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

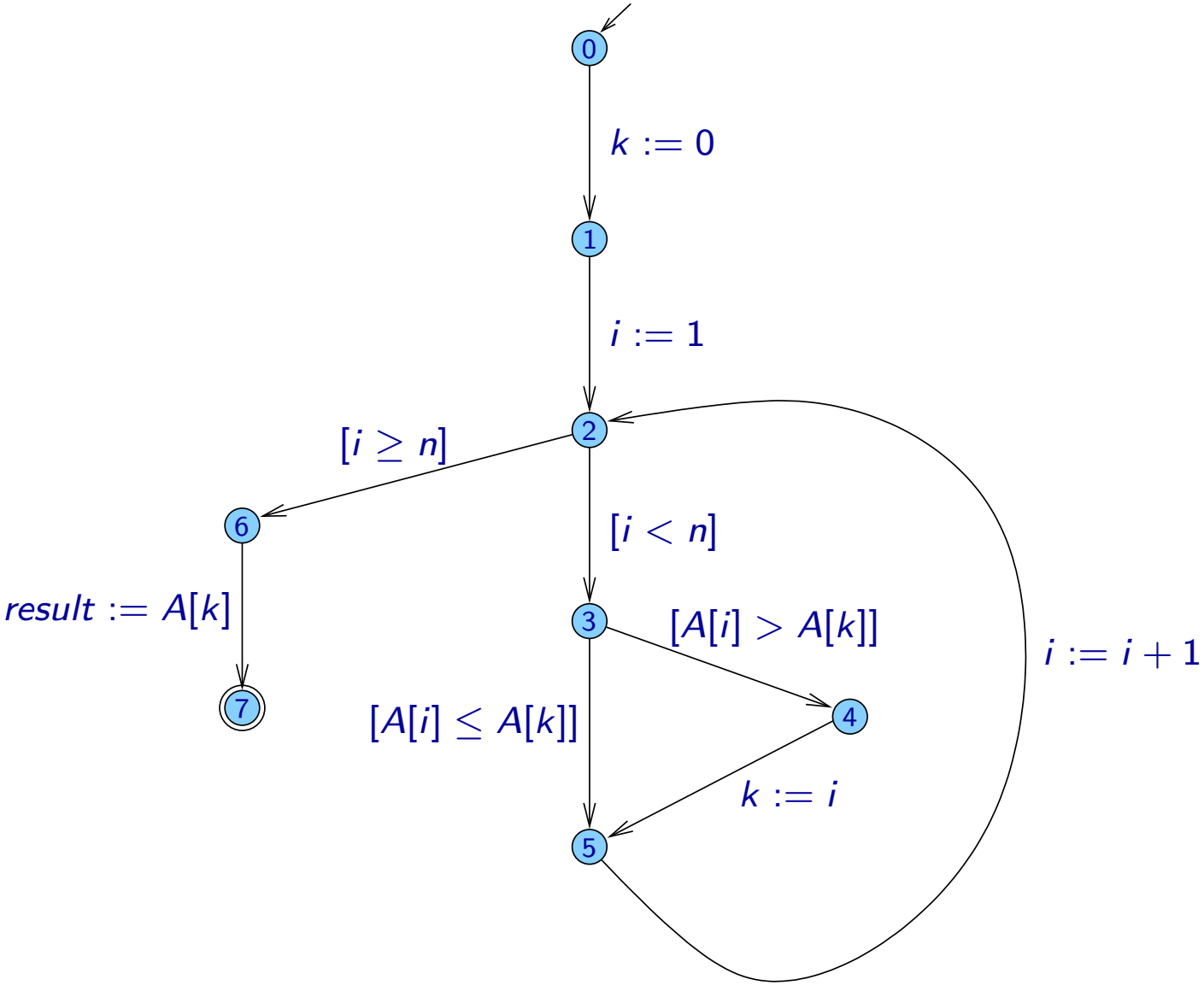
- reads input data from a memory (e.g., they are given to it as parameters)
- writes data somewhere to memory (e.g., it returns them as a return value)

Control Flow

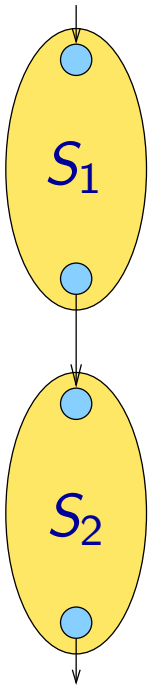
Instructions can be roughly devided into two groups:

- 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, ...)
  - ...

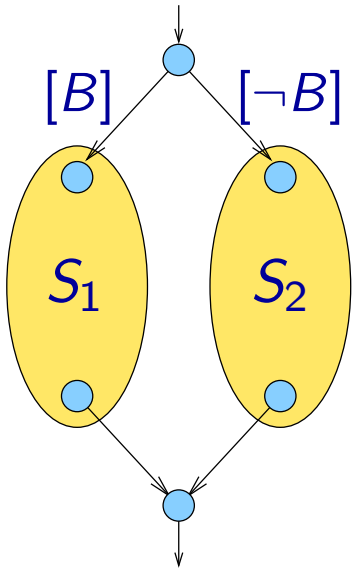
Control Flow Graph



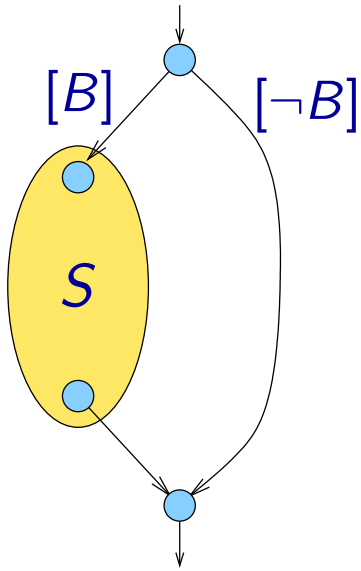
Some Basic Constructions of Structured Programming



$S_1 ; S_2$

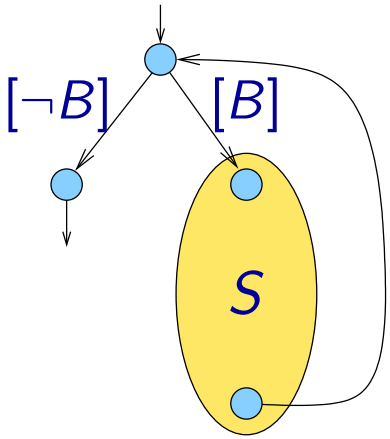


if B **then** S_1 **else** S_2

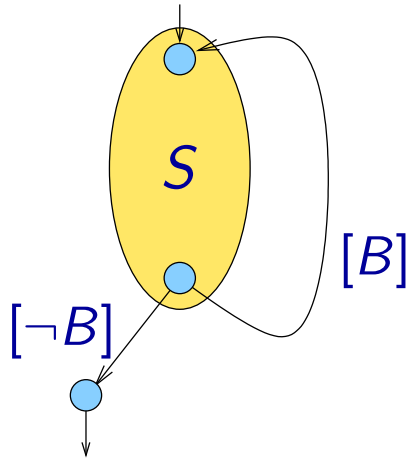


if B **then** S

Some Basic Constructions of Structured Programming

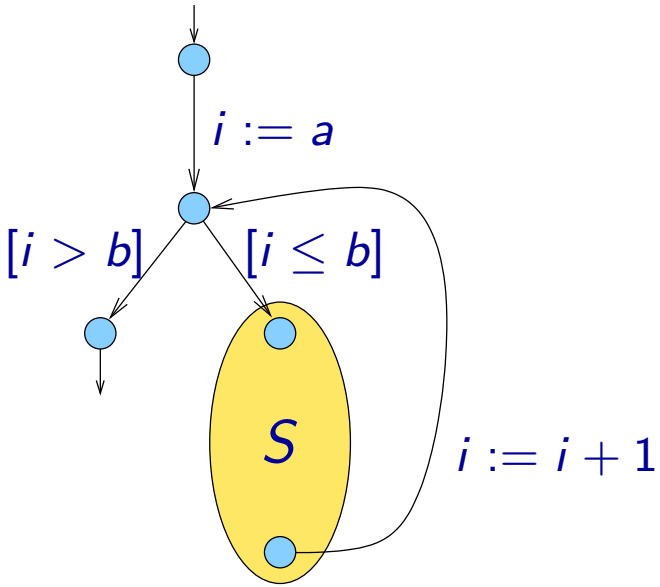


while B **do** S



do S **while** B

Some Basic Constructions of Structured Programming



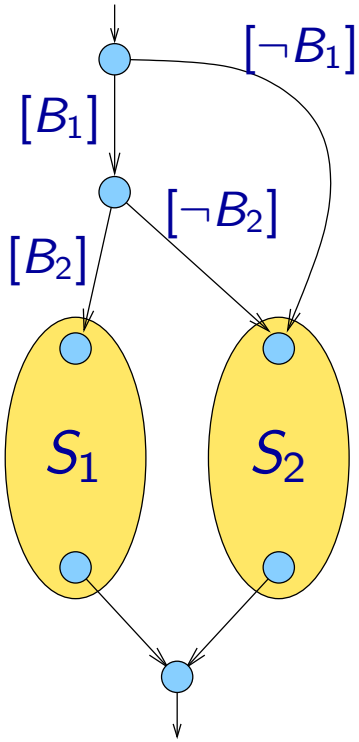
$i := a$
while $i \leq b$ **do**
  S
  $i := i + 1$

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

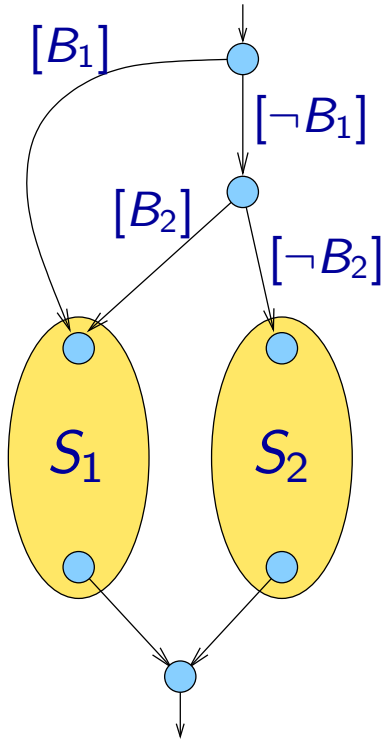
Some Basic Constructions of Structured Programming

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:

$$
\begin{array}{ll}
\mathit{start}: & k := 0 \\
& i := 1 \\
& \textbf{goto } \mathit{L3} \\
\mathit{L1}: & \textbf{if } A[i] \leq A[k] \textbf{ then goto } \mathit{L2} \\
& k := i \\
\mathit{L2}: & i := i + 1 \\
\mathit{L3}: & \textbf{if } i < n \textbf{ then goto } \mathit{L1} \\
& \textbf{return } A[k]
\end{array}
$$

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

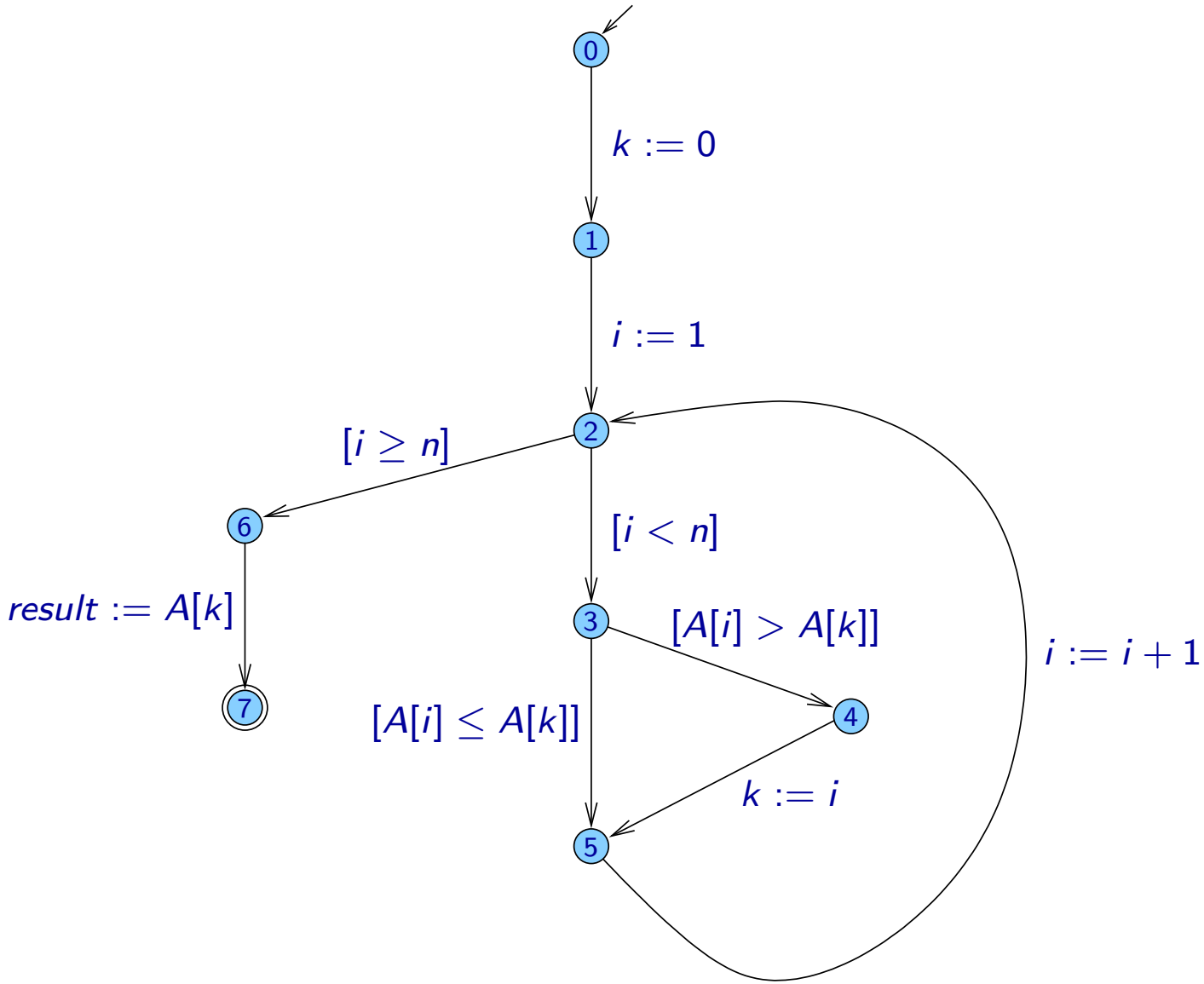
An example of a configuration:

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

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

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

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

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

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

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

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

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

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

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

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: $(0, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } ?, \ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } ?, \ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } ?, \ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } ?, \ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{15}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 5, \ k\text{: } 1, \ result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } ?, \ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{15}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 5, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{16}: $(6, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 5, \ k\text{: } 1, \ result\text{: } ?\rangle)$

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: $(0, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } ?, \ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } ?, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 0, \ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 1, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 2, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 3, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 4, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{15}: $(2, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 5, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{16}: $(6, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 5, \ k\text{: } 1, \ result\text{: } ?\rangle)$
α_{17}: $(7, \langle A\text{: } [3,8,1,3,6], \ n\text{: } 5, \ i\text{: } 5, \ k\text{: } 1, \ result\text{: } 8\rangle)$

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:

- $A[j] \leq \mathit{result}$ 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 $A[j] = \mathit{result}$.

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:

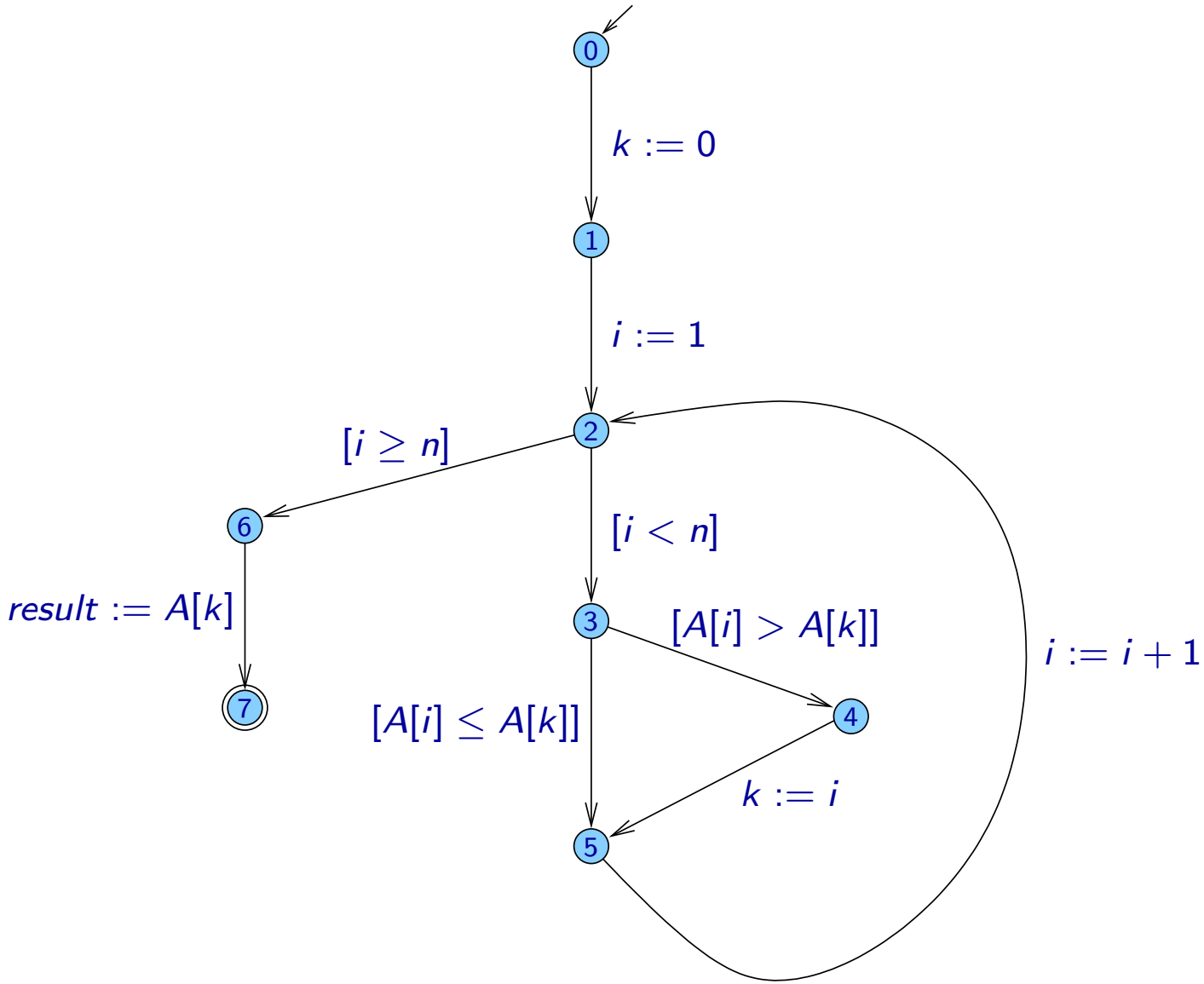
- **free** variables correspond to variables of the program
- a **valuation** is determined by values of program variables in a given configuration

Example: Formula

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

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

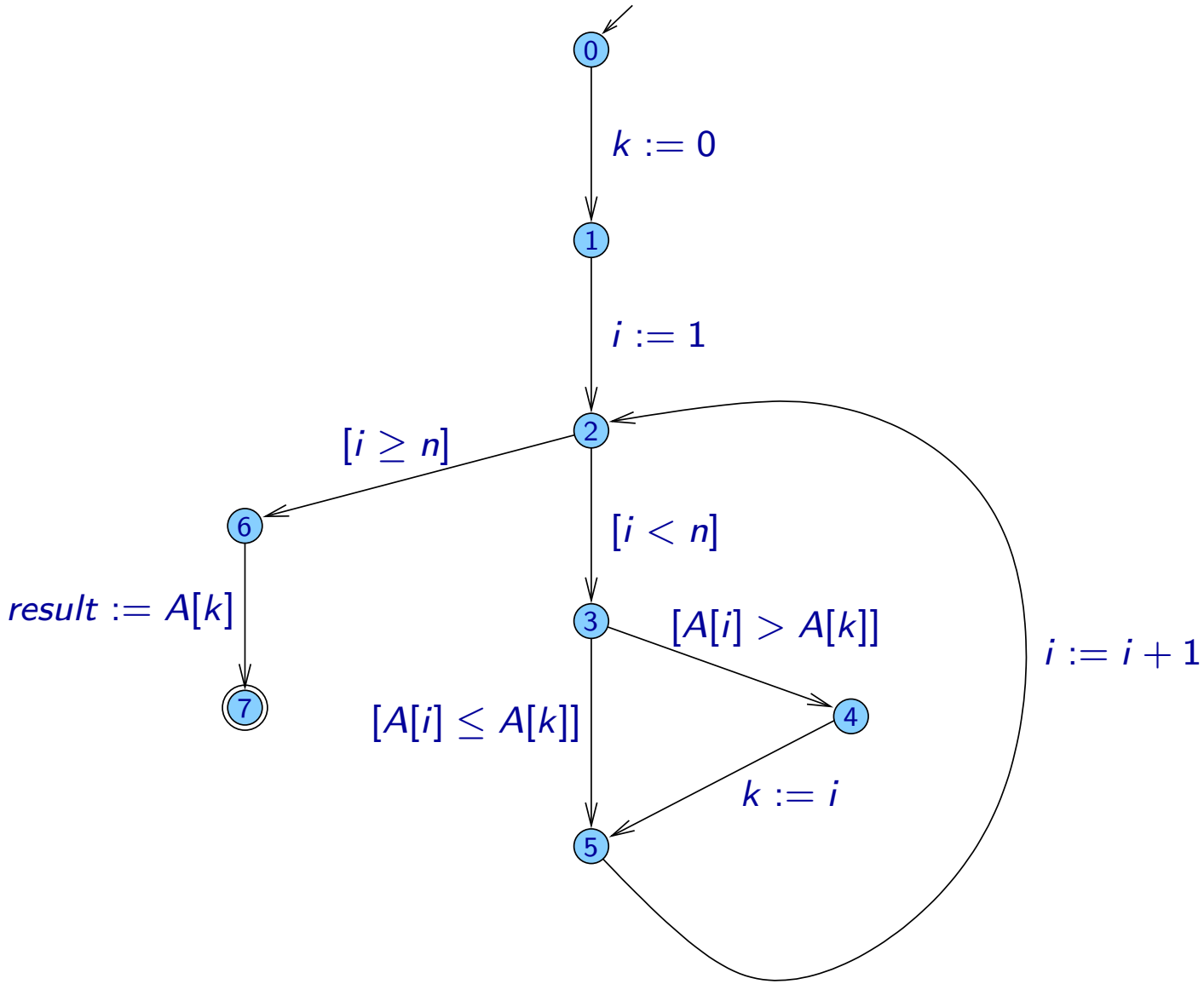
As a set W, we can use for example:

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ with ordering $\leq$.
- The set of vectors of natural numbers with lexicographic ordering, i.e., the ordering where vector $(a_1, a_2, \dots, a_m)$ is smaller than $(b_1, b_2, \dots, b_n)$, 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: $f(\alpha) = (0)$

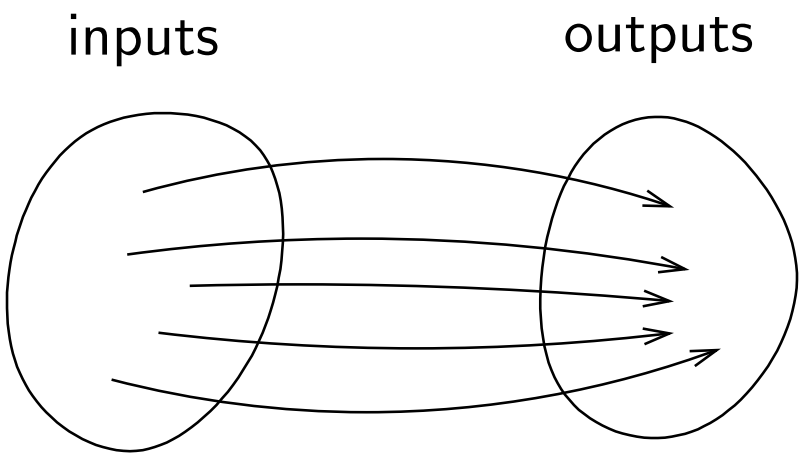
Algorithmic problems

Problems

Problem

When specifying a **problem** we must determine:

- what is the set of possible inputs
- what is the set of possible outputs
- what is the relationship between inputs and outputs



Examples of Problems

Problem "Sorting"

Input: A sequence of elements $a_1, a_2, \ldots, a_n$.

Output: Elements of the sequence $a_1, a_2, \ldots, a_n$ ordered from the least to the greatest.

Example:

- Input: $8, 13, 3, 10, 1, 4$
- Output: $1, 3, 4, 8, 10, 13$

Remark: A particular input of a problem is called an **instance** of the problem.

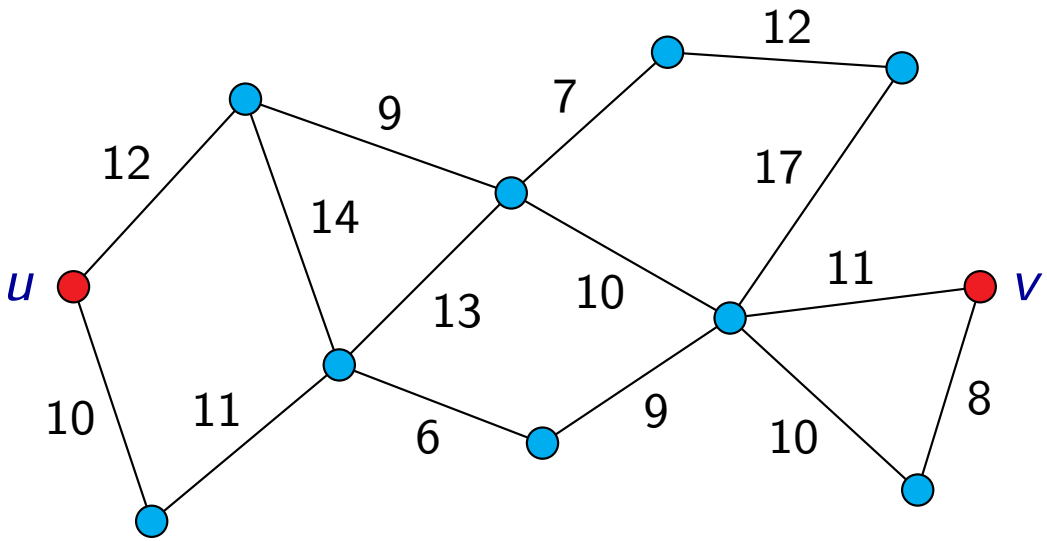
Examples of Problems

Problem "'Finding the shortest path in an (undirected) graph'

Input: An undirected graph $G = (V, E)$ with edges labelled with numbers, and a pair of nodes $u, v \in V$.

Output: The shortest path from node u to node v.

Example:



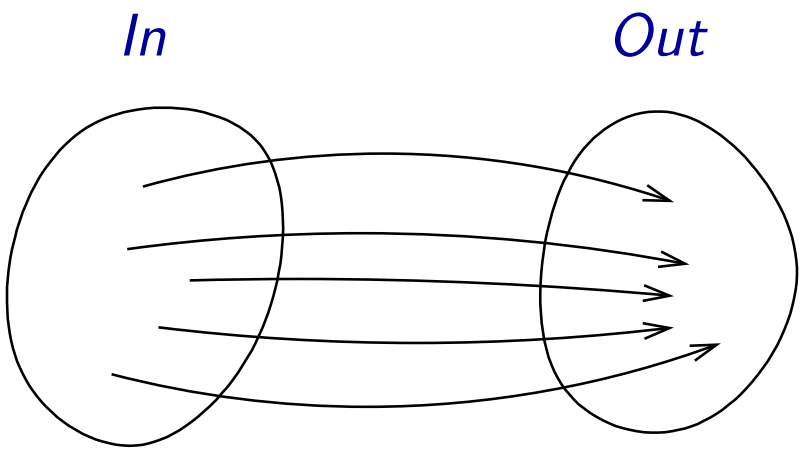
Problems

Problem

So formally, a **problem** can be defined as a tuple (In, Out, R), where:

- In is the set of possible inputs
- Out is the set of possible outputs
- $R \subseteq In \times Out$ is a relation assigning corresponding outputs to each input. This relation must satisfy

$$\forall x \in In : \exists y \in Out : R(x, y).$$



Encoding of Input and Output

In general, we can restrict to the case, where inputs and outputs of a problem are words over some Σ, i.e., $In = Out = \Sigma^*$.

Some other object (numbers, sequences of numbers, graphs, . . . ,) then can be written (encoded) as words over this alphabet.

Example: In the problem "Sorting", we can select as an alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,\}$.

Then an input can be for example the word

$$826,13,3901,101,128,562$$

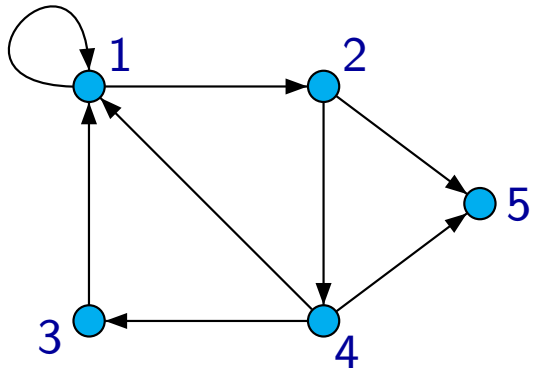
and the output is then the word

$$13,101,128,562,826,3901$$

Encoding of Input and Output

Example: If an input of some problem is for example a graph, it can be represented as a list of nodes and edges:

For example, the following graph



can be represented as word

$$(1,2,3,4,5),((1,2),(2,4),(4,3),(3,1),(1,1),(2,5),(4,5),(4,1))$$

over alphabet $\Sigma = \{0,1,2,3,4,5,6,7,8,9,,,(,)\}$.

Encoding of Input and Output

Remark: Not all words from Σ^* necessarily represent some input. We should choose such encoding that allows us to recognize easily if a word represents some input or not.

Encoding of Input and Output

We can restrict our attention to the case where both inputs and outputs are encoded as words over alphabet $\{0, 1\}$ (i.e., as sequences of bits).

Symbols of any other alphabet can be represented as sequences of bits.

Example: Alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$$
\begin{aligned}
\mathtt{a} &\leftrightarrow 001 \\
\mathtt{b} &\leftrightarrow 010 \\
\mathtt{c} &\leftrightarrow 011 \\
\mathtt{d} &\leftrightarrow 100 \\
\mathtt{e} &\leftrightarrow 101 \\
\mathtt{f} &\leftrightarrow 110 \\
\mathtt{g} &\leftrightarrow 111
\end{aligned}
$$

Word '$\mathtt{defb}$' can be represented as '100101110010'.

Encoding of Input and Output

Words over alphabet $\Sigma = \{0,1\}^*$ (i.e., sequences of bits) can be viewed as representations of numbers written in binary.

So, alternatively we could restrict to the case where

$$In = Out = \mathbb{N},$$

where $\mathbb{N} = \{0,1,2,3,\ldots\}$ is the set of natural numbers.

Other Examples of Problems

Problem "Primality"

Input: A natural number n.

Output: YES if n is a prime, NO otherwise.

Remark: A natural number n is a **prime** if it is greater than 1 and is divisible only by numbers 1 and n.

Few of the first primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

Decision Problems

The situation when the set of outputs *Out* is $\{\text{YES}, \text{NO}\}$ is quite frequent. Such problems are called **decision problems**.

We usually specify decision problems in such a way that instead describing what the output is, we introduce a question.

Example:

Problem "Primality"

Input: A natural number n.

Question: Is n a prime?

Decision Problems

Decision problem

One possibility, how the notion of a **decision problem** can be defined formally, is to define it as a pair (In, T), where:

- In is the set of all inputs,
- $T \subseteq In$ is the set of all inputs, for which the answer is YES.

Decision Problems and Languages

If we restrict to the cases where inputs are words over some alphabet Σ, then decision problems can be viewed as languages.

A language corresponding to a given decision problem is the set of those words from Σ^* that represent inputs for which the answer is $\textsc{Yes}$.

Example: A language consisting of those words from $\{0,1\}^*$ that are binary representations of primes.

For example, $101 \in L$ but $110 \notin L$.

An Example of a Decision Problem

SAT problem (boolean satisfiability problem)

Input: Boolean formula φ.

Question: Is φ satisfiable?

Example:

Formula $\varphi_1 = x_1 \wedge (\neg x_2 \vee x_3)$ is satisfiable:
e.g., for valuation ν where $\nu(x_1) = 1$, $\nu(x_2) = 0$, $\nu(x_3) = 1$, it holds that $\nu(\varphi_1) = 1$.

Formula $\varphi_2 = (x_1 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge x_2)$ is not satisfiable:
for every valuation ν it holds that $\nu(\varphi_2) = 0$.

Optimization Problems

Other special case are the so called optimization problems.

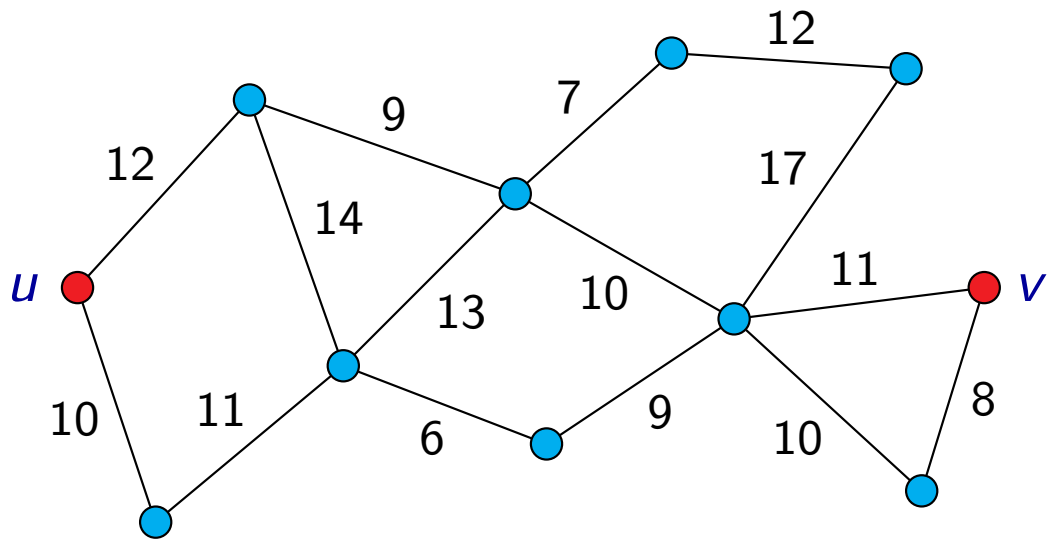
An **optimization problem** is a problem where the aim is to choose, from a set of feasible solutions, a solution that in some respect is optimal.

Optimization Problems

Other special case are the so called optimization problems.

An **optimization problem** is a problem where the aim is to choose, from a set of feasible solutions, a solution that in some respect is optimal.

Example: In the problem "Finding the shortest path in a graph", the set of feasible solutions consists of all paths from the node u to the node v. The criterion by which we compare the paths is the length of a path.



Optimization Problems

Formally, an **optimization problems** can be defined as a tuple (In, Out, f, m, g), where:

- In is the set of inputs,
- Out is the set of **solutions**,
- $f : In \to \mathcal{P}(Out)$ is a function assigning to each input x a set of corresponding **feasible solutions** $f(x)$,
- $m : \bigcup_{x \in In}(\{x\} \times f(x)) \to \mathbb{R}$ is an **optimization function** (**cost function**),
- g is min or max.

The goal is to find for a given input $x \in In$ some feasible solution $y \in f(x)$ such that

$$m(x, y) = g\{m(x, y') \mid y' \in f(x)\},$$

or to find out that there is no such feasible solution for the input x (i.e., $f(x) = \emptyset$).

Optimization Problems

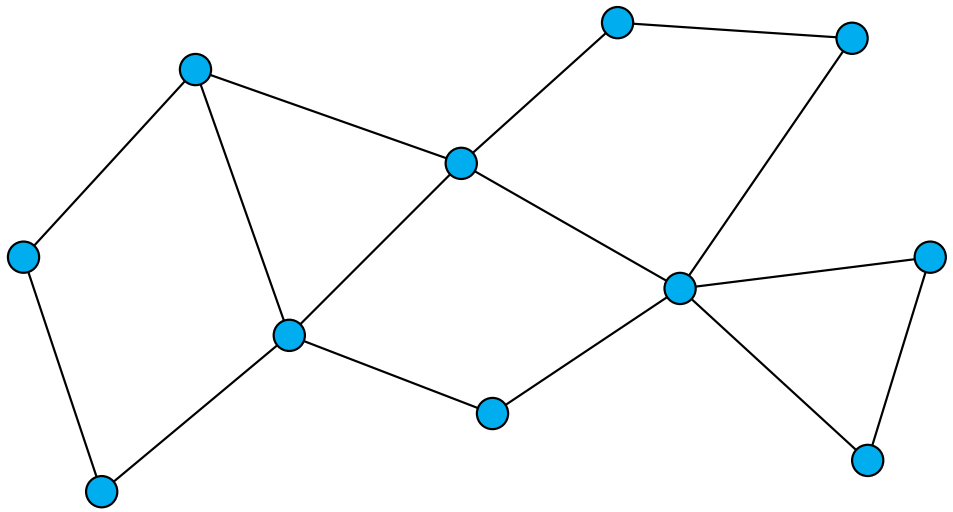
- The optimization problems, where g is min, are called **minimization problems**.
- The optimization problems, where g is max, are called **maximization problems**.

Examples of Problems

Problem "Coloring of a graph with k colors"

Input: An undirected graph G and a natural number k.

Question: Is it possible to color the nodes of the graph G with k colors in such a way that no two nodes connected with an edge are colored with the same color?



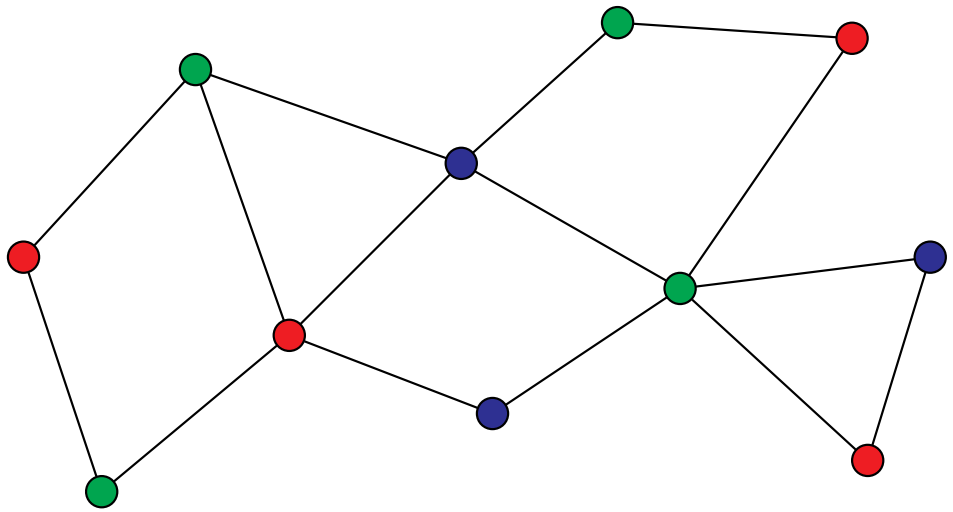
$k = 3$

Examples of Problems

Problem "Coloring of a graph with k colors"

Input: An undirected graph G and a natural number k.

Question: Is it possible to color the nodes of the graph G with k colors in such a way that no two nodes connected with an edge are colored with the same color?



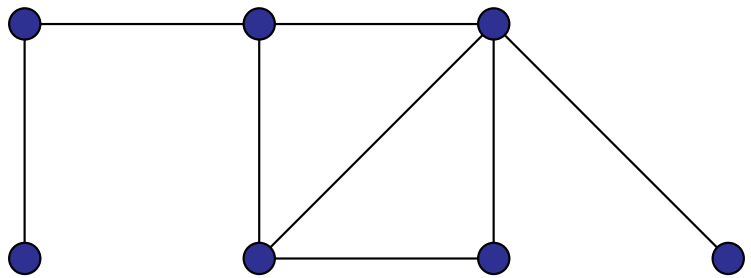
$k = 3$

Examples of Problems

Independent set (IS) problem

Input: An undirected graph G, a number k.

Question: Is there an independent set of size k in the graph G?



$k = 4$

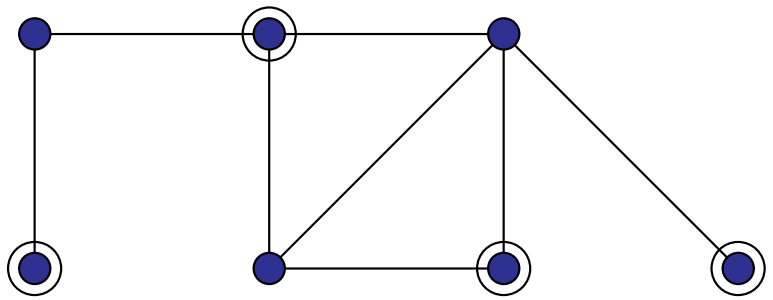
Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

Examples of Problems

Independent set (IS) problem

Input: An undirected graph G, a number k.

Question: Is there an independent set of size k in the graph G?

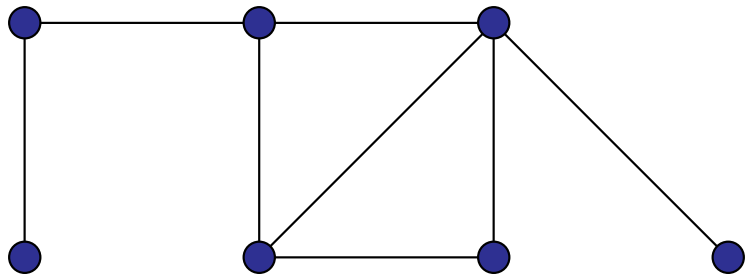


$k = 4$

Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

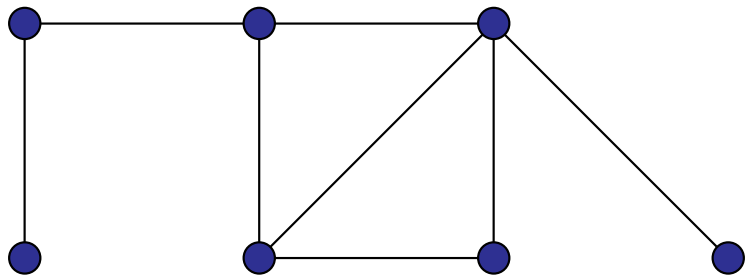
Independent Set (IS) Problem

An example of an instance where the answer is YES:



$k = 4$

An example of an instance where the answer is NO:



$k = 5$

ILP – Integer Linear Programming

Problem ILP (integer linear programming)

Input: An integer matrix A and an integer vector b.

Question: Is there an integer vector x such that $Ax \leq b$?

An example of an instance of the problem:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

So the question is if the following system of inequations has some integer solution:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

ILP – Integer Linear Programming

One of solutions of the system

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

is for example $x_1 = -4$, $x_2 = 1$, $x_3 = 1$, i.e.,

$$x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$$

because

$$\begin{aligned} 3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\ -4 + 1 &= -3 \leq -3 \\ 2 \cdot (-4) + 1 &= -7 \leq 5 \end{aligned}$$

So the answer for this instance is YES.

Examples of Problems

Problem

Input: Deterministic finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

Question: Is $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$?

Problem

Input: Context-free grammars $\mathcal{G}_1$ and $\mathcal{G}_2$.

Question: Is $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$?

Algorithmically Solvable Problems

Let us assume we have a problem P.

If there is an algorithm solving the problem P then we say that the problem P is **algorithmically solvable**.

If P is a decision problem and there is an algorithm solving the problem P then we say that the problem P is **decidable (by an algorithm)**.

If we want to show that a problem P is algorithmically solvable, it is sufficient to show some algorithm solving it (and possibly show that the algorithm really solves the problem P).

Algorithmically Solvable Problems

For many problems it is immediately obvious that they are algorithmically solvable, as for example:

- Sorting
- Finding a shortest path in a graph
- Primality

where it is sufficient to test all possible solutions (in all these examples there are only finitely many possibilities that must be tested), although such trivial algorithm based on **brute force** solution is usually not very efficient.

On the other hand, there are many problems where it is not so clear.

- It can be a nontrivial task to find an algorithm solving the given problem and to show that it really solves it.
- It is possible that there is no algorithm solving the given problem.

Algorithmically Unsolvable Problems

A problem that is not algorithmically solvable is **algorithmically unsolvable**.

A decision problem that is not decidable is **undecidable**.