Algorithms

Algorithms

Example: An algorithm described by **pseudocode**:

Algorithm 1: An algorithm for finding the maximal element in an array

```
FIND-MAX (A, n):
    k := 0
    for i := 1 to n − 1 do
        if A[i] > A[k] then
            k := i
    return A[k]
```

Algorithms

Algorithm

- processes an **input**
- generates an **output**

From the point of view of an analysis how a given algorithm works, it usually makes only a little difference if the algorithm:

- reads input data from some input device (e.g., from a file, from a keyboard, etc.)
- writes data to some output device (e.g., to a file, on a screen, etc.)

or

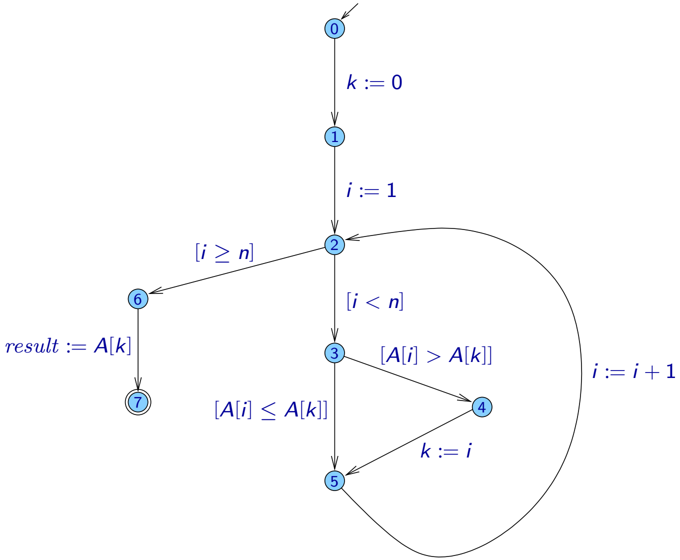
- reads input data from a memory (e.g., they are given to it as parameters)
- writes data somewhere to memory (e.g., it returns them as a return value)

Control Flow

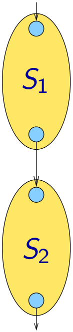
Instructions can be roughly devided into two groups:

- instructions working directly with data:
  - assignment
  - evaluation of values of expressions in conditions
  - reading input, writing output
  - ...
- instructions affecting the **control flow** — they determine, which instructions will be executed, in what order, etc.:
  - branching (if, switch, ...)
  - cycles (while, do .. while, for, ...)
  - organisation of instructions into blocks
  - returns from subprograms (return, ...)
  - ...

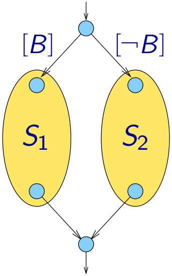
Control Flow Graph



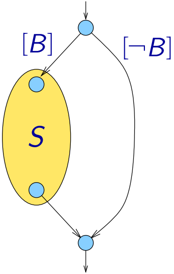
Some Basic Constructions of Structured Programming



$S_1; S_2$

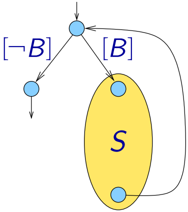


if B **then** S_1 **else** S_2

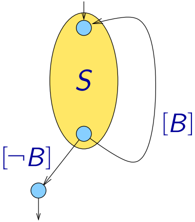


if B **then** S

Some Basic Constructions of Structured Programming

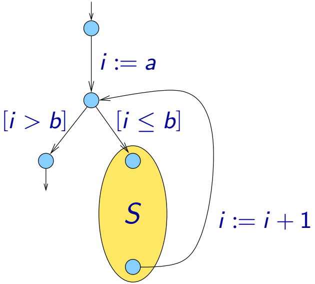


while B **do** S



do S **while** B

Some Basic Constructions of Structured Programming



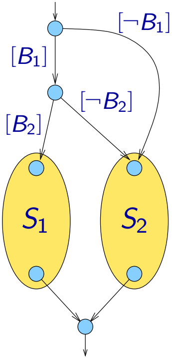
$i := a$
while $i \leq b$ **do**
  S
  $i := i + 1$

for $i := a$ **to** b **do** S

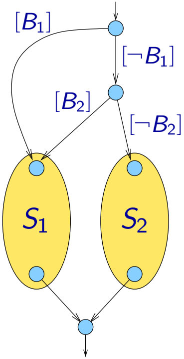
Some Basic Constructions of Structured Programming

Short-circuit evaluation of compound conditions, e.g.:

while $i < n$ **and** $A[i] > x$ **do** ...



if B_1 **and** B_2 **then** S_1 **else** S_2



if B_1 **or** B_2 **then** S_1 **else** S_2

Control-flow Realized by GOTO

- **goto** ℓ — **unconditional jump**
- **if** B **then goto** ℓ — **conditional jump**

Example:

0: $k := 0$
1: $i := 1$
2: **goto** *6*
3: **if** $A[i] \leq A[k]$ **then goto** *5*
4: $k := i$
5: $i := i + 1$
6: **if** $i < n$ **then goto** *3*
7: **return** $A[k]$

Control-flow Realized by GOTO

- **goto** ℓ — **unconditional jump**
- **if** B **then goto** ℓ — **conditional jump**

Example:

```
start:  k := 0
        i := 1
        goto L3
   L1:  if A[i] ≤ A[k] then goto L2
        k := i
   L2:  i := i + 1
   L3:  if i < n then goto L1
        return A[k]
```

Evaluation of Complicated Expressions

Evaluation of a complicated expression such as

$$A[i+s] := (B[3*j+1]+x)*y+8$$

can be replaced by a sequence of simpler instructions on the lower level, such as

$$\begin{aligned}
t_1 &:= i+s\\
t_2 &:= 3*j\\
t_2 &:= t_2+1\\
t_3 &:= B[t_2]\\
t_3 &:= t_3+x\\
t_3 &:= t_3*y\\
t_3 &:= t_3+8\\
A[t_1] &:= t_3
\end{aligned}$$

Computation of an Algorithm

An algorithm is executed by a machine — it can be for example:

- real computer — executes instructions of a machine code
- virtual machine — executes instructions of a bytecode
- some idealized mathematical model of a computer
- ...

The machine can be:

- specialized — executes only one algorithm
- universal — can execute arbitrary algorithm, given in a form of **program**

The machine performs **steps**.

The algorithm processes a particular **input** during its computation.

Computation of an Algorithm

During a computation, the machine must remember:

- the current instruction
- the content of its working memory

It depends on the type of the machine:

- what is the type of data, with which the machine works
- how this data are organized in its memory

Depending on the type of the algorithm and the type of analysis, which we want to do, we can decide if it makes sense to include in memory also the places

- from which the input data are read
- where the output data are written

Computation of an Algorithm

Configuration — the description of the global state of the machine in some particular step during a computation

Example: A configuration of the form

$$(q, mem)$$

where

- q — the current control state
- mem — the current content of memory of the machine — the values assigned currently to variables.

An example of a content of memory mem:

$$\langle A\colon\ [3,8,1,3,6],\quad n\colon\ 5,\quad i\colon\ 1,\quad k\colon\ 0,\quad result\colon\ ?\rangle$$

Computation of an Algorithm

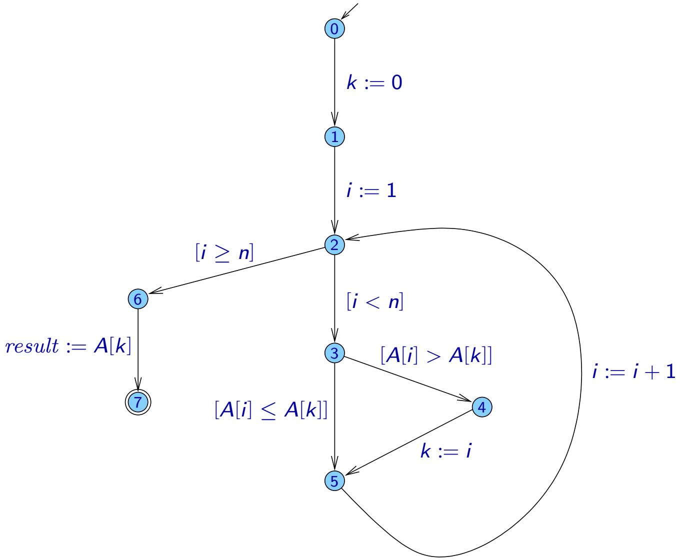
An example of a configuration:

$$(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle)$$

A **computation** of a machine $\mathcal{M}$ executing an algorithm Alg, where it processes an input w, in a sequence of configurations.

- It starts in an **initial configuration**.
- In every step, it goes from one configuration to another.
- The computation ends in a **final configuration**.

Computation of an Algorithm



Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle)$
α_1: $(1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ result\colon ?\rangle)$
α_1: $(1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ result\colon ?\rangle)$
α_2: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon ?, \ \ k\colon ?, \ \ result\colon ?\rangle)$
α_1: $(1, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon ?, \ \ k\colon 0, \ \ result\colon ?\rangle)$
α_2: $(2, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon 1, \ \ k\colon 0, \ \ result\colon ?\rangle)$
α_3: $(3, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon 1, \ \ k\colon 0, \ \ result\colon ?\rangle)$

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle)$
α_1: $(1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_2: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_3: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_4: $(4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon ?, \ \ k\colon ?, \ \ result\colon ?\rangle)$
α_1: $(1, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon ?, \ \ k\colon 0, \ \ result\colon ?\rangle)$
α_2: $(2, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon 1, \ \ k\colon 0, \ \ result\colon ?\rangle)$
α_3: $(3, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon 1, \ \ k\colon 0, \ \ result\colon ?\rangle)$
α_4: $(4, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon 1, \ \ k\colon 0, \ \ result\colon ?\rangle)$
α_5: $(5, \ \langle A\colon [3,8,1,3,6], \ \ n\colon 5, \ \ i\colon 1, \ \ k\colon 1, \ \ result\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{llllllll}
\alpha_0: & (0, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon ?, & k\colon ?, & \mathit{result}\colon ?\rangle) \\
\alpha_1: & (1, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon ?, & k\colon 0, & \mathit{result}\colon ?\rangle) \\
\alpha_2: & (2, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon 1, & k\colon 0, & \mathit{result}\colon ?\rangle) \\
\alpha_3: & (3, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon 1, & k\colon 0, & \mathit{result}\colon ?\rangle) \\
\alpha_4: & (4, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon 1, & k\colon 0, & \mathit{result}\colon ?\rangle) \\
\alpha_5: & (5, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon 1, & k\colon 1, & \mathit{result}\colon ?\rangle) \\
\alpha_6: & (2, & \langle A\colon [3,8,1,3,6], & n\colon 5, & i\colon 2, & k\colon 1, & \mathit{result}\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ result\colon ?\rangle)$
α_1: $(1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ result\colon ?\rangle)$
α_2: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle)$
α_3: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle)$
α_4: $(4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle)$
α_5: $(5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ result\colon ?\rangle)$
α_6: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ result\colon ?\rangle)$
α_7: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ result\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3,8,1,3,6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle)$
α_1: $(1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_2: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_3: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_4: $(4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_5: $(5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_6: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_7: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_8: $(5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

α_0: $(0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle)$
α_1: $(1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_2: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_3: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_4: $(4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$
α_5: $(5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_6: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_7: $(3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_8: $(5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$
α_9: $(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0: & (0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_1: & (1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_2: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_3: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_4: & (4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_5: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_6: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_7: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_8: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_9: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{10}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0: & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ \mathit{result}\colon ?\rangle) \\
\alpha_1: & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ \mathit{result}\colon ?\rangle) \\
\alpha_2: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ \mathit{result}\colon ?\rangle) \\
\alpha_3: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ \mathit{result}\colon ?\rangle) \\
\alpha_4: & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ \mathit{result}\colon ?\rangle) \\
\alpha_5: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_6: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_7: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_8: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_9: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{10}: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{11}: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ \mathit{result}\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3,8,1,3,6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0: & (0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_1: & (1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_2: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_3: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_4: & (4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_5: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_6: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_7: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_8: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_9: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{10}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{11}: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{12}: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{ll}
\alpha_0: & (0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ result\colon ?\rangle) \\
\alpha_1: & (1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ result\colon ?\rangle) \\
\alpha_2: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle) \\
\alpha_3: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle) \\
\alpha_4: & (4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ result\colon ?\rangle) \\
\alpha_5: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_6: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_7: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_8: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_9: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_{10}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_{11}: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_{12}: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ result\colon ?\rangle) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ result\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{13}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{14}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0: & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1: & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4: & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{14}: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{15}: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0: & (0,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ ?,\ \ k{:}\ ?,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_1: & (1,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ ?,\ \ k{:}\ 0,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_2: & (2,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 1,\ \ k{:}\ 0,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_3: & (3,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 1,\ \ k{:}\ 0,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_4: & (4,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 1,\ \ k{:}\ 0,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_5: & (5,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 1,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_6: & (2,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 2,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_7: & (3,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 2,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_8: & (5,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 2,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_9: & (2,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 3,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{10}: & (3,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 3,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{11}: & (5,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 3,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{12}: & (2,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 4,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{13}: & (3,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 4,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{14}: & (5,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 4,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{15}: & (2,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 5,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle) \\
\alpha_{16}: & (6,\ \langle A{:}\ [3,8,1,3,6],\ \ n{:}\ 5,\ \ i{:}\ 5,\ \ k{:}\ 1,\ \ \mathit{result}{:}\ ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon ?,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon ?,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 2,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 3,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{13}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{14}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{15}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{16}\colon & (6,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{17}\colon & (7,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ k\colon 1,\ \ \mathit{result}\colon 8\rangle)
\end{array}
$$

Computation of an Algorithm

By executing an instruction I, the machine goes from configuration α to configuration α':

$$\alpha \xrightarrow{I} \alpha'$$

A computation can be:

- **Finite**:

$$\alpha_0 \xrightarrow{I_0} \alpha_1 \xrightarrow{I_1} \alpha_2 \xrightarrow{I_2} \alpha_3 \xrightarrow{I_3} \alpha_4 \xrightarrow{I_4} \cdots \xrightarrow{I_{t-2}} \alpha_{t-1} \xrightarrow{I_{t-1}} \alpha_t$$

  where α_t is a final configuration

- **Infinite**:

$$\alpha_0 \xrightarrow{I_0} \alpha_1 \xrightarrow{I_1} \alpha_2 \xrightarrow{I_2} \alpha_3 \xrightarrow{I_3} \alpha_4 \xrightarrow{I_4} \cdots$$

Computation of an Algorithm

A computation can be described in two different ways:

- as a sequence of configurations $\alpha_0, \alpha_1, \alpha_2, \ldots$
- as a sequence of executed instructions $I_0, I_1, I_2, \ldots$

Correctness of Algorithms

Algorithms are used for solving **problems**.

- **Problem** — a specification **what** should be computed by an algorithm:
  - Description of inputs
  - Description of outputs
  - How outputs are related to inputs
- **Algorithm** — a particular procedure that describes **how** to compute an output for each possible input

Correctness of Algorithms

Example: The problem of finding a maximal element in an array:

Input: An array A indexed from zero and a number n representing the number of elements in array A. It is assumed that $n \geq 1$.

Output: A value $result$ of a maximal element in the array A, i.e., the value $result$ such that:

- $A[j] \leq result$ for all $j \in \mathbb{N}$, where $0 \leq j < n$, and
- there exists $j \in \mathbb{N}$ such that $0 \leq j < n$ and $A[j] = result$.

An **instance** of a problem — concreate input data, e.g.,

$$A = [3, 8, 1, 3, 6], \; n = 5.$$

The output for this instance is value 8.

Correctness of Algorithms

Definition

An algorithm Alg **solves** a given problem P, if for **each** instance w of problem P, the following conditions are satisfied:

- (a) The computation of algorithm Alg on input w halts after finite number of steps.
- (b) Algorithm Alg generates a correct output for input w according to conditions in problem P.

An algorithm that solves problem P is a correct solution of this problem.

Correctness of Algorithms

Algorithm Alg is **not** a correct solution of problem P if there exists an input w such that in the computation on this input, one of the following incorrect behaviours occurs:

- some incorrect illegal operation is performed (an access to an element of an array with index out of bounds, division by zero, ...),
- the generated output does not satisfy the conditions specified in problem P,
- the computation never halts.

Testing — running the algorithm with different inputs and checking whether the algorithm behaves correctly on these inputs.

Testing can be used to show the presence of bugs but not to show that algorithm behaves correctly for **all** inputs.

Correctness of Algorithms

Generally, it is reasonable to divide a proof of correctness of an algorithm into two parts:

- Showing that the algorithm never does anything "wrong" for any input:
  - no illegal operation is performed during a computation
  - if the program halts, the generated output will be "correct"
- Showing that for every input the algorithm halts after a finite number of steps.

Correctness of Algorithms

Invariant — a condition that must be always satisfied in a given position in a code of the algorithm (i.e., in all possible computations for all allowed inputs) whenever the algorithm goes through this position.

We say that a configuration α is **reachable** if there exists an input w such that α is one of configurations through which the algorithm goes in the computation on input w.

If an algorithm is represented by a control-flow graph, for a given **control state** q (i.e., a node of the graph) we can specify invariants that hold in every reachable configuration with control state q.

Invariants

Invariants can be written as formulas of predicate logic:

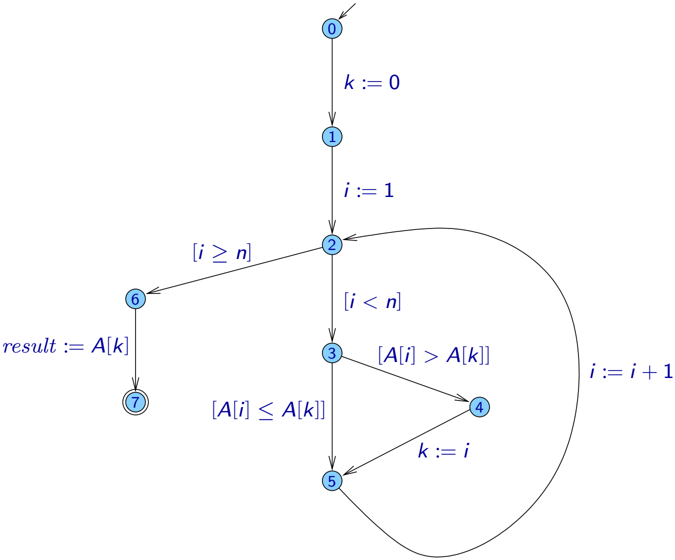
- **free** variables correspond to variables of the program
- a **valuation** is determined by values of program variables in a given configuration

Example: Formula

$$(1 \leq i) \land (i \leq n)$$

holds for example in a configuration where variable i has value 5 and variable n has value 14.

Invariants



Invariants

Examples of invariants:

- an invariant in a control state q is represented by a formula φ_q

Invariants for individual control states (so far only hypotheses):

- φ_0: $(n \geq 1)$
- φ_1: $(n \geq 1) \wedge (k = 0)$
- φ_2: $(n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i)$
- φ_3: $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
- φ_4: $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
- φ_5: $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k \leq i)$
- φ_6: $(n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$
- φ_7: $(n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$

Invariants

Examples of invariants:

- an invariant in a control state q is represented by a formula φ_q

Invariants for individual control states (so far only hypotheses):

- φ_0: $n \geq 1$
- φ_1: $n \geq 1, \;\; k = 0$
- φ_2: $n \geq 1, \;\; 1 \leq i \leq n, \;\; 0 \leq k < i$
- φ_3: $n \geq 1, \;\; 1 \leq i < n, \;\; 0 \leq k < i$
- φ_4: $n \geq 1, \;\; 1 \leq i < n, \;\; 0 \leq k < i$
- φ_5: $n \geq 1, \;\; 1 \leq i < n, \;\; 0 \leq k \leq i$
- φ_6: $n \geq 1, \;\; i = n, \;\; 0 \leq k < n$
- φ_7: $n \geq 1, \;\; i = n, \;\; 0 \leq k < n$

Invariants

Checking that the given invariants really hold:

- It is necessary to check for each instruction of the algorithm that under the assumption that a specified invariant holds before an execution of the instruction, the other specified invariant holds after the execution of the instruction.

Let us assume the algorithm is represented as a control-flow graph:

- edges correspond to instructions
- consider an edge from state q to state q' labelled with instruction I
- let us say that (so far non-verified) invariants for states q and q' are expressed by formulas φ and φ'
- for this edge we must check that for every configurations $\alpha = (q, mem)$ and $\alpha' = (q', mem')$ such that $\alpha \xrightarrow{I} \alpha'$, it holds that if
  - φ holds is configuration α,

  then
  - φ' holds in configuration α'

Invariants

Checking instructions, which are conditional tests:

- an edge labelled with a conditional test $[B]$

A content of memory is not modified.
It is sufficient to check that the following implication holds

$$(\varphi \wedge B) \rightarrow \varphi'$$

Remark: The given implication must hold for all possible values of variables.

Example: Let us assume that formulas contain only variables n, i, k, and that values of these variables are integers:

$$(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z})\,(\varphi \wedge B \rightarrow \varphi')$$

Invariants

Checking those instructions that assign values to variables (they modify a content of memory):

- an edge labelled with assignment $x := E$

φ'' — a formula obtained from formula φ' by renaming of all free occurrences of variable x to x'

It is necessary to check the validity of implication

$$(\varphi \wedge (x' = E)) \rightarrow \varphi''$$

Example: Assignment $k := 3 * k + i + 1$:

$$(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z})(\forall k' \in \mathbb{Z}) \, (\varphi \wedge (k' = 3 * k + i + 1) \rightarrow \varphi'')$$

Invariants

Finishing the checking that the algorithm for finding maximal element in an array returns a correct result (under assumption that it halts):

- ψ_0: φ_0
- ψ_1: $\varphi_1 \wedge (\forall j \in \mathbb{N})(0 \leq j < 1 \rightarrow A[j] \leq A[k])$
- ψ_2: $\varphi_2 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- ψ_3: $\varphi_3 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- ψ_4: $\varphi_4 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k]) \wedge (A[i] > A[k])$
- ψ_5: $\varphi_5 \wedge (\forall j \in \mathbb{N})(0 \leq j \leq i \rightarrow A[j] \leq A[k])$
- ψ_6: $\varphi_6 \wedge (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq A[k])$
- ψ_7: $\varphi_7 \wedge (\mathit{result} = A[k]) \wedge (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq \mathit{result}) \wedge (\exists j \in \mathbb{N})(0 \leq j < n \wedge A[j] = \mathit{result})$

Invariants

Usually it is not necessary to specify invariants in all control states but only in some “important” states — in particular, in states where the algorithm enters or leaves loops:

It is necessary to verify:

- That the invariant holds before entering the loop.
- That if the invariant holds before an iteration of the loop then it holds also after the iteration.
- That the invariant holds when the loop is left.

Invariants

Example: In algorithm FIND-MAX, state 2 is such "important" state.

In state 2, the following holds:

- $n \geq 1$
- $1 \leq i \leq n$
- $0 \leq k < i$
- For each j such that $0 \leq j < i$ it holds that $A[j] \leq A[k]$.

Finiteness of a Computation

Two possibilities how an infinite computation can look:

- some configuration is repeated — then all following configurations are also repeated
- all configurations in a computation are different but a final configuration is never reached

Finiteness of a Computation

One of standard ways of proving that an algorithm halts for every input after a finite number of steps:

- to assign a value from a set W to every (reachable) configuration
- to define an order $\leq$ on set W such that there are no infinite (strictly) decreasing sequences of elements of W
- to show that the values assigned to configuration decrease with every execution of each instruction, i.e., if $\alpha \xrightarrow{I} \alpha'$ then

  $$f(\alpha) > f(\alpha')$$

  ($f(\alpha)$, $f(\alpha')$ are values from set W assigned to configurations α and α')

Finiteness of a Computation

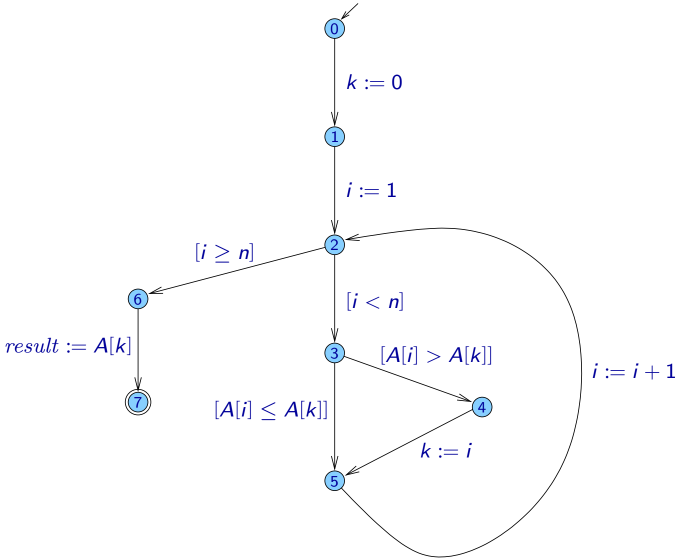
As a set W, we can use for example:

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ with ordering $\leq$.
- The set of vectors of natural numbers with lexicographic ordering, i.e., the ordering where vector $(a_1, a_2, \ldots, a_m)$ is smaller than $(b_1, b_2, \ldots, b_n)$, if
  - there exists i such that $1 \leq i \leq m$ and $i \leq n$, where $a_i < b_i$ and for all j such that $1 \leq j < i$ it holds that $a_j = b_j$, or
  - $m < n$ and for all j such that $1 \leq j \leq m$ is $a_j = b_j$.

  For example, $(5, 1, 3, 6, 4) < (5, 1, 4, 1)$ and $(4, 1, 1) < (4, 1, 1, 3)$.

  Remark: The number of elemets in vectors must be bounded by some constant.

Finiteness of a Computation



Finiteness of a Computation

Example: Vectors assigned to individual configurations:

- State 0: $f(\alpha) = (4)$
- State 1: $f(\alpha) = (3)$
- State 2: $f(\alpha) = (2, n-i, 3)$
- State 3: $f(\alpha) = (2, n-i, 2)$
- State 4: $f(\alpha) = (2, n-i, 1)$
- State 5: $f(\alpha) = (2, n-i, 0)$
- State 6: $f(\alpha) = (1)$
- State 7: $f(\alpha) = (0)$