMW
  Automotive system
- T-Systems
  Chipcard based biometric identification system
- AG KI @ Uni-Koblenz
  Verified E-Mail Client as part of a fully verified system
  KeY system for verifying Java programs

Part II

Formal Verifi cation of Software

What is a state?

States describe configurations of a system.

State = Heap + Stack

What is a state?

States describe configurations of a system.

State = Heap + Stack

Heap = Current values of all program variables.

(We will ignore the stack in this talk)

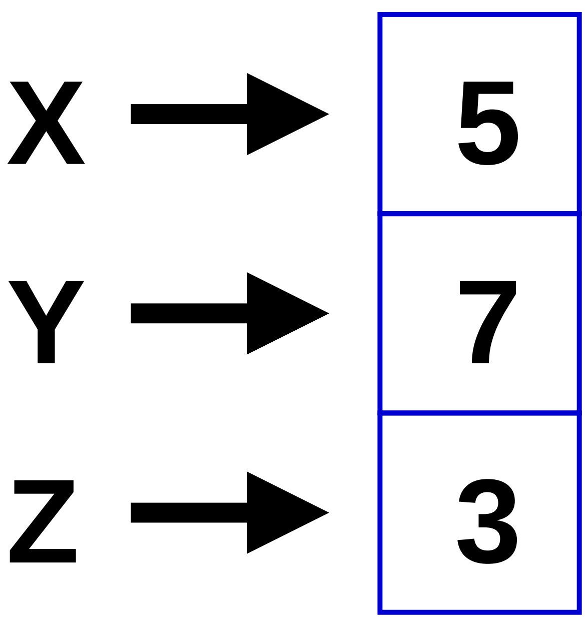
What is a state?

States describe configurations of a system.

State = Heap + Stack

Heap = Current values of all program variables.

(We will ignore the stack in this talk)



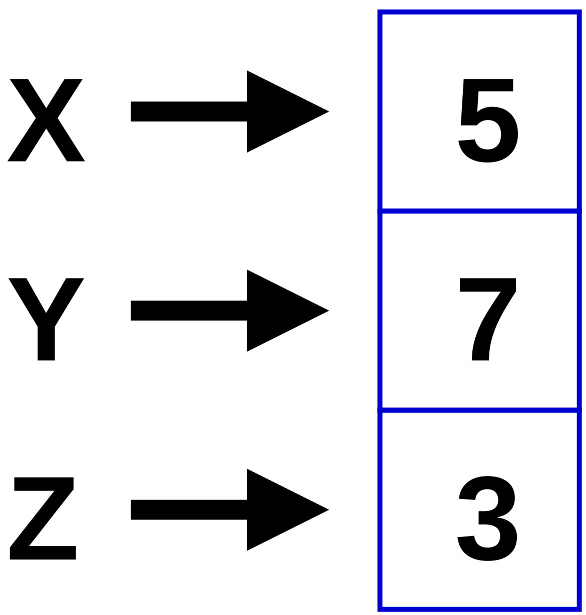
What is a state?

States describe configurations of a system.

State = Heap + Stack

Heap = Current values of all program variables.

(We will ignore the stack in this talk)

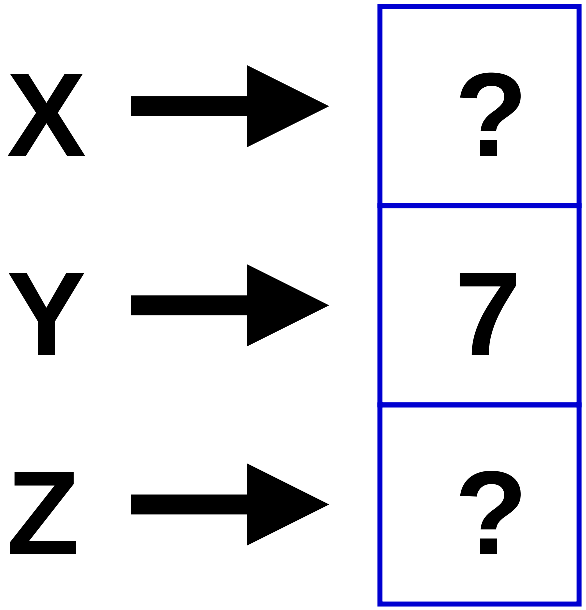


We describe states by logical formulae called conditions.

$$\{x = 5 \wedge y = 7 \wedge z = 3\}$$

What is a state?

Here is another one:

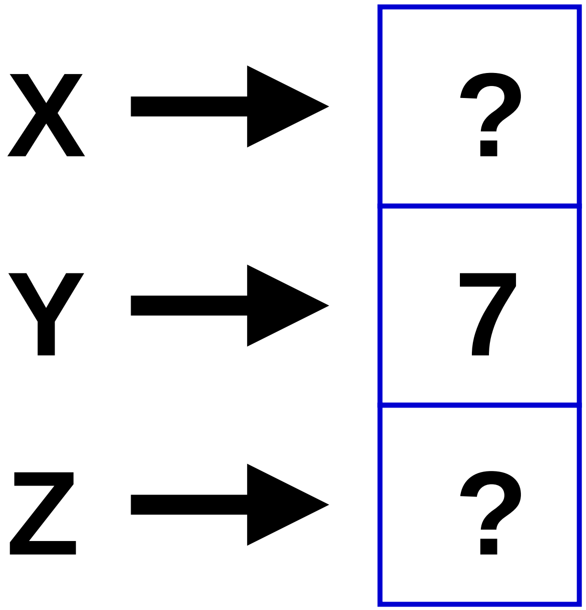


$$\{y = 7\}$$

This describes all states in which $y = 7$. Note that we do not say anything about the values of x and z.

What is a state?

Here is another one:

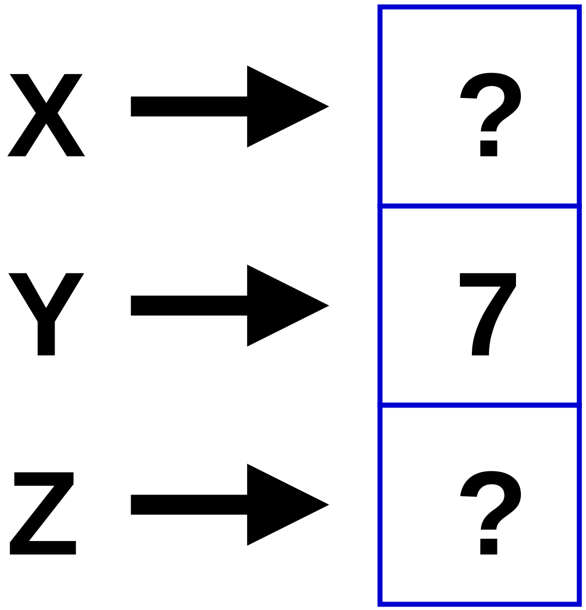


$$\{y < 10\}$$

This conditions requires that the value of y is smaller than 10. The examples also satisfies this condition.

What is a state?

Here is another one:

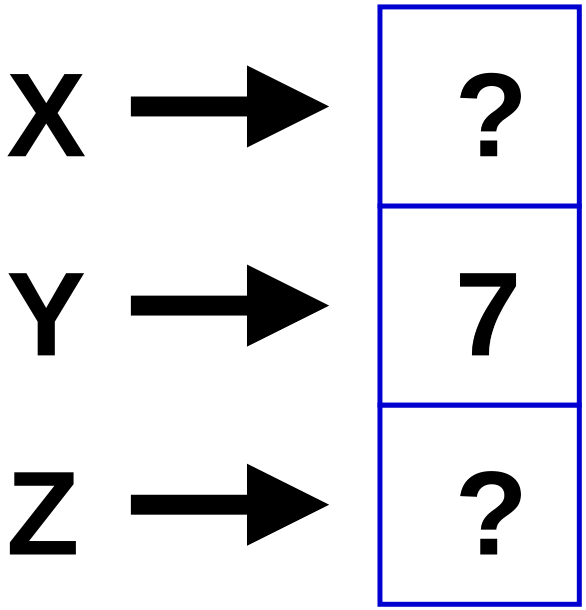


$$\{y = 7\} \Rightarrow \{y < 10\}$$

$\{y = 7\}$ is stronger than $\{y < 10\}$. All states satisfying $\{y = 7\}$ also satisfy $\{y < 10\}$.

What is a state?

Here is another one:



$$\{y = 7\} \Rightarrow \{y < 10\}$$

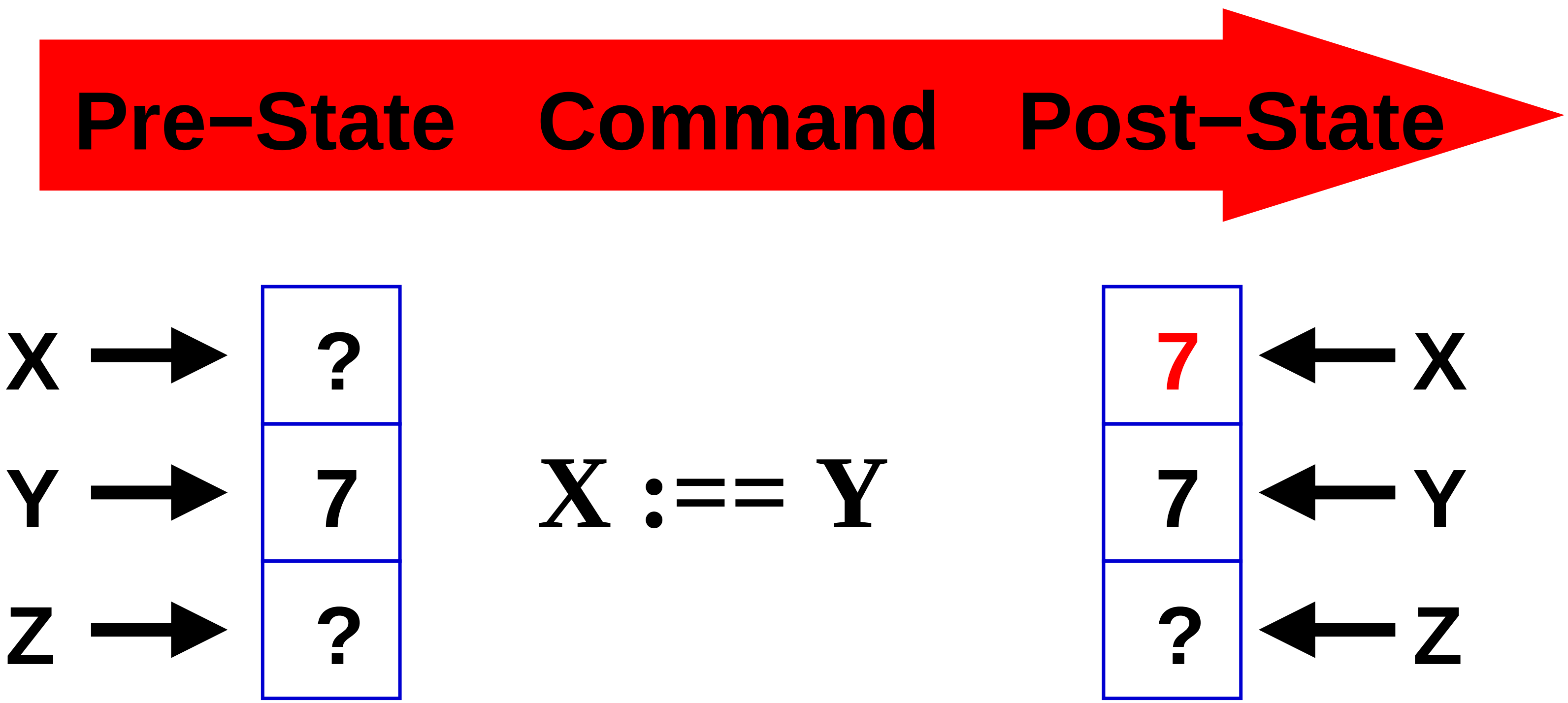
$\{y = 7\}$ is stronger than $\{y < 10\}$. All states satisfying $\{y = 7\}$ also satisfy $\{y < 10\}$.

Wow, we just started reasoning about states!

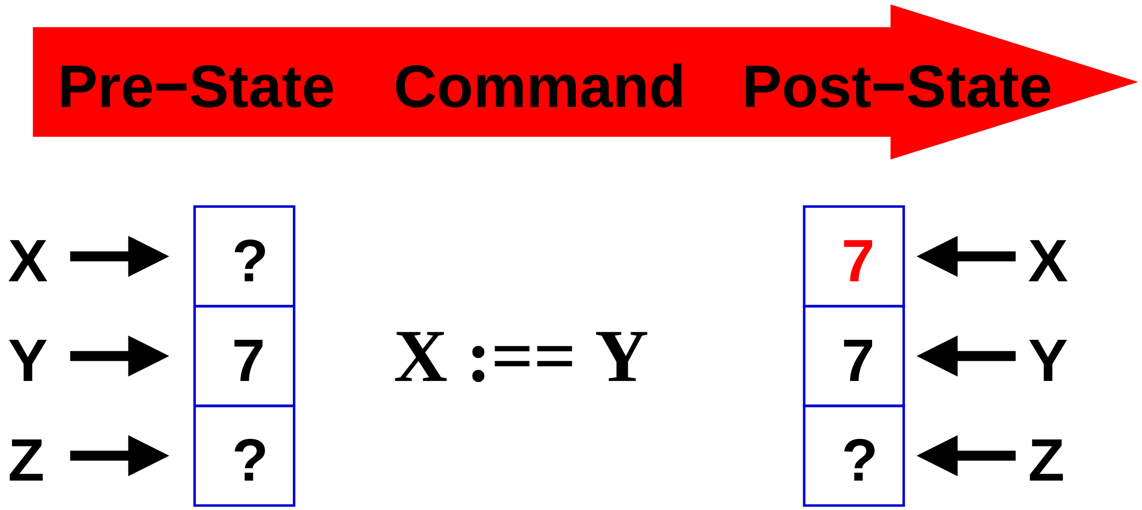
If we could get program instructions into the game, we could prove properties of programs!

How do states change?

Program instructions may change the state.



Hoare Triples



Effects of instructions can be described by Hoare Triples

$$\{\phi\}\ \mathsf{P}\ \{\psi\}$$

$\{\phi\}$ Precondition

P Instruction

$\{\psi\}$ Postcondition

$$\{y = 7\}\ \texttt{x:=y}\ \{x = 7 \wedge y = 7\}$$

Rules describing state changes

$$\{y = 7\}\ \ \texttt{x:=y}\ \ \{x = 7 \wedge y = 7\}$$

This describes only the results of one specific command for a certain set of pre- and postconditions.

Let us try to generalize!

Rules describing state changes

$$\{y = 7\}\ \texttt{x:=y}\ \{x = 7 \wedge y = 7\}$$

This describes only the results of one specific command for a certain set of pre- and postconditions.

Let us try to generalize!

$$\{Q_{[j/i]}\}\ \texttt{i := j}\ \{Q\}$$

($Q_{[x/i]}$ means “replace all occurrences of term i in Q with term x)

Rules describing state changes

$$\{y = 7\}\ \texttt{x:=y}\ \{x = 7 \land y = 7\}$$

This describes only the results of one specific command for a certain set of pre- and postconditions.

Let us try to generalize!

$$\{Q_{[j/i]}\}\ \texttt{i := j}\ \{Q\}$$

($Q_{[x/i]}$ means “replace all occurrences of term i in Q with term x)

We have described the semantics of variable assignments!

Proving programs

We use rule

$$\{Q_{[j/i]}\}\ \texttt{i := j}\ \{Q\}$$

to prove the correctness of our little example

$$\{y = 7\}\ \texttt{x:=y}\ \{x = 7 \wedge y = 7\}$$

Proving programs

We use rule

$$\{Q_{[j/i]}\}\ \texttt{i := j}\ \{Q\}$$

to prove the correctness of our little example

$$\{y = 7\}\ \texttt{x:=y}\ \{x = 7 \wedge y = 7\}$$

$$\dfrac{\dfrac{\top}{\{Q_{[y/x]}\}\ \texttt{x := y}\ \{Q\}}}{\dfrac{\{x = 7 \wedge y = 7_{[y/x]}\}\ \texttt{x:=y}\ \{x = 7 \wedge y = 7\}}{\{y = 7\}\ \texttt{x:=y}\ \{x = 7 \wedge y = 7\}}}$$

From instructions to programs

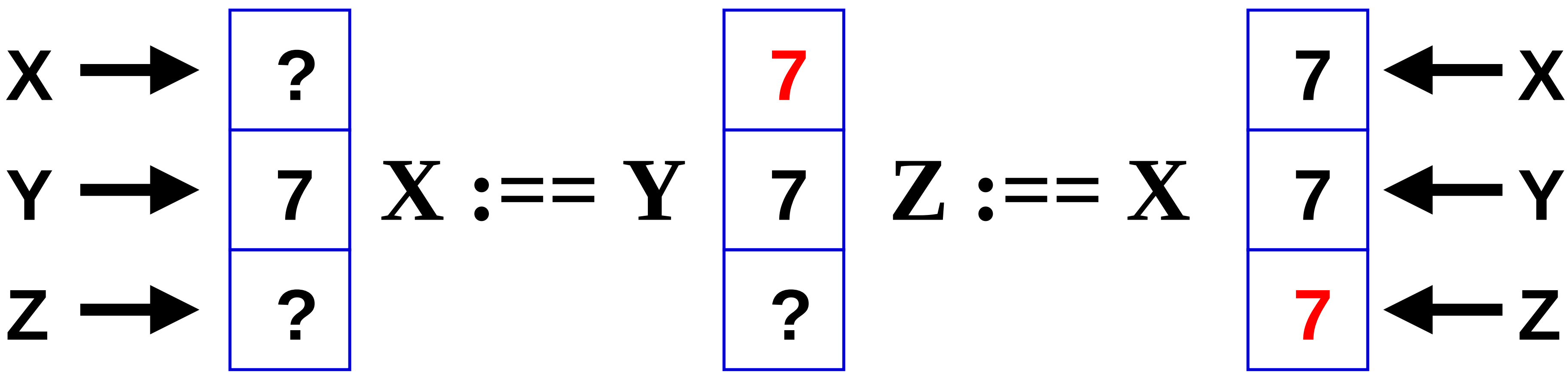
Most programs consist of more than one instruction.

```
x := y;
z := x;
```

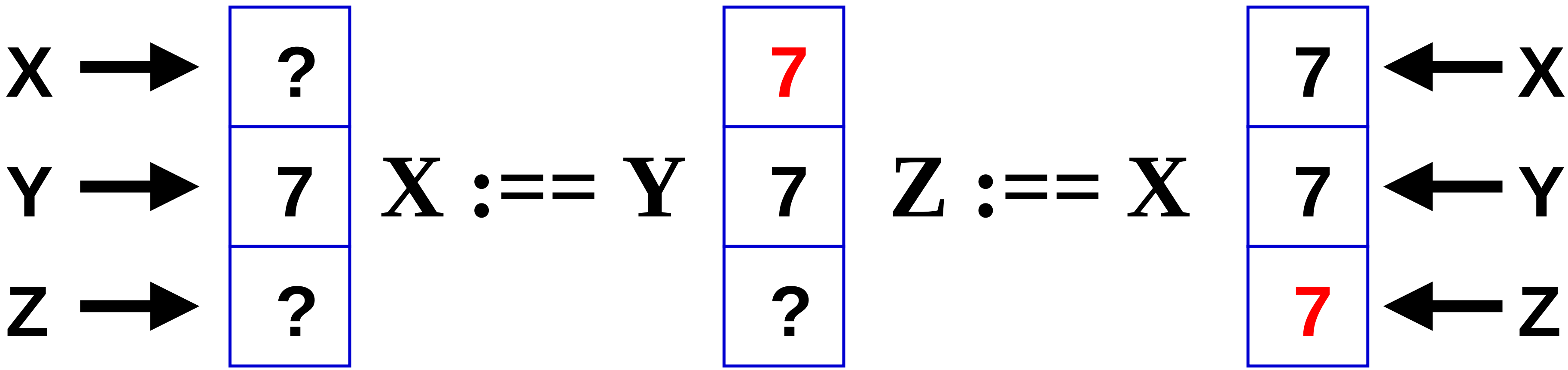
From instructions to programs

Most programs consist of more than one instruction.

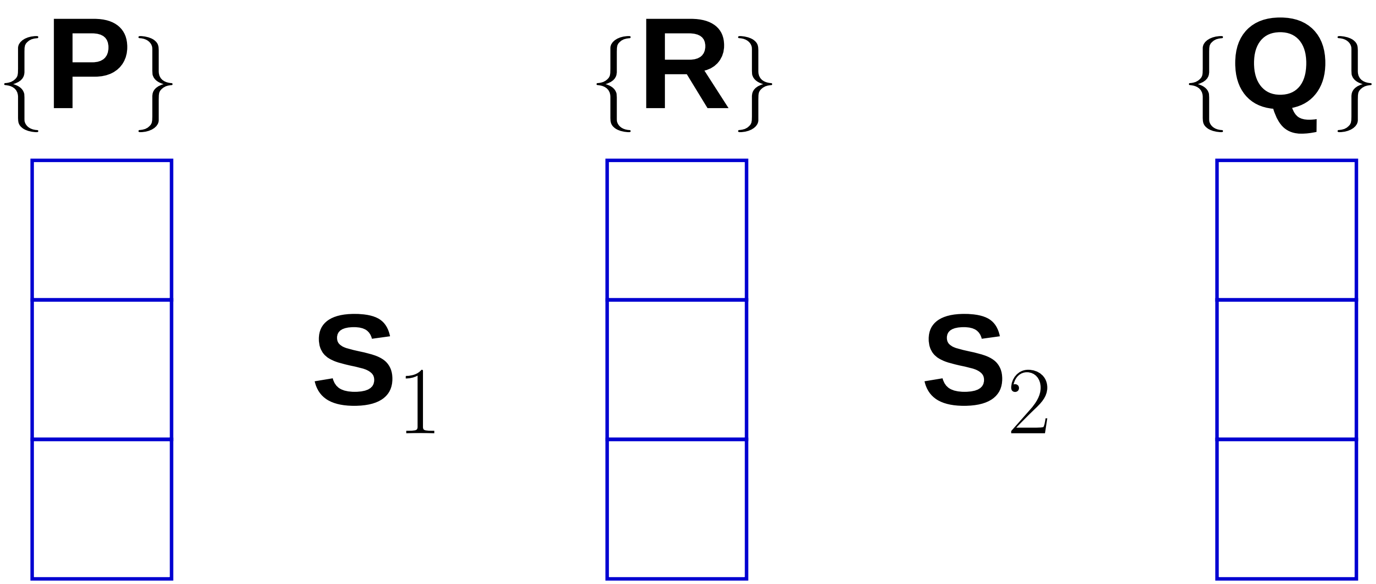
```
x := y;
z := x;
```



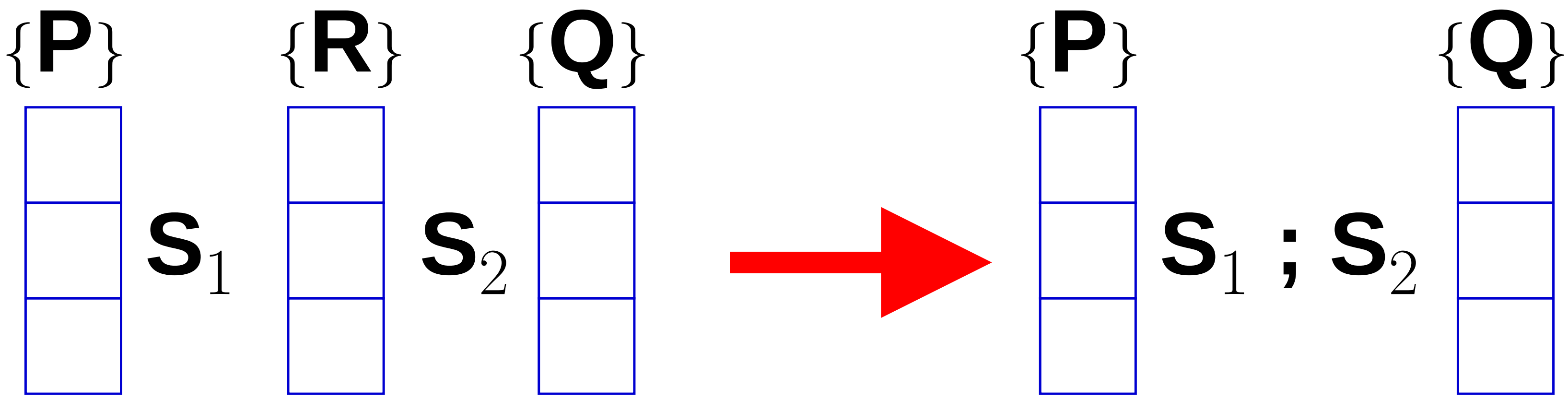
Sequence Rule



Generalized:

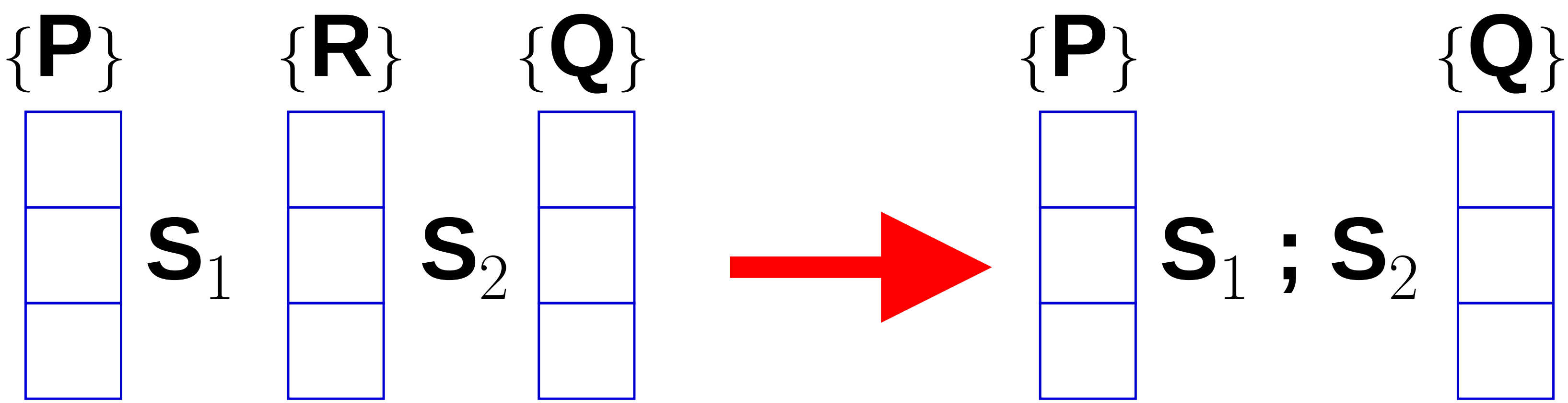


Sequence Rule



$$\frac{\{P\}\ \ S_1\ \ \{R\} \qquad \{R\}\ \ S_2\ \ \{Q\}}{\{P\}\ \ S_1\,\mathbf{;}\,S_2\ \ \{Q\}}$$

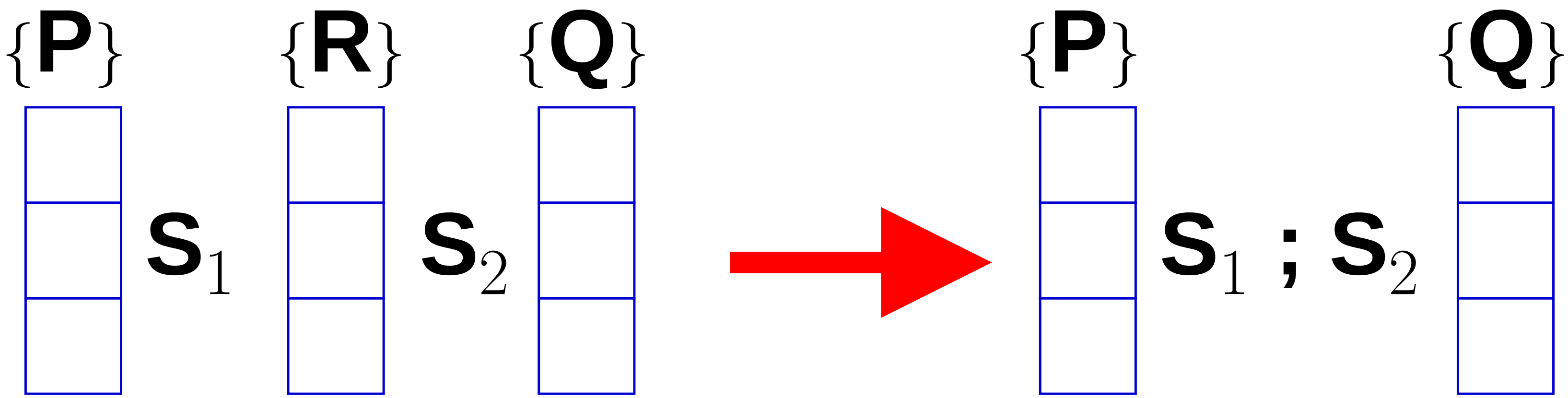
Sequence Rule



$$\frac{\{P\}\ \ S_1\ \ \{R\} \qquad \{R\}\ \ S_2\ \ \{Q\}}{\{P\}\ \ S_1 ; S_2\ \ \{Q\}}$$

Example: $\{y = 7\}$ `x := y ; z := x` $\{R \wedge z = 7\}$

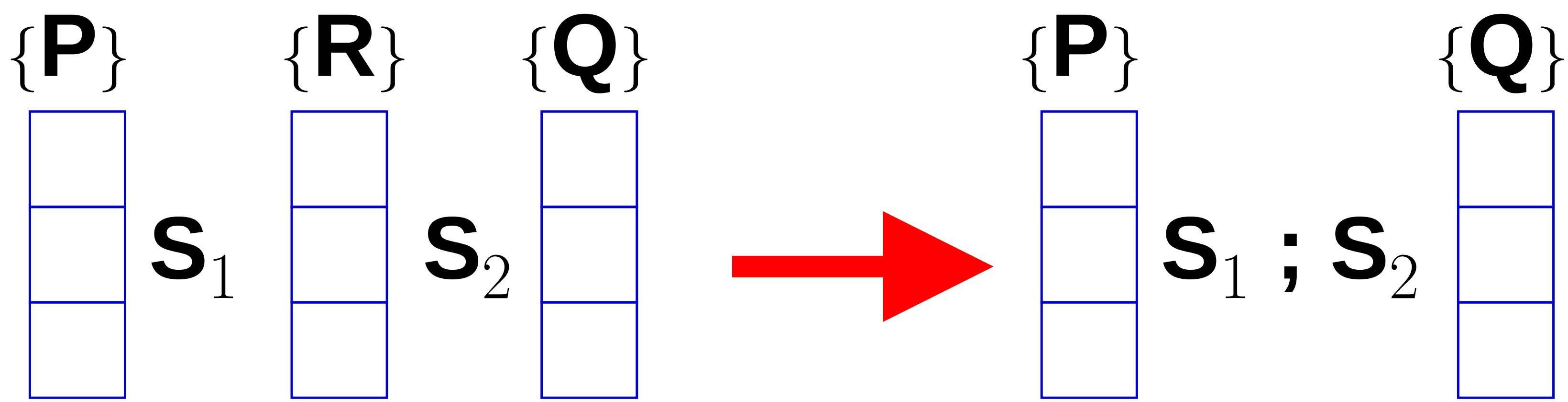
Sequence Rule



$$\frac{\{P\}\ S_1\ \{R\} \qquad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1\, \mathbf{;}\, S_2\ \{Q\}}$$

Example: $\{y = 7\}$ `x := y ; z := x` $\{R \wedge z = 7\}$
Let $R \equiv (x = 7 \wedge y = 7)$

Sequence Rule



$$\frac{\{P\}\ S_1\ \{R\} \qquad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1\,\texttt{;}\,S_2\ \{Q\}}$$

Example: $\{y = 7\}$ `x := y ; z := x` $\{R \wedge z = 7\}$

Let $R \equiv (x = 7 \wedge y = 7)$

$$\dfrac{\dfrac{\dfrac{\top}{\{R_{[y/x]}\}\ \texttt{x:=y}\ \{R\}}}{\{y = 7\}\ \texttt{x:=y}\ \{R\}} \qquad \dfrac{\dfrac{\top}{\{R \wedge z = 7_{[y/z]}\}\ \texttt{z := y}\ \{R \wedge z = 7\}}}{\{R\}\ \texttt{z := y}\ \{R \wedge z = 7\}}}{\{y = 7\}\ \texttt{x := y ; z := x}\ \{R \wedge z = 7\}}$$

Example: Change values of two variables

Specification:

$$\{i = x_0 \wedge j = x_1\} \ \ \mathsf{P} \ \ \{i = x_1 \wedge j = x_0\}$$

Example: Change values of two variables

Specification:

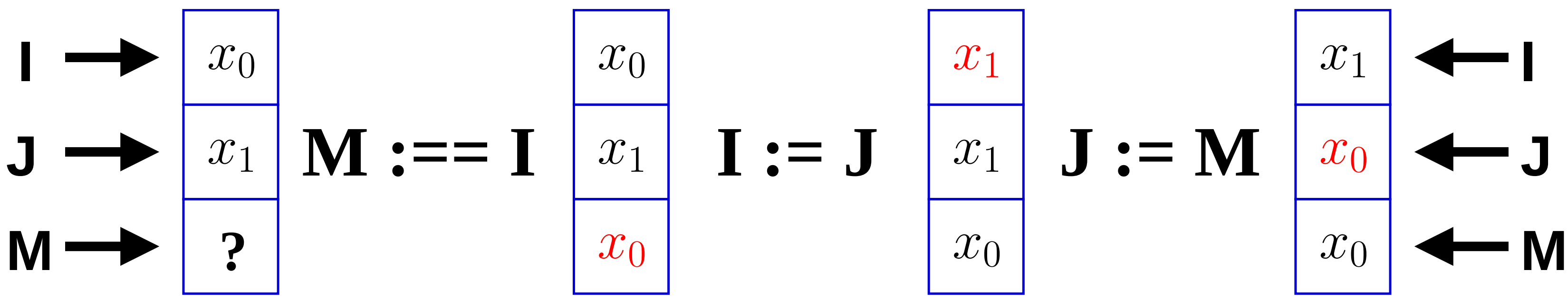
$$\{i = x_0 \wedge j = x_1\} \ \mathsf{P} \ \{i = x_1 \wedge j = x_0\}$$

Lemma:
Satisfied by P =

```
m := i;
i := j;
j := m
```

Example: Change values of two variables

```
m := i;
i := j;
j := m
```



Proof

$$\begin{array}{rcl} P & = & \{i = x_0 \wedge j = x_1\} = R_{[i/m]} \\ Q & = & \{i = x_1 \wedge j = x_0\} \\ R & = & \{m = x_0 \wedge j = x_1\} = S_{[j/i]} \\ S & = & \{m = x_0 \wedge i = x_1\} = Q_{[m/j]} \end{array}$$

$$\dfrac{\dfrac{\dfrac{\top}{\{P\}\ \texttt{m := i}\ \{R\}} \quad \dfrac{\top}{\{R\}\ \texttt{i := j}\ \{S\}}}{\{P\}\ \texttt{m := i ; i := j}\ \{S\}} \quad \dfrac{\top}{\{S\}\ \texttt{j := m}\ \{Q\}}}{\{P\}\ \texttt{m := i ; i := j ; j := m}\ \{Q\}}$$

Program Language Constructs

What do we need for a “real” programming language?

Program Language Constructs

What do we need for a “real” programming language?

Assignments	i := x	✓
Sequences	S_1 ; S_2	✓
Conditionals	IF E THEN S_1 ELSE S_2	Next slide
Loops	WHILE E DO S_1	Later…

Example: Calculate maximum

Specification:

$$\{i = x_0 \wedge j = x_1\}\ \texttt{P}\ \{k = \max(x_0, x_1)\}$$

Lemma: Satisfied by P =

```
IF(i < j) THEN
   k := j
ELSE
   k := i
FI
```

Example: Calculate maximum

Specification:

$$\{i = x_0 \wedge j = x_1\} \ \mathsf{P} \ \{k = \max(x_0, x_1)\}$$

Side note: Why not specify this as

$$\{\} \ \mathsf{P} \ \{k = \max(i, j)\}?$$

Example: Calculate maximum

Specification:

$$\{i = x_0 \land j = x_1\} \;\; \mathsf{P} \;\; \{k = \max(x_0, x_1)\}$$

Side note: Why not specify this as

$$\{\} \;\; \mathsf{P} \;\; \{k = \max(i, j)\}?$$

Because a valid solution would be

P =

```
i := 5;
j := 5;
k := 5;
```

Example: Calculate maximum

$$\{i = x_0 \wedge j = x_1\}\ \mathsf{P}\ \{k = \max(x_0, x_1)\}$$

$$\frac{\{P \wedge E\}\ S_1\ \{Q\} \qquad \{P \wedge \neg E\}\ S_2\ \{Q\}}{\{P\}\ \texttt{if}\ E\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}\ \{Q\}}$$

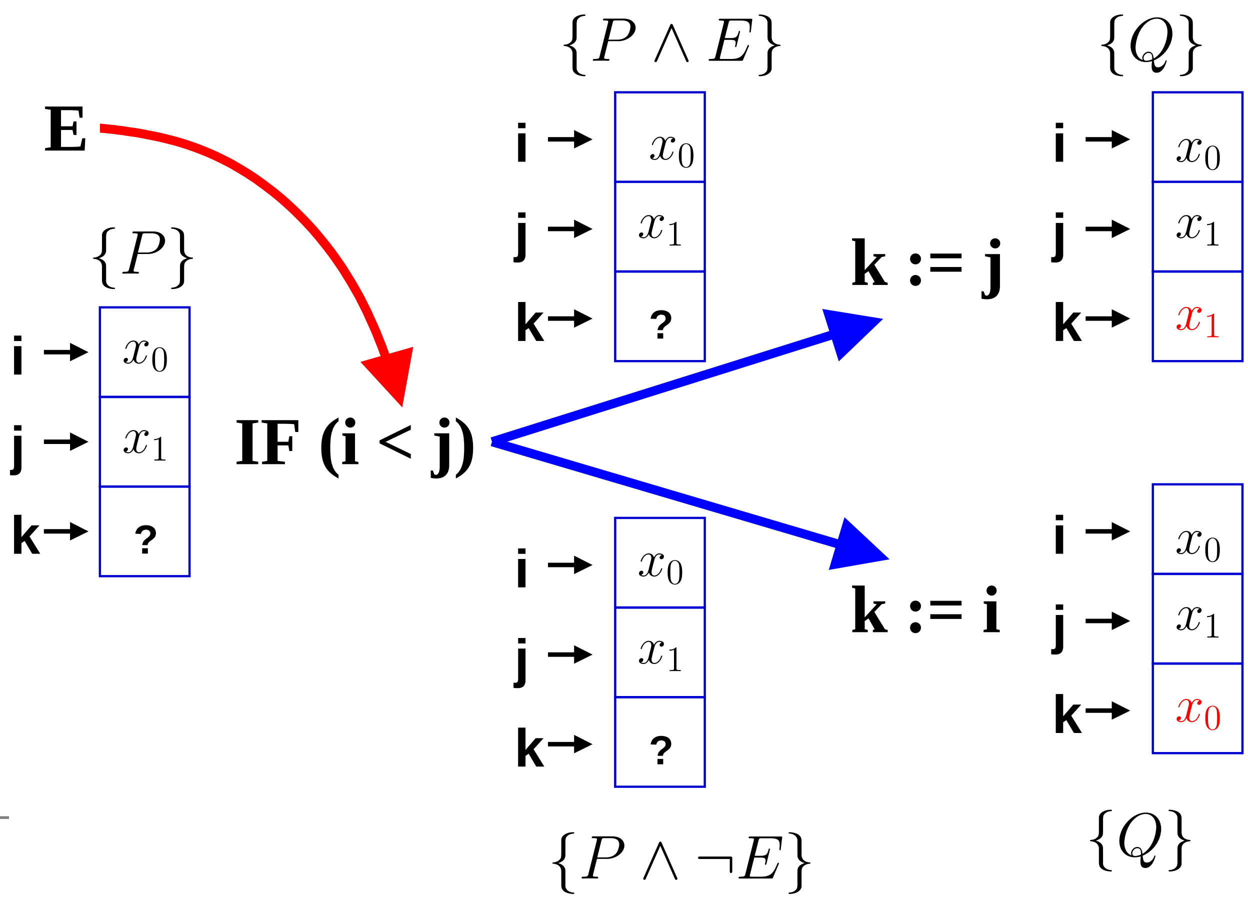
```
IF(i < j) THEN
    k := j
ELSE
    k := i
FI
```

Example: Calculate maximum

$\{i = x_0 \wedge j = x_1\}$ P $\{k = \max(x_0, x_1)\}$

$$\frac{\{P \wedge E\}\ S_1\ \{Q\} \qquad \{P \wedge \neg E\}\ S_2\ \{Q\}}{\{P\}\ \texttt{if}\ E\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}\ \{Q\}}$$

```
IF(i < j) THEN
    k := j
ELSE
    k := i
FI
```



Proof

$$\begin{aligned}
P &= \{i = x_0 \wedge j = x_1\} \\
Q &= \{k = \max(x_0, x_1)\} \\
&= \{((x_0 < x_1) \rightarrow k = x_1) \wedge ((x_0 \geq x_1) \rightarrow k = x_0)\} \\
E &= \{i < j\}
\end{aligned}$$

$$\dfrac{\dfrac{\text{Next Slide}}{\{P \wedge E\}\ \texttt{k := j}\ \{Q\}} \qquad \dfrac{\text{Next Slide}}{\{P \wedge \neg E\}\ \texttt{k := i}\ \{Q\}}}{\{P\}\ \texttt{if i < j then k := j else k := i fi}\ \{Q\}}$$

Proof

$$
\begin{aligned}
P &= \{i = x_0 \wedge j = x_1\} \\
Q &= \{((x_0 < x_1) \rightarrow k = x_1) \wedge ((x_0 \geq x_1) \rightarrow k = x_0)\} \\
E &= \{i < j\}
\end{aligned}
$$

$$
\dfrac{\dfrac{\text{Next Slide}}{\{P \wedge E\}\ \texttt{k := j}\ \{Q\}} \qquad \dfrac{\text{Next Slide}}{\{P \wedge \neg E\}\ \texttt{k := i}\ \{Q\}}}{\{P\}\ \texttt{if i < j then k := j else k := i fi}\ \{Q\}}
$$

Proof

$$
\begin{aligned}
P &= \{i = x_0 \wedge j = x_1\} \\
Q &= \{((x_0 < x_1) \to k = x_1) \wedge ((x_0 \geq x_1) \to k = x_0)\} \\
E &= \{i < j\}
\end{aligned}
$$

We want to show:
$\{P \wedge E\}$ `k := j` $\{Q\}$ and $\{P \wedge \neg E\}$ `k := i` $\{Q\}$

$$
\dfrac{\dfrac{?}{\{i = x_0 \wedge j = x_1 \wedge x_0 < x_1\}\ \texttt{k := j}\ \{((x_0 < x_1) \to k = x_1) \wedge ((x_0 \geq x_1) \to k}}{\dfrac{\{i = x_0 \wedge j = x_1 \wedge i < j\}\ \texttt{k := j}\ \{((x_0 < x_1) \to k = x_1) \wedge ((x_0 \geq x_1) \to k}{\{P \wedge E\}\ \texttt{k := j}\ \{Q\}}}
$$

Proof

$$
\begin{array}{rcl}
P & = & \{i = x_0 \land j = x_1\} \\
Q & = & \{((x_0 < x_1) \rightarrow k = x_1) \land ((x_0 \geq x_1) \rightarrow k = x_0)\} \\
E & = & \{i < j\}
\end{array}
$$

We want to show:
$\{P \land E\}$ `k := j` $\{Q\}$ and $\{P \land \neg E\}$ `k := i` $\{Q\}$

$$
\dfrac{\dfrac{?}{\{i = x_0 \land j = x_1 \land x_0 < x_1\} \ \texttt{k := j} \ \{((x_0 < x_1) \rightarrow k = x_1) \land ((x_0 \geq x_1) \rightarrow k}}{\dfrac{\{i = x_0 \land j = x_1 \land i < j\} \ \texttt{k := j} \ \{((x_0 < x_1) \rightarrow k = x_1) \land ((x_0 \geq x_1) \rightarrow k}{\{P \land E\} \ \texttt{k := j} \ \{Q\}}}
$$

We need one more rule!

Strength

Remember what we said about some conditions being stronger than other?

Strength

Remember what we said about some conditions being stronger than other?

If condition A is stronger than condition B ($A \Rightarrow B$), then for all states where A holds, B holds as well.

Strength

Remember what we said about some conditions being stronger than other?

If condition A is stronger than condition B ($A \Rightarrow B$), then for all states where A holds, B holds as well.

Examples:

$$\begin{aligned} \{y = 7\} &\quad\Rightarrow\quad \{y < 10\} \\ \{y = 7 \wedge x = 5\} &\quad\Rightarrow\quad \{y = 7\} \\ \{y = 7 \wedge x = 5\} &\quad\not\Rightarrow\quad \{x < 7\} \end{aligned}$$

Pre-Strengthening/ Post-Weakening

Rule:

$$\frac{P \Rightarrow P' \qquad \{P'\}\ \mathsf{S}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ \mathsf{S}\ \{Q\}}$$

Pre-Strengthening/ Post-Weakening

Rule:

$$\frac{P \Rightarrow P' \qquad \{P'\}\ \mathsf{S}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ \mathsf{S}\ \{Q\}}$$

Example:

$$\frac{\dfrac{\top}{\{z \leq 5_{[y/z]}\}\ \mathsf{z\ :=\ y}\ \{z \leq 5\}} \qquad \dfrac{\top}{z \leq 5 \Rightarrow z \leq 10}}{\{y \leq 5\}\ \mathsf{z\ :=\ y}\ \{z \leq 10\}}$$

Let's continue the proof for max!

We want to show:

$\{P \wedge E\}$ `k := j` $\{Q\}$ (and $\{P \wedge \neg E\}$ `k := i` $\{Q\}$)

Let's continue the proof for max!

We want to show:

$\{P \wedge E\}$ `k := j` $\{Q\}$ (and $\{P \wedge \neg E\}$ `k := i` $\{Q\}$)

We use post-weakening

$$\frac{\{P \wedge E\}\ \texttt{k := j}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P \wedge E\}\ \texttt{k := j}\ \{Q\}}$$

with $Q' = \{i = x_0 \wedge k = x_1 \wedge i < k\}$

Let's continue the proof for max!

We want to show:

$\{P \wedge E\}$ `k := j` $\{Q\}$ (and $\{P \wedge \neg E\}$ `k := i` $\{Q\}$)

We use post-weakening

$$\frac{\{P \wedge E\}\ \texttt{k := j}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P \wedge E\}\ \texttt{k := j}\ \{Q\}}$$

with $Q' = \{i = x_0 \wedge k = x_1 \wedge i < k\}$

First part of the proof:

$$\frac{\dfrac{\top}{\{i = x_0 \wedge k = x_1 \wedge i < k_{[j/k]}\}\ \texttt{k := j}\ \{i = x_0 \wedge k = x_1 \wedge i < k\}}}{\dfrac{\{i = x_0 \wedge j = x_1 \wedge i < j\}\ \texttt{k := j}\ \{i = x_0 \wedge k = x_1 \wedge i < k\}}{\{P \wedge E\}\ \texttt{k := j}\ \{Q'\}}}$$

. . . going on. . .

We have to show: $Q' \Rightarrow Q$

$$\begin{aligned}(i = x_0 \wedge k = x_1 \wedge x_0 < x_1) \quad \Rightarrow \quad & ((x_0 < x_1) \rightarrow k = x_1) \\ & \wedge((x_0 \geq x_1) \rightarrow k = x_0)\end{aligned}$$

Breaking up the conjunction

$$\frac{\dfrac{\top}{(\top \rightarrow \top)}}{(i = x_0 \wedge k = x_1 \wedge x_0 < x_1) \Rightarrow ((x_0 < x_1) \rightarrow k = x_1)}$$

and

$$\frac{\dfrac{\top}{(\bot \rightarrow k = x_0)}}{(i = x_0 \wedge k = x_1 \wedge x_0 < x_1) \Rightarrow ((x_0 \geq x_1) \rightarrow k = x_0)}$$

. . . (still) going on. . .

We have shown:

$$\{P \wedge E\}\ \texttt{k := j}\ \{Q\}$$

. . . (still) going on. . .

We have shown:

$$\{P \wedge E\}\ \texttt{k := j}\ \{Q\}$$

Next step is to prove the else clause:

$$\{P \wedge \neg E\}\ \texttt{k := i}\ \{Q\}$$

. . . (still) going on. . .

We have shown:

$$\{P \wedge E\}\ \texttt{k := j}\ \{Q\}$$

Next step is to prove the else clause:

$$\{P \wedge \neg E\}\ \texttt{k := i}\ \{Q\}$$

Now you understand why we use automatic theorem provers!

Infinite sequences

So far, we had programs executing in a finite number of steps.

Infinite sequences

So far, we had programs executing in a finite number of steps.

We have shown that The Right Thing (tm) happens in each step.

Infinite sequences

So far, we had programs executing in a finite number of steps.

We have shown that The Right Thing (tm) happens in each step.

Problem: Do all programs execute in a finite number of steps?

Infinite sequences

So far, we had programs executing in a finite number of steps.

We have shown that The Right Thing (tm) happens in each step.

Problem: Do all programs execute in a finite number of steps?

```
10 PRINT "HALLO!"
20 GOTO 10
```

Infinite sequences

So far, we had programs executing in a finite number of steps.

We have shown that The Right Thing (tm) happens in each step.

Problem: Do all programs execute in a finite number of steps?

```
10 PRINT "HALLO!"
20 GOTO 10
```

So we have a problem with. . .

- Loops
- Recursion

Induction to the rescue!

$\{(a > 0) \wedge (b > 0)\}$

```
c := 0
i := 0
WHILE i < a DO
```

$\quad\quad \boldsymbol{\{}(c = b \cdot i) \wedge (i \leq a)\boldsymbol{\}}$

```
    c := c + b
    i := i + 1
OD
```

$\boldsymbol{\{}(c = a \cdot b)\boldsymbol{\}}$

Induction to the rescue!

```
{(a > 0) ∧ (b > 0)}
c := 0
i := 0
WHILE i < a DO
    {(c = b · i) ∧ (i ≤ a)}
    c := c + b
    i := i + 1
OD
{(c = a · b)}
```

$\{(c = b \cdot i) \wedge (i \leq a)\}$ is the loop invariant.

An invariant holds each time the loop test is evaluated. Correctness of loops is shown in two steps

1. The invariant holds on the first iteration.
2. If the invariant held last iteration, it holds this iteration, too.

Induction to the rescue!

$\{(a > 0) \wedge (b > 0)\}$

```
c := 0
i := 0
WHILE i < a DO
```

$\quad\{(c = b \cdot i) \wedge (i \leq a)\}$

```
    c := c + b
    i := i + 1
OD
```

$\{(c = a \cdot b)\}$

The invariant holds on the first iteration.

$$(a > 0) \wedge (c = 0) \wedge (i = 0) \wedge (i < a) \Rightarrow (c = b \cdot i) \wedge (i \leq a)$$

Induction to the rescue!

$\{(a > 0) \wedge (b > 0)\}$

```
c := 0
i := 0
WHILE i < a DO
```

$\quad\{(c = b \cdot i) \wedge (i \leq a)\}$

```
    c := c + b
    i := i + 1
OD
```

$\{(c = a \cdot b)\}$

If the invariant held last iteration, it holds this iteration, too.

$$((c - b) = b \cdot (i - 1)) \wedge ((i - 1) \leq a) \wedge ((i - 1) < a)$$
$$\Rightarrow (c = b \cdot i) \wedge (i \leq a)$$

Induction to the rescue!

$\{(a > 0) \wedge (b > 0)\}$

```
c := 0
i := 0
WHILE i < a DO
```

$\quad\{(c = b \cdot i) \wedge (i \leq a)\}$

```
    c := c + b
    i := i + 1
OD
```

$\{(c = a \cdot b)\}$

No for the postcondition. It must hold if the invariant but not the loop test holds.

$(c = b \cdot i) \wedge (i \leq a) \wedge \neg(i < a)$
$\Rightarrow (c = a \cdot b)$

Partial and total correctness

Important distinction:

Partial Correctness If the program terminates, the post condition holds.

Total Correctness The program terminates and holds.

Partial and total correctness

Important distinction:

Partial Correctness If the program terminates, the post condition holds.

Total Correctness The program terminates and holds.

Is it always possible to prove termination?

Partial and total correctness

Important distinction:

Partial Correctness If the program terminates, the post condition holds.

Total Correctness The program terminates and holds.

Is it always possible to prove termination?

No! $\Rightarrow$ Haltproblem

Soundness and Completeness

Hoare logic is sound and complete. . .

Soundness and Completeness

Hoare logic is sound and complete…

…if the underlying logic is sound and complete.

Soundness and Completeness

Hoare logic is sound and complete...

...if the underlying logic is sound and complete.

In most cases, the logic is sound but incomplete!