Introduction to Theoretical Computer Science

Zdeněk Sawa

Department of Computer Science, FEI,
Technical University of Ostrava
17. listopadu 2172/15, Ostrava-Poruba 708 00
Czech republic

February 14, 2024

Lecturer

Name:	doc. Ing. Zdeněk Sawa, Ph.D.
E-mail:	`zdenek.sawa@vsb.cz`
Room:	EA413
Web:	`https://www.cs.vsb.cz/sawa/uti/index-en.html`

On these pages you will find:

- Information about the course
- Study texts
- Slides from lectures
- Exercises for tutorials
- Recent news for the course
- A link to a page with animations

Requirements

- **Credit** (30 points):
  - Written test (24 points) — it will be written on a tutorial
    - The minimal requirement for obtaining the credit is 12 points.
    - A correcting test for 20 points.
  - Activity on tutorials (6 points)
    - The minimal requirement for obtaining the credit is 3 points.
- **Exam** (70 points)
  - A written exam consisting of two parts (35 points for each part); it is necessary to obtain at least 12 points for each part.
  - It is necessary to obtain at least 30 points.

Theoretical Computer Science

Theoretical computer science — a scientific field on the border between computer science and mathematics

- investigation of general questions concerning algorithms and computations
- study of different kinds of formalisms for description of algorithms
- study of different approaches for description of syntax and semantics of formal languages (mainly programming languages)
- a mathematical approach to analysis and solution of problems (proofs of general mathematical propositions concerning algorithms)

Theoretical Computer Science

Examples of some typical questions studied in theoretical computer science:

- Is it possible to solve the given problem using some algorithm?
- If the given problem can be solved by an algorithm, what is the computational complexity of this algorithm?
- Is there an efficient algorithm solving the given problem?
- How to check that a given algorithm is really a correct solution of the given problem?
- What kinds instructions are sufficient for a given machine to perform a given algorithm?

Algorithms and Problems

Algorithm — mechanical procedure that computes something (it can be executed by a computer)

Algorithms are used for solving **problems**.

An example of an algorithmic problem:

Input:	Natural numbers x and y.
Output:	Natural number z such that $z = x + y$.

Algorithms and Problems

Algorithm — mechanical procedure that computes something (it can be executed by a computer)

Algorithms are used for solving **problems**.

An example of an algorithmic problem:

Input: Natural numbers x and y.

Output: Natural number z such that $z = x + y$.

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

Example: An example of an instance of the problem given above is a pair of numbers 728 and 34.

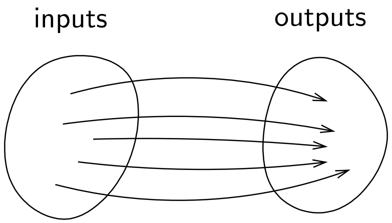
The corresponding output for this instance is number 762.

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: $1, 3, 4, 8, 10, 13$

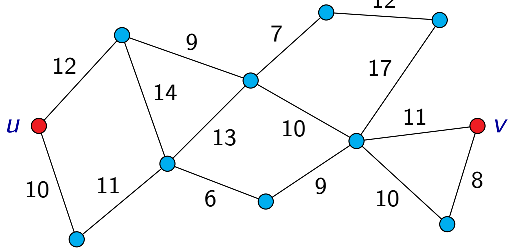
An example of an algorithmic problem

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.
(Or information that there is no such path.)

Example:



Algorithms and Problems

An algorithm **solves** a given problem if:

- For each input, the computation of the algorithm halts after a finite number of steps.
- For each input, the algorithm produces a correct output.

Correctness of an algorithm — verifying that the algorithm really solves the given problem

Computational complexity of an algorithm:

- **time complexity** — how the running time of the algorithm depends on the size of input data
- **space complexity** — how the amount of memory used by the algorithm depends on the size of input data

Remark: For one problem there can be many diffent algorithms that correctly solve the problem.

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: $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots$

Decision Problems

The problems, where the set of outputs is $\{\textsc{Yes}, \textsc{No}\}$ are called **decision problems**.

Decision problems are usually specified in such a way that instead of describing what the output is, a question is formulated.

Example:

Problem "Primality"

Input: A natural number n.

Question: Is n a prime?

Optimization Problems

Those problems where for each input instance there is a corresponding set of **feasible solutions** and where the aim is to select between these feasible solutions that is some respect minimal or maximal (or possibly to find out that there are no feasible solutions), are called **optimization problems**.

Example:

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 $u, v \in V$.

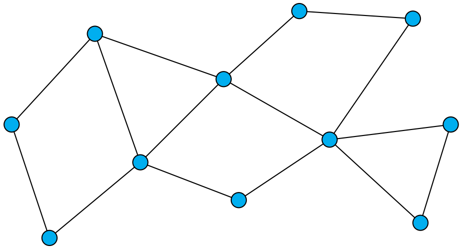
Output: The shortest path from node u to node v.

Optimization Problems

Problem "Coloring of a graph"

Input: An undirected graph G.

Output: The minimal number of colors to color the nodes of the graph G in such a way that no two nodes connected with an edge are colored with the same color, and a concrete example of such coloring using this minimal number of colors.

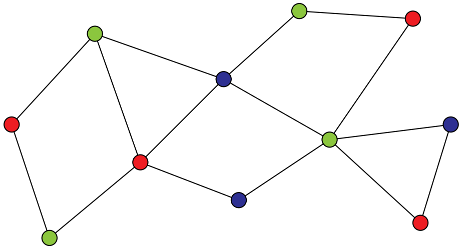


Optimization Problems

Problem "Coloring of a graph"

Input: An undirected graph G.

Output: The minimal number of colors to color the nodes of the graph G in such a way that no two nodes connected with an edge are colored with the same color, and a concrete example of such coloring using this minimal number of colors.



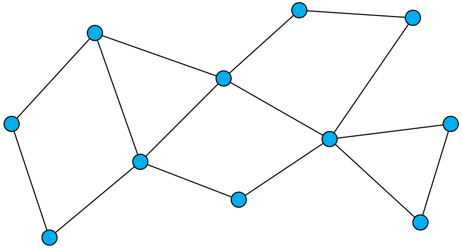
Colors: 3

Optimization 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?



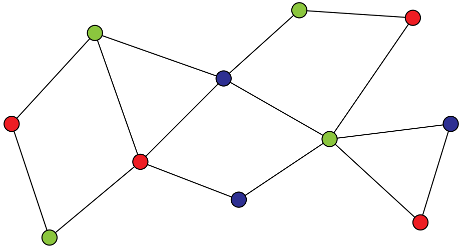
$k = 3$

Optimization 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?



$k = 3$

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 Unsolvable Problems

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

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

Surprisingly, there are many (exactly defined) problems, for which it was proved that they are not algorithmically solvable.

Computability theory — area of theoretical computer science studying, which problems can be solved algorithmically and which cannot.

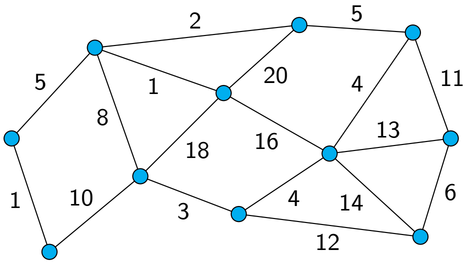
Complexity Theory

Many problems are algorithmically solvable but there do not exist (or are not known) efficient algorithms solving them:

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers.

Output: A shortest closed path that goes through all vertices of the graph.



Theoretical Computer Science

Some other areas of theoretical computer science:

- complexity theory
- theory of formal languages
- models of computation
- parallel and distributed algorithms
- ...

Theory of Formal Languages

An area of theoretical computer science dealing with questions concerning **syntax**.

- **Language** — a set of words
- **Word** — a sequences of symbols from some alphabet
- **Alphabet** — a set of **symbols** (or **letters**)

Words and languages appear in computer science on many levels:

- Representation of input and output data
- Representation of programs
- Manipulation with character strings or files
- ...

Theory of Formal Languages – Motivation

Examples of problem types, where theory of formal languages is useful:

- Construction of compilers:
  - Lexical analysis
  - Syntactic analysis
- Searching in text:
  - Searching for a given text pattern
  - Seaching for a part of text specified by a regular expression

Alphabet, Word

- **Alphabet** — a nonempty finite set of **symbols**

  Example: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

- **Word** — a finite sequence of symbols from the given alphabet

  Example: `cabcbba`

  The set of all words of alphabet Σ is denoted with Σ^*.

  For variables, whose values are words, we will use names such as w, u, v, x, y, z, etc., possibly with indexes (e.g., w_1, w_2)

  So when we write $w = \mathtt{cabcbba}$, it means that the value of variable w is word `cabcbba`.

  Similarly, the notation $w \in \Sigma^*$ means that the value of a variable w is some word consisting of symbols belonging to alphabet Σ.

Formal Languages

Definition

A **(formal) language** L over an alphabet Σ is a subset of Σ^*, i.e., $L \subseteq \Sigma^*$.

Example: Let us assume that $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

- Language $L_1 = \{\, \mathtt{aab},\ \mathtt{bcca},\ \mathtt{aaaaa} \,\}$
- Language $L_2 = \{\, w \in \Sigma^* \mid \text{the number of occurrences of } \mathtt{b} \text{ in } w \text{ is even} \,\}$

Formal Languages

Example:

Alphabet Σ is the set of all ASCII characters.

Example of a word:

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

`#include␣<stdio.h>` $\hookleftarrow\hookleftarrow$ `int␣main()` $\hookleftarrow$ `{` $\hookleftarrow$ `␣␣␣␣printf("He` $\cdots$

Formal Languages

Formalisms used for description of formal languages:

- automata
- grammars
- regular expressions

Encoding of Input and Output

Inputs and outputs of an algorithm could be encoded as words over some alphabet Σ.

Example: For example, for problem "Sorting" we can take alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,\}$.

An example of input data (as a word over alphabet Σ):

`826,13,3901,128,562`

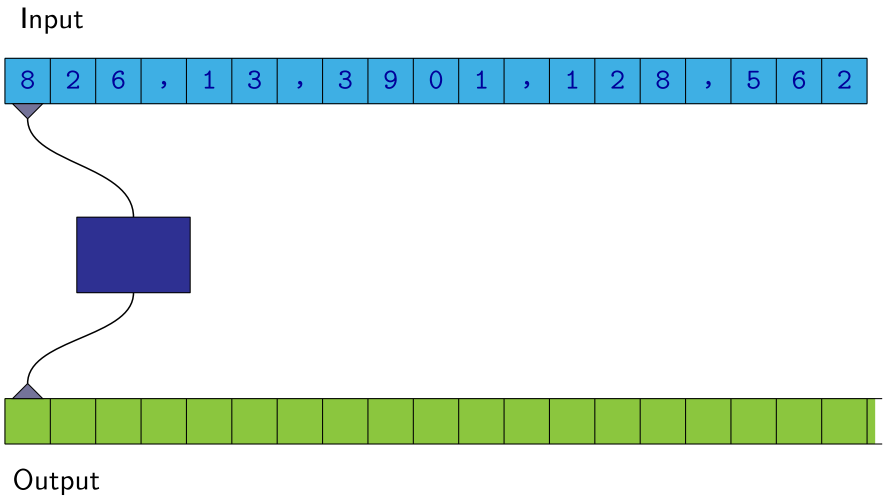
and the corresponding output data (as a word over alphabet Σ)

`13,128,562,826,3901`

Remark: It is often the case that only some words over the given alphabet represent valid input or output.

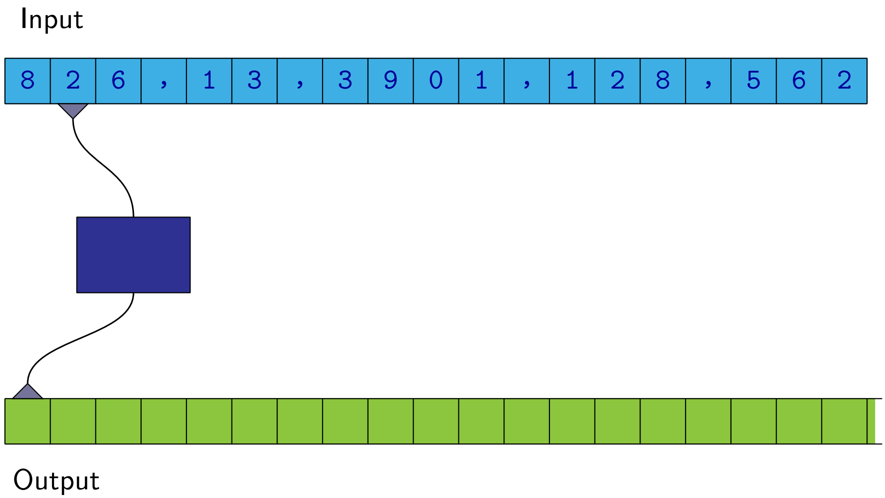
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



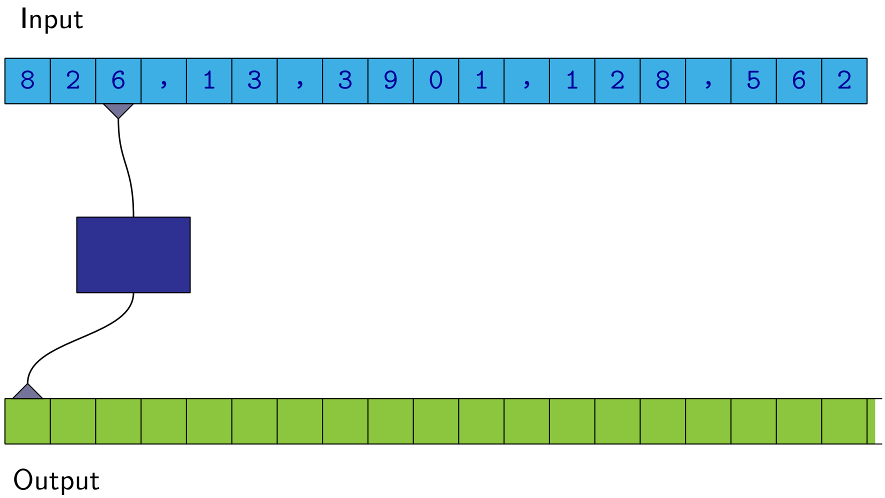
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



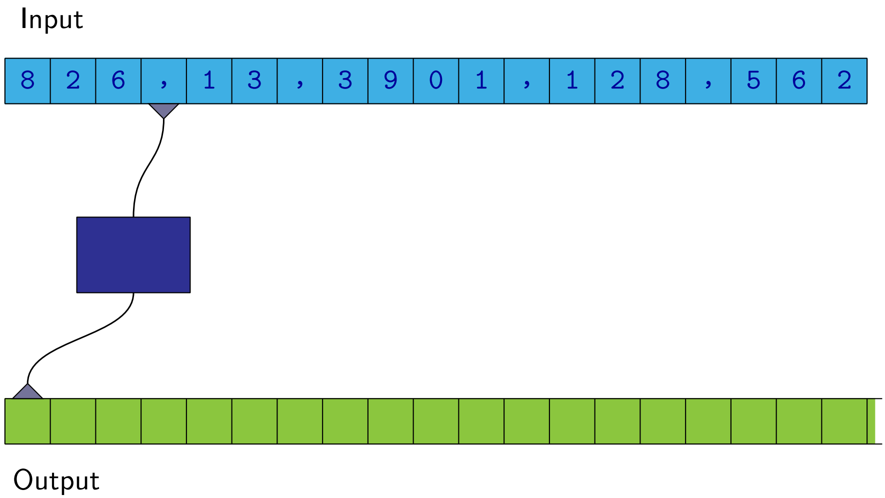
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



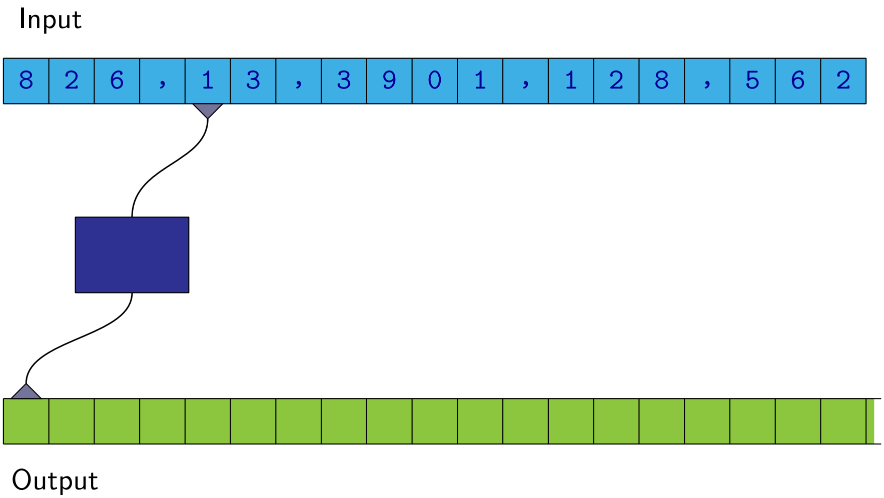
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



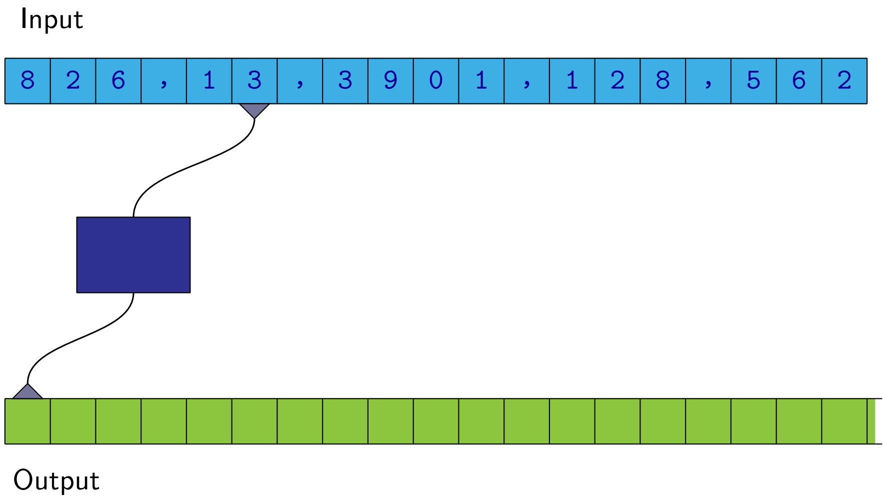
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



Input/Output Behaviour of an Algorithm

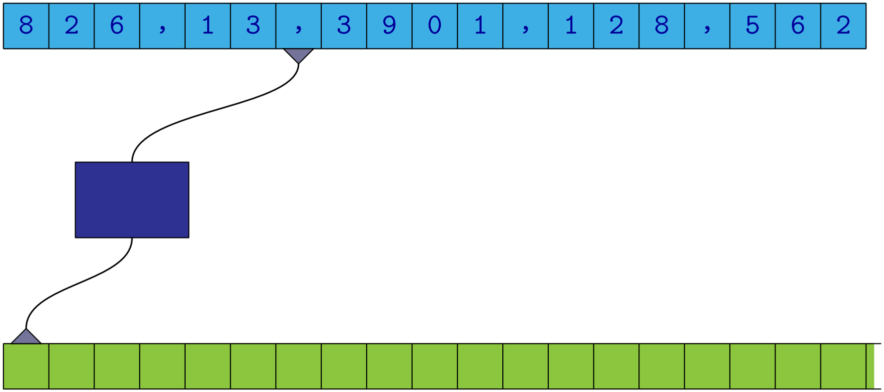
We can assume that the algorithm is executed on a certain type of machine.



Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

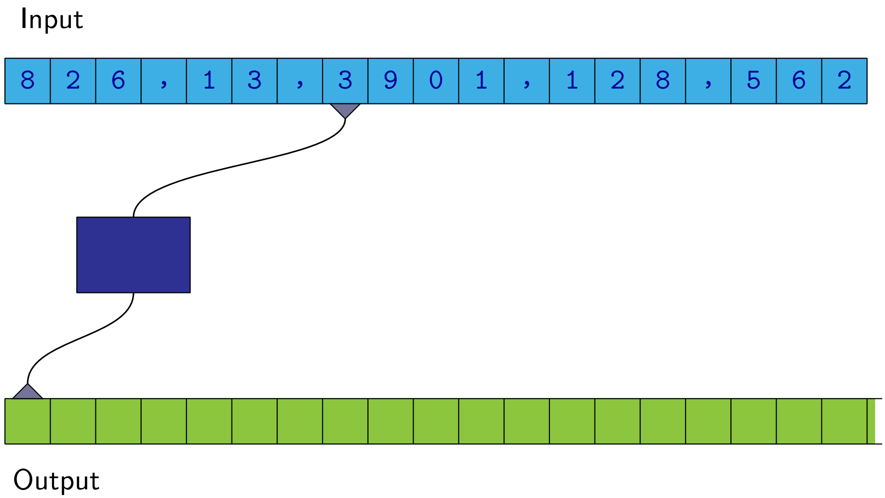
Input



Output

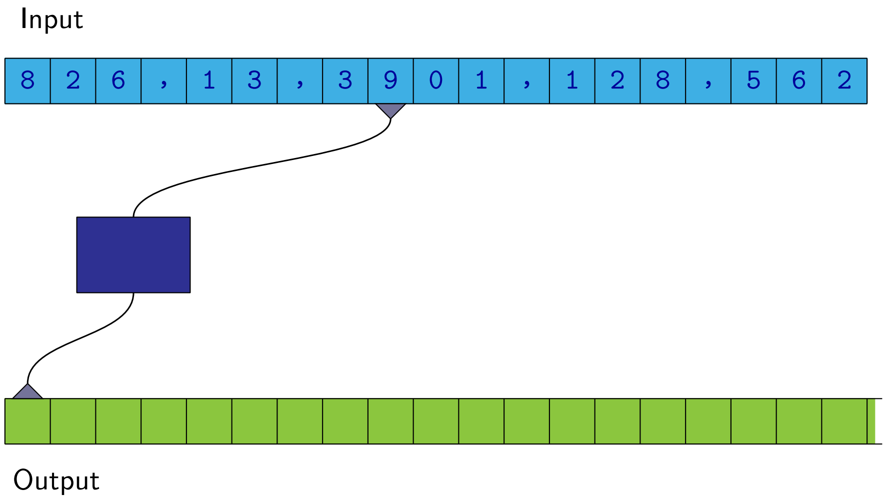
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



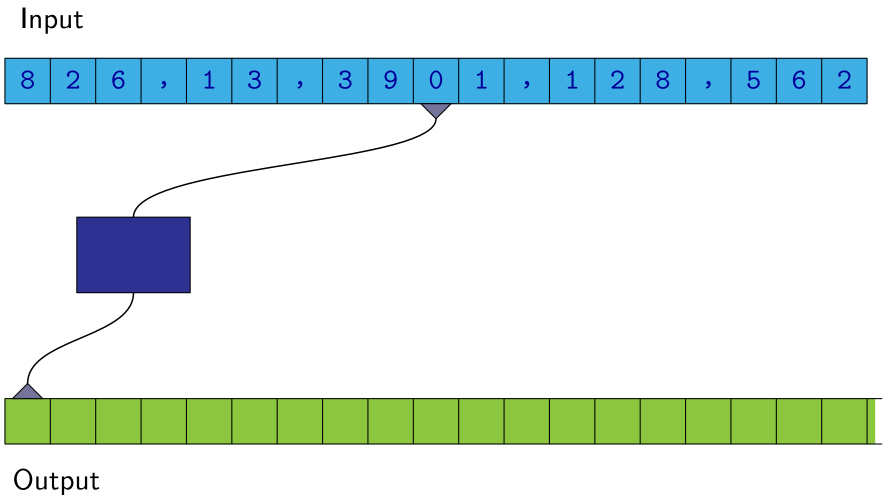
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



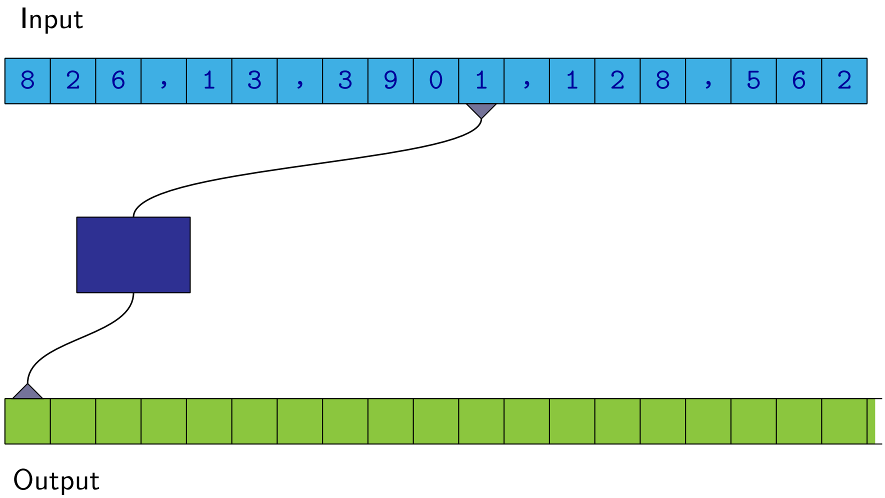
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



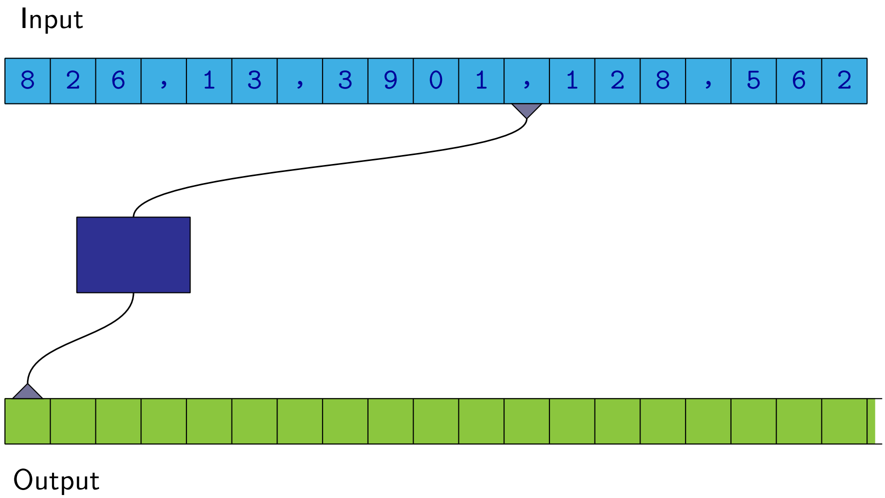
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



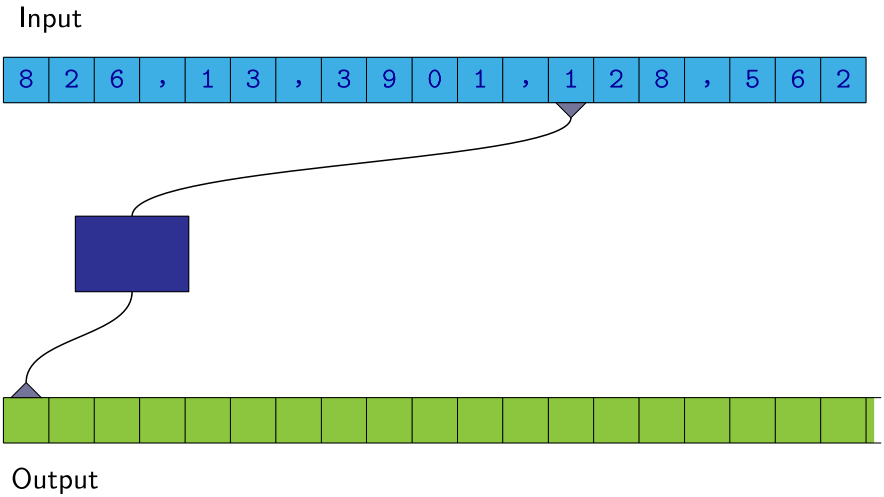
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



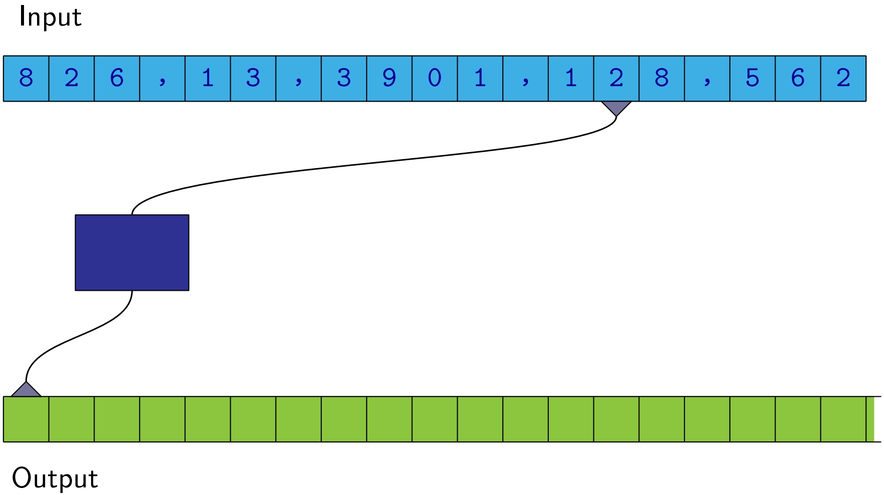
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



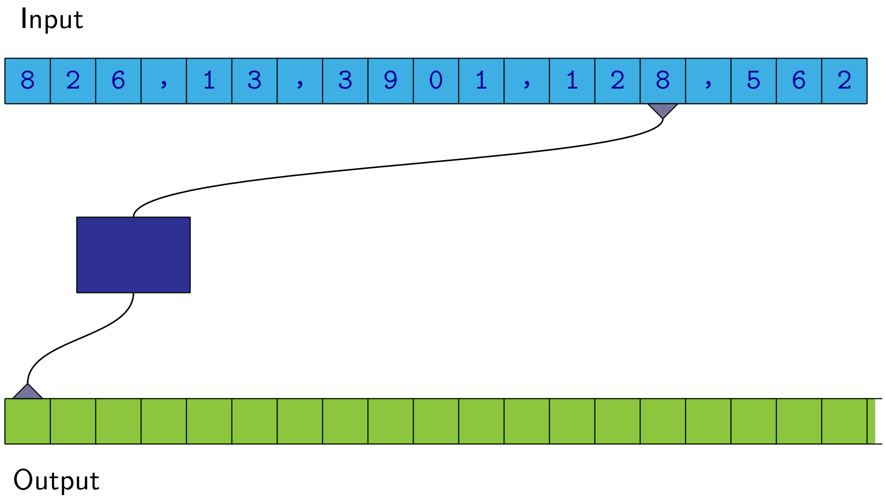
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



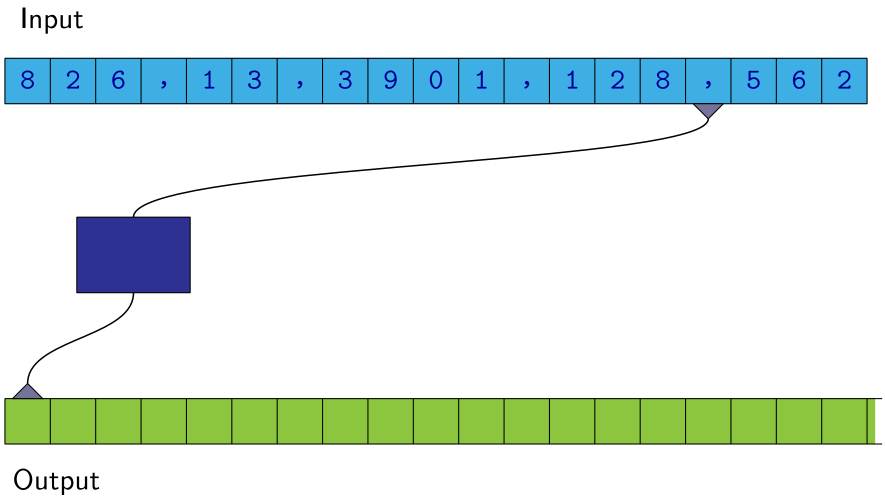
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



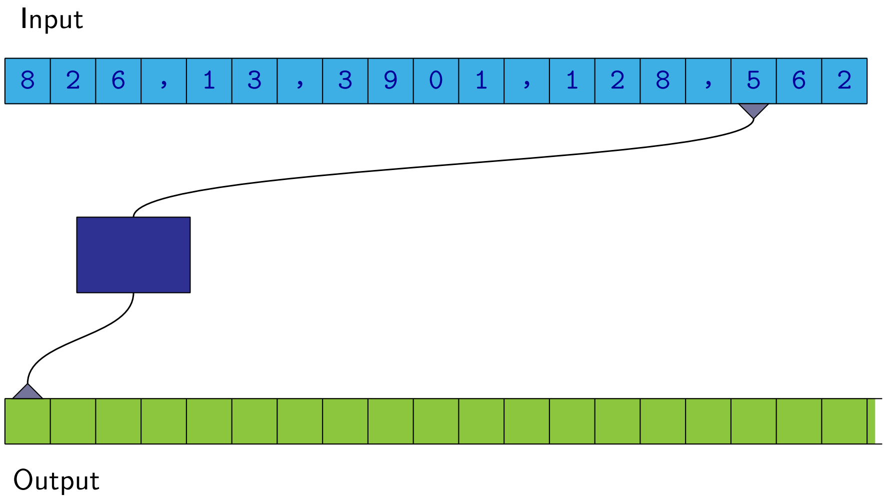
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



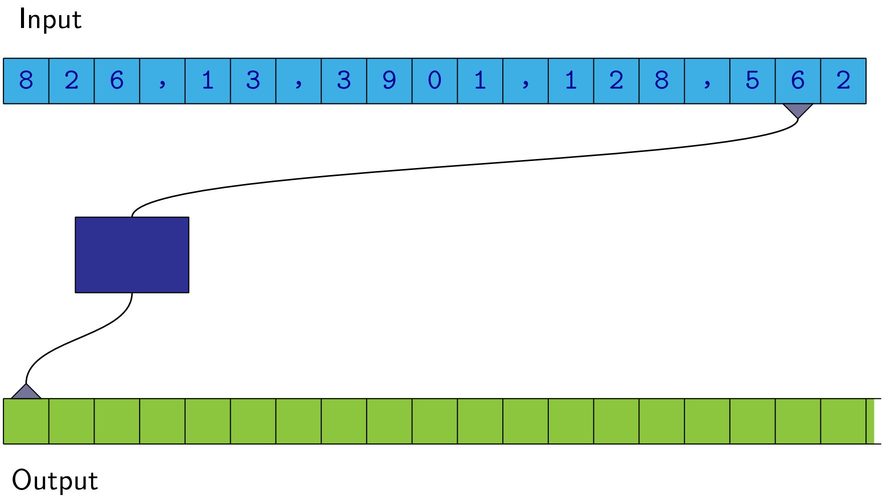
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



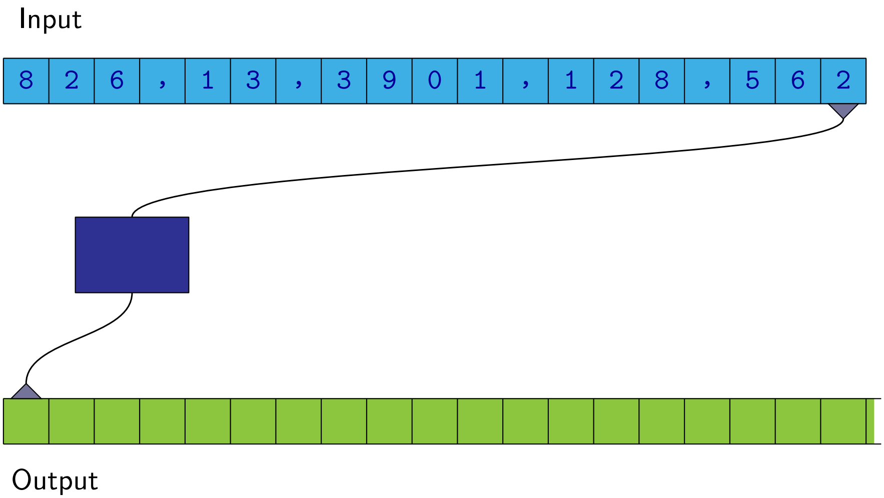
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



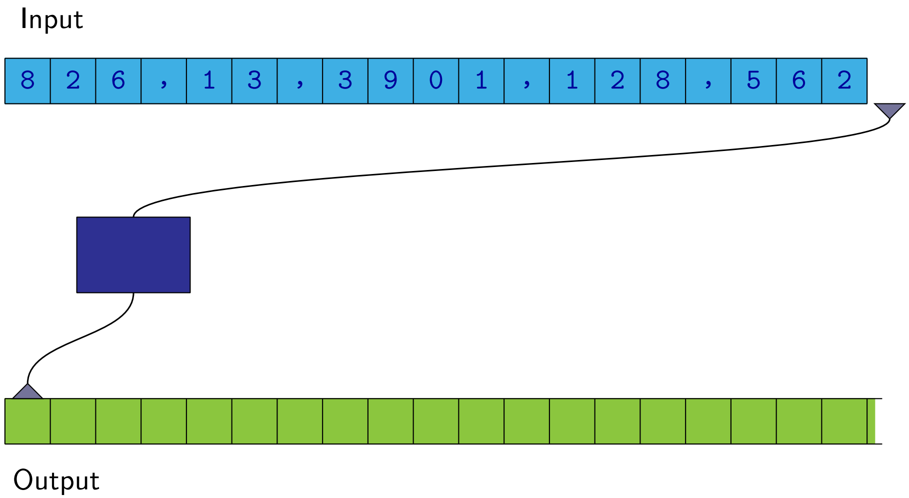
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



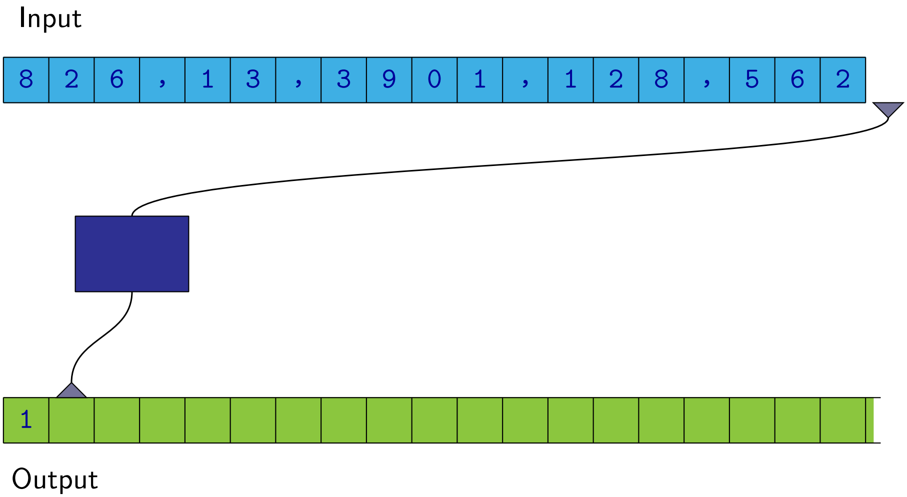
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



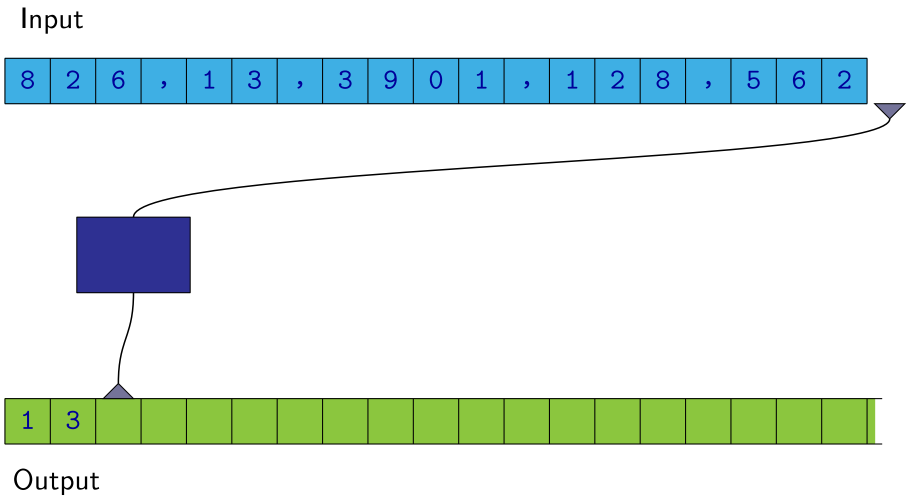
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



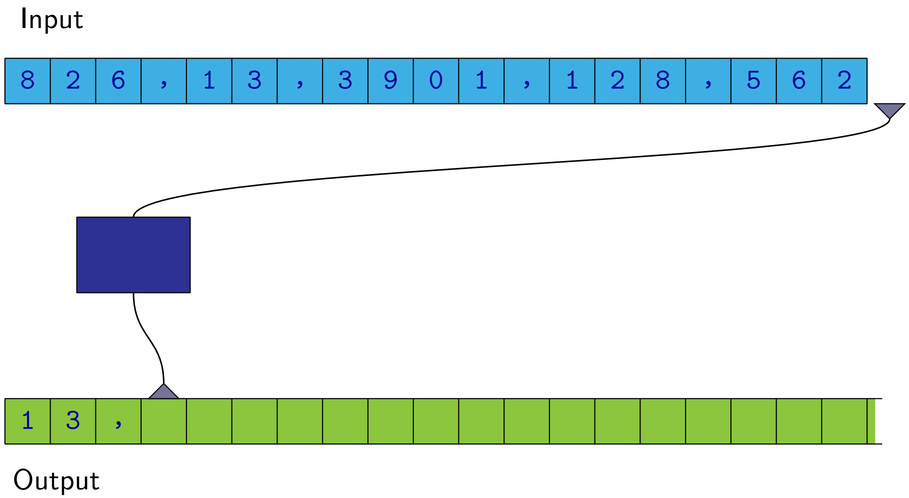
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



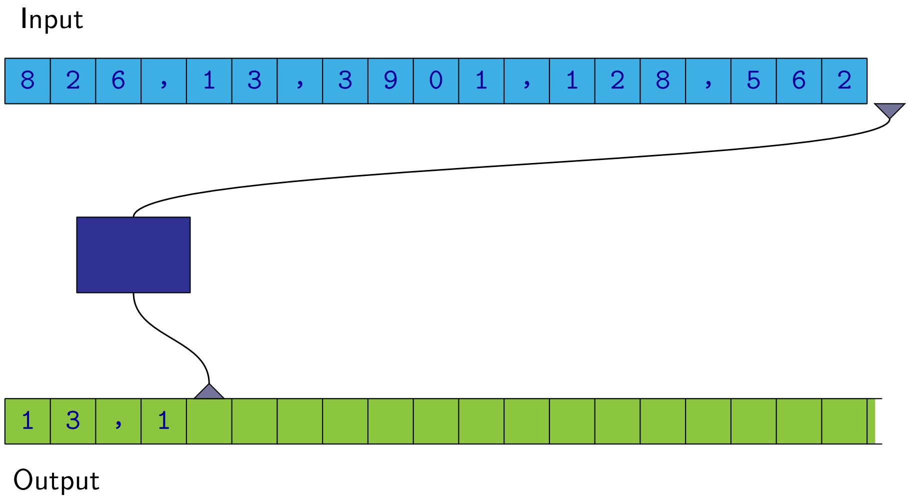
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



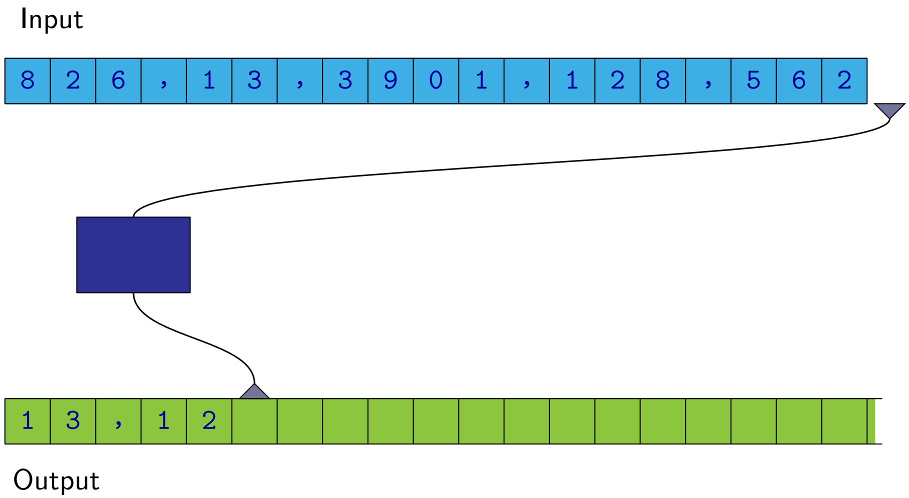
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



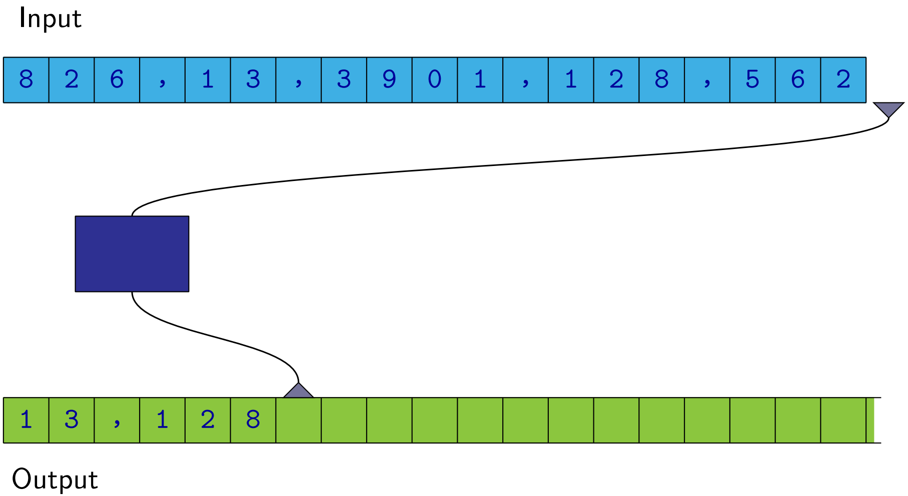
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



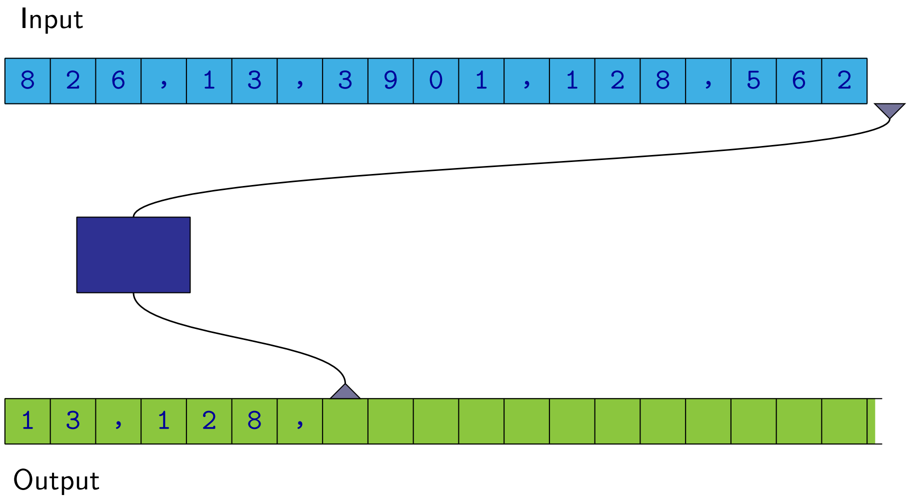
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



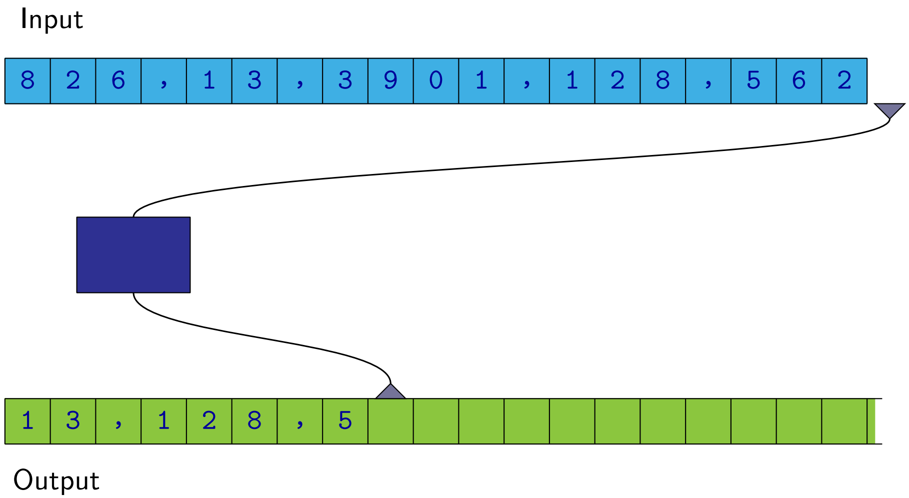
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



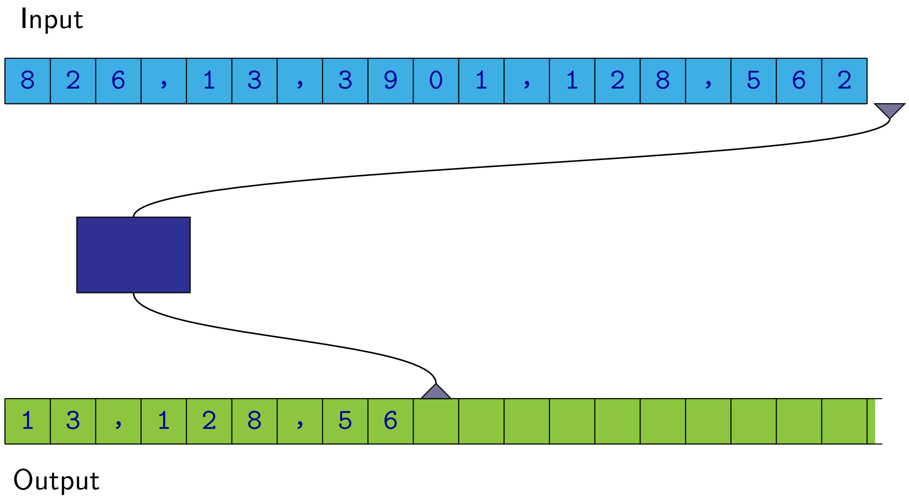
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



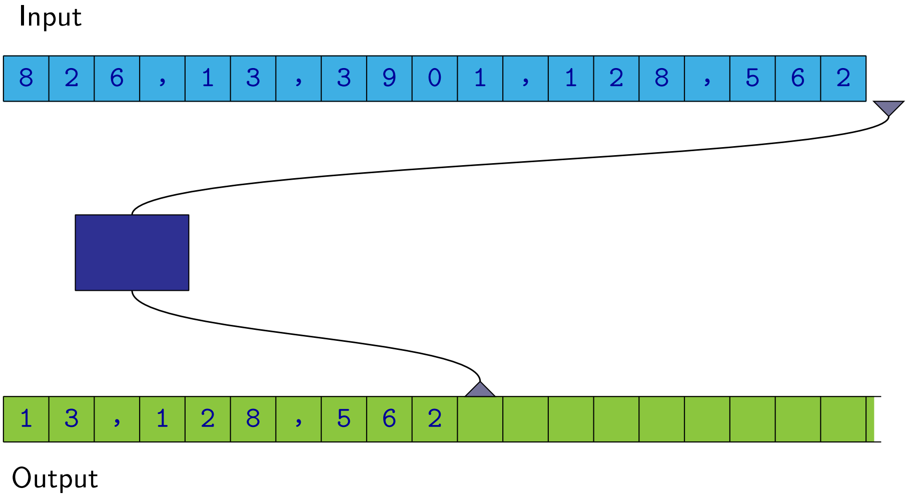
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



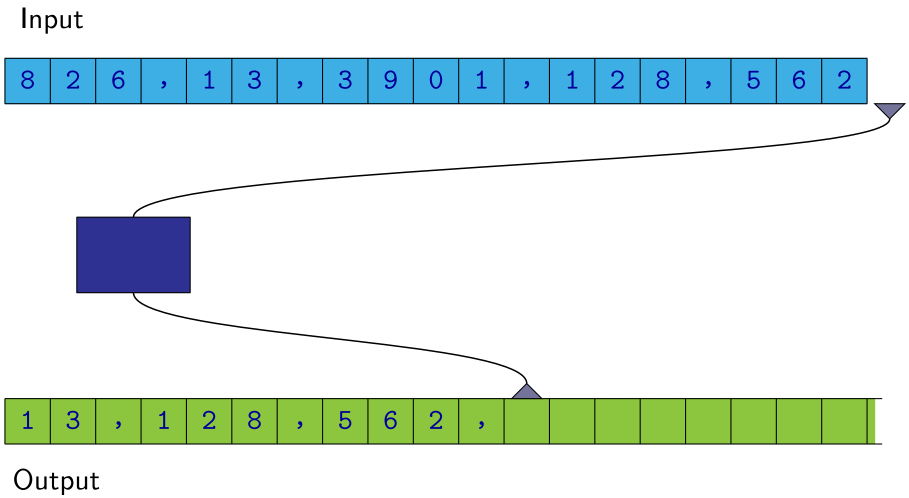
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



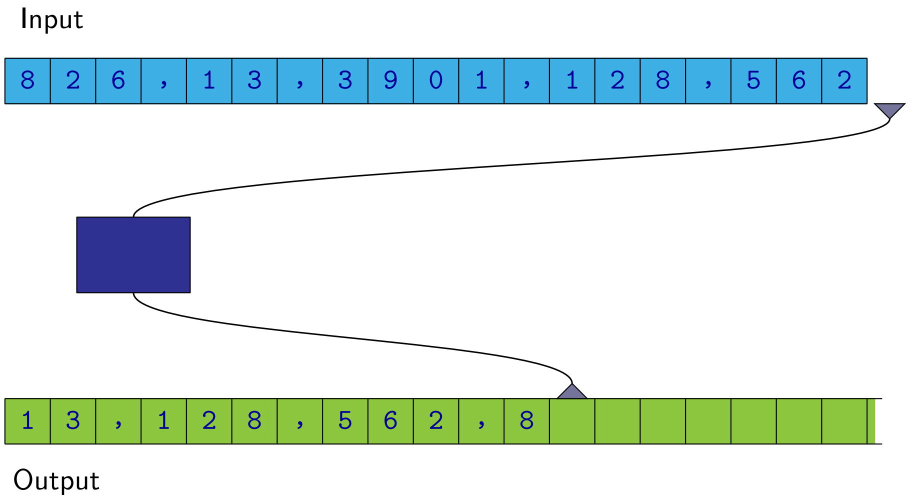
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



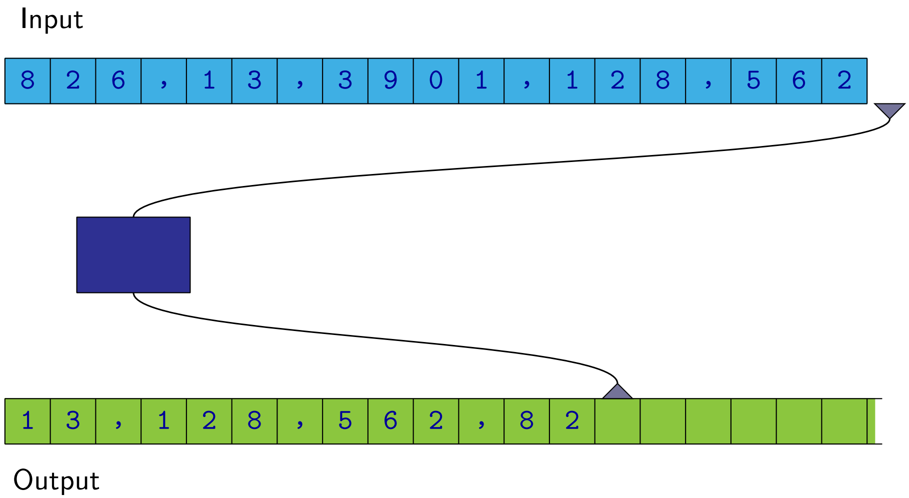
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



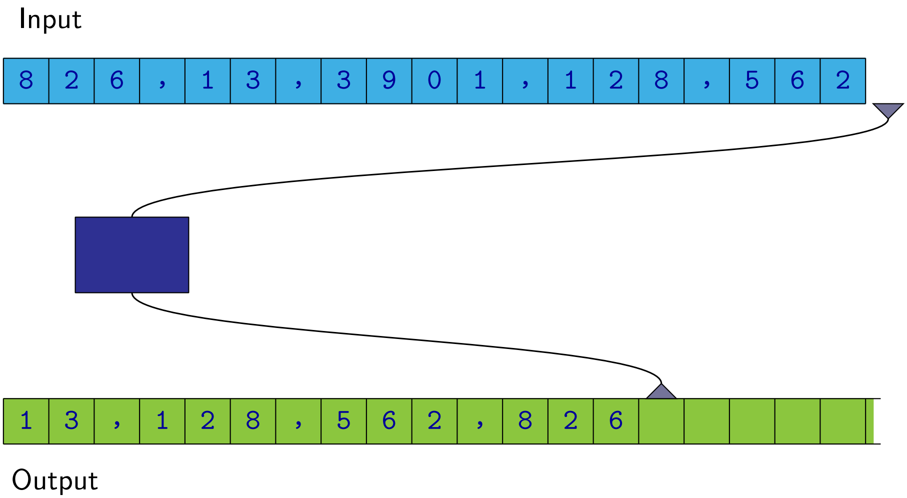
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



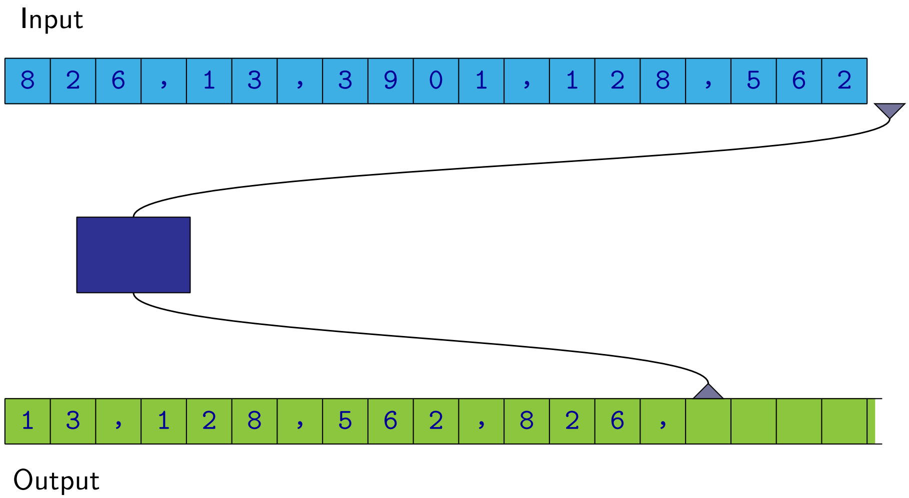
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



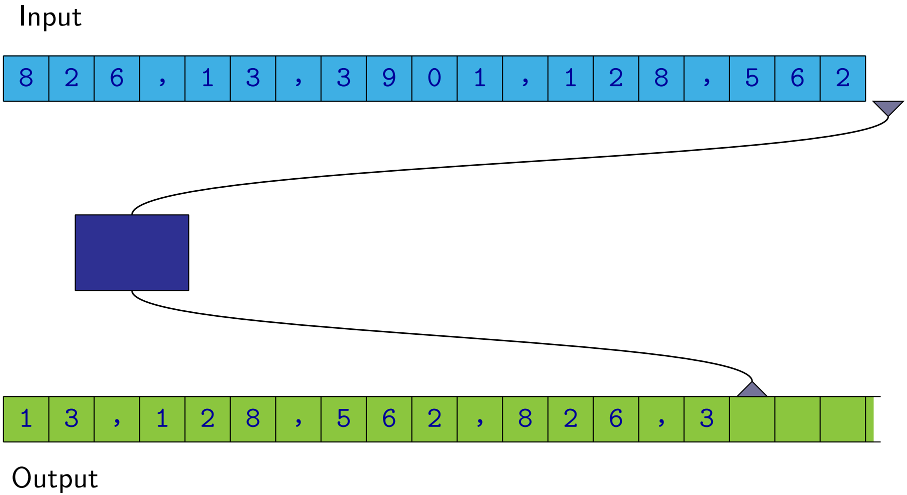
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



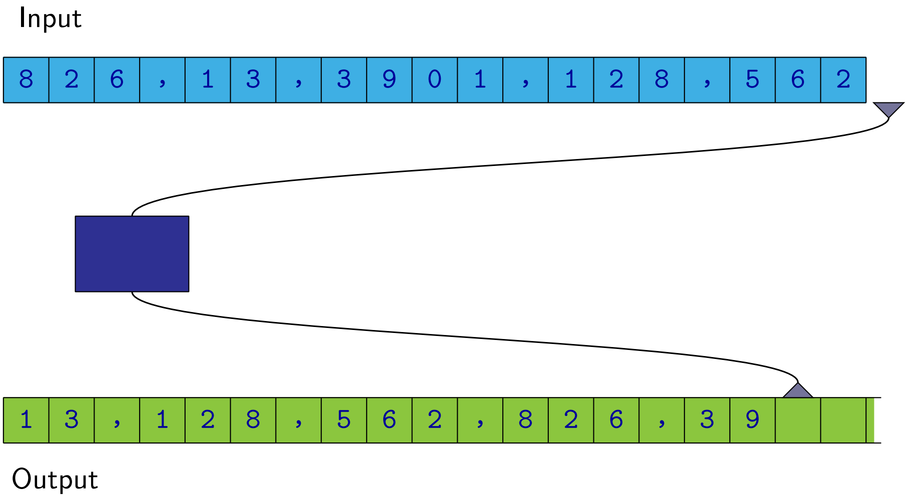
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



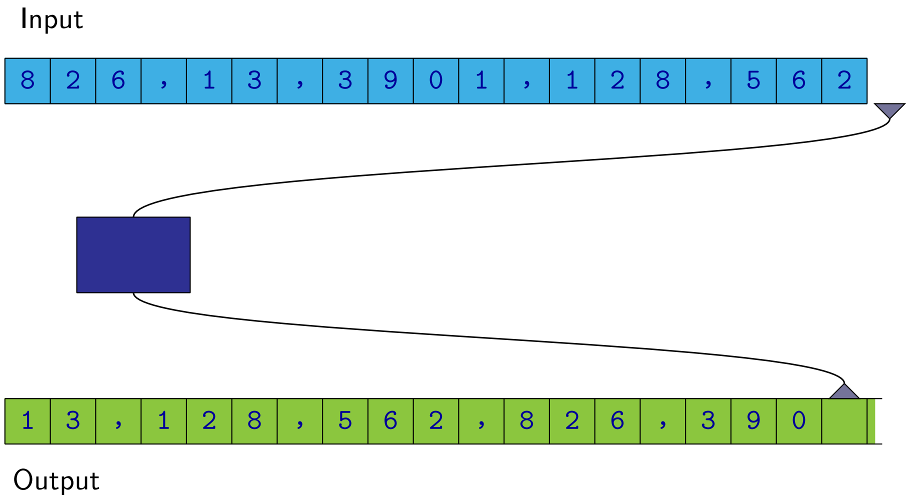
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



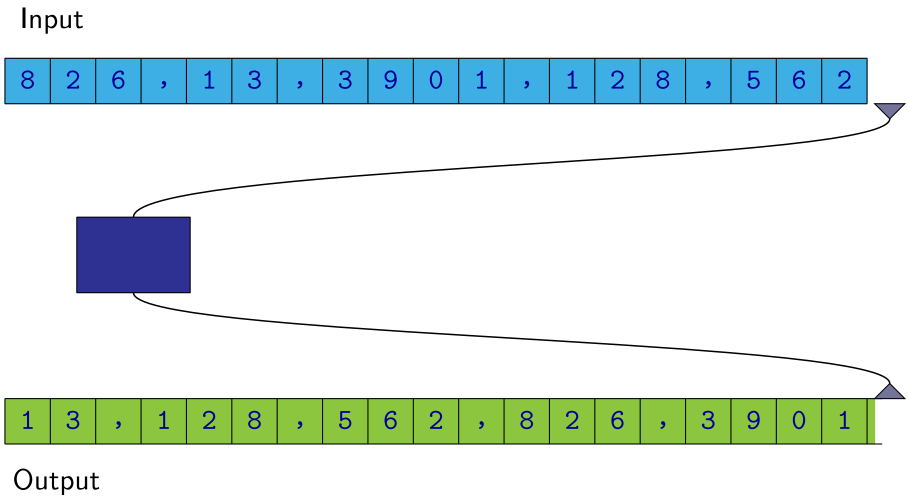
Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.



Input/Output Behaviour of an Algorithm

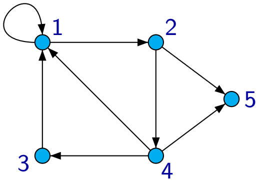
We can assume that the algorithm is executed on a certain type of machine.



Encoding of Input and Output

Example: If an input for a given problem is graph, it could be represented as a pair of two lists — a list of nodes and a list of edges:

For example, the following graph



could be represented as a word

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

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

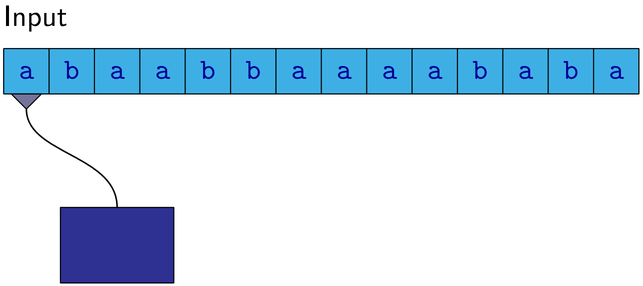
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



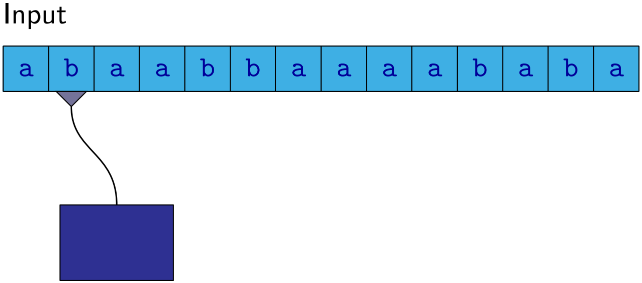
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



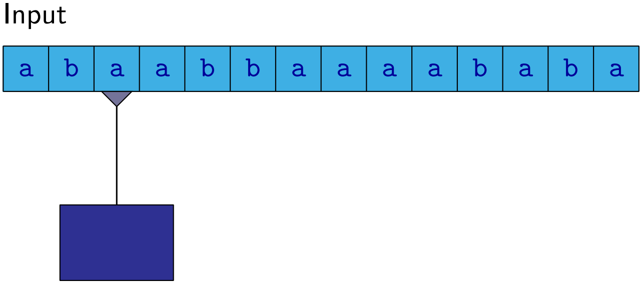
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



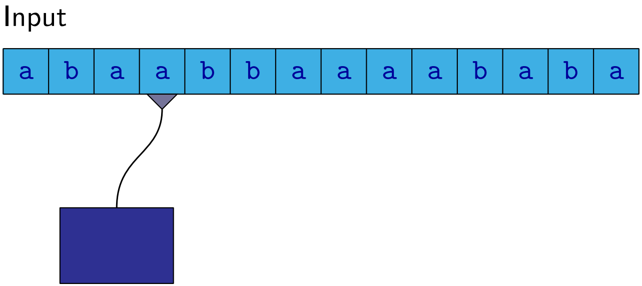
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



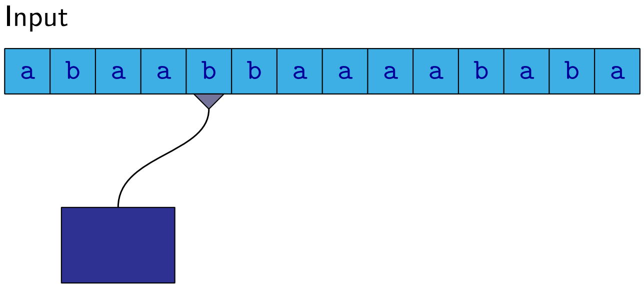
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



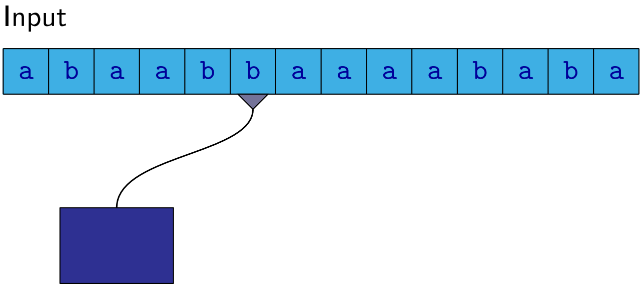
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



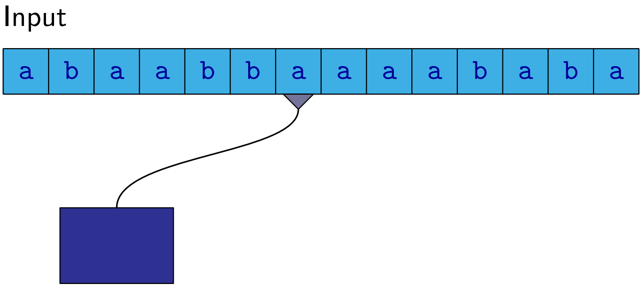
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



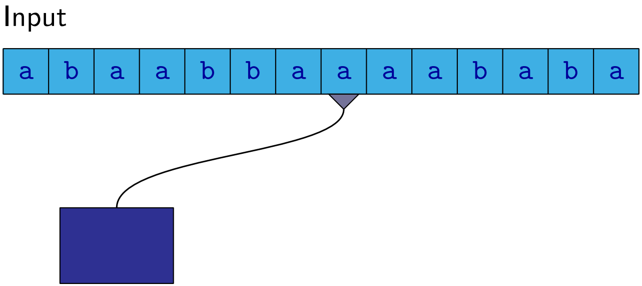
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



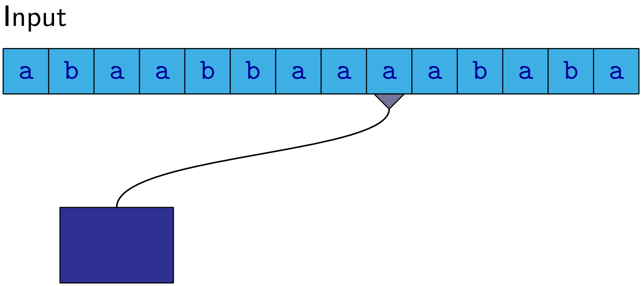
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



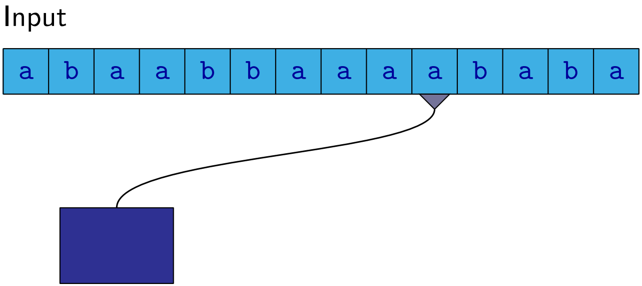
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



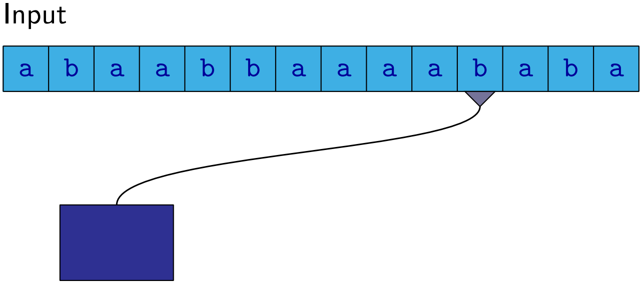
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



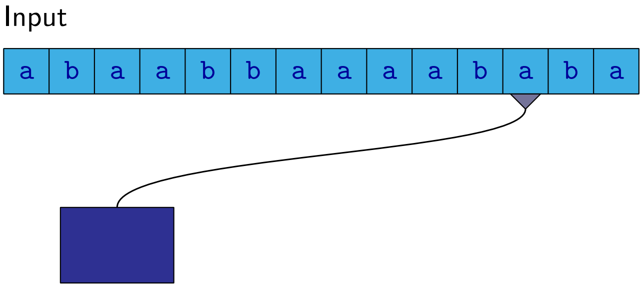
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



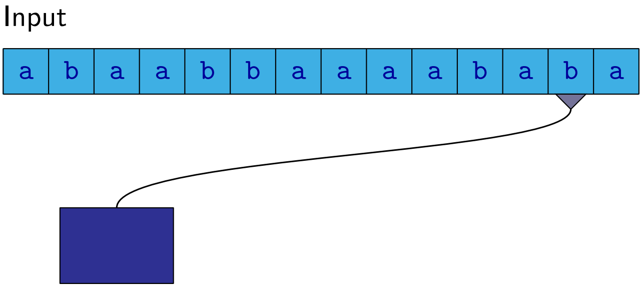
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



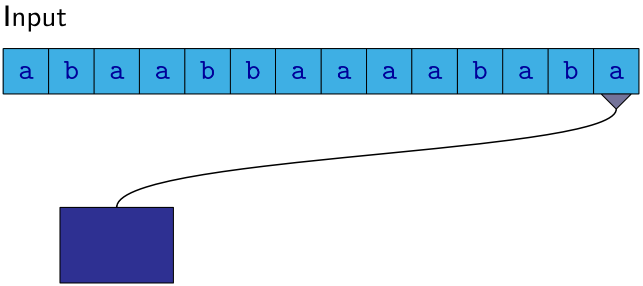
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?



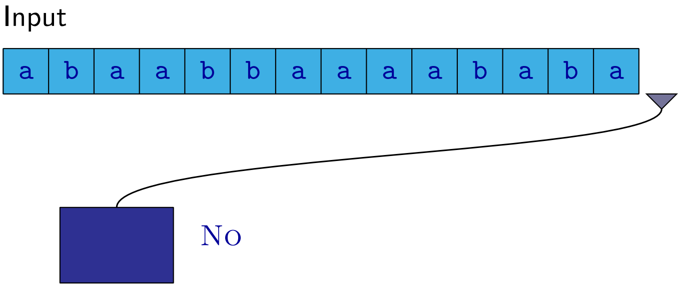
Algorithms for Decision Problems

In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer YES or NO.

Problem

Input: A word w over alphabet $\{a, b\}$.

Question: Does the word w contain an even number of occurrences of symbol b ?



Correspondence between Recognizing Formal Languages and Decision Problems

There is a close correspondence between recognizning words from a given language and decision problems:

- For each language L over some alphabet Σ there is a corresponding decision problem:

Input: A word w over alphabet Σ.

Question: Does w belong to L?

- For each decision problem P where inputs are encoded as words over alphabet Σ there is a corresponding language:

The language L containing of exactly those words w over alphabet Σ, for which the answer to the question stated in problem P is "YES".

Correspondence between Recognizing Formal Languages and Decision Problems

Example: The following decision problem can be viewed as the language L given below and vice versa.

Problem

Input: A word w over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Question: Does the word w contain an even number of occurrences of symbol $\mathtt{b}$?

Language

$L = \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ contains an even number of occurrences of symbol } \mathtt{b} \,\}$

Models of Computation

We can consider different types of machines that are able to perform an algorithm.

There can be many different kinds of differences between these types of machines:

- what types of instructions they can execute
- what types of dates they can store in their memory and this memory is organised
- ...

Different kinds of such machines are called **models of computation**.

In the case of very simple kinds of such machines they are usually called **automata** in the formal language theory.

In this course we will see several types of such automata.

Models of Computation

For different types of models of computation analyse for example:

- what algorithmic problems can be solved by such machines and what languages they can recognise.
- how efficiently they can execute different algorithms
- how machines of a certain type can simulate the computations of some other type of machines
- how the number of instructions that are executed by the machine in such simulaton grows compared to the original machine
- ...

Formal Languages

Alphabet and Word

Definition

Alphabet is a nonempty finite set of **symbols**.

Remark: An alphabet is often denoted by the symbol Σ (upper case sigma) of the Greek alphabet.

Definition

A **word** over a given alphabet is a finite sequence of symbols from this alphabet.

Example 1:

$\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{J}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{U}, \mathtt{V}, \mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$

Words over alphabet Σ: $\mathtt{HELLO}$ $\mathtt{XYZZY}$ $\mathtt{COMPUTER}$

Alphabet and Word

Example 2:

$\Sigma_2 = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{J}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{U}, \mathtt{V}, \mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}, \textvisiblespace\}$

A word over alphabet Σ_2: $\mathtt{HELLO\textvisiblespace WORLD}$

Example 3:

$\Sigma_3 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Words over alphabet Σ_3: 0, 31415926536, 65536

Example 4:

Words over alphabet $\Sigma_4 = \{0, 1\}$: 011010001, 111, 1010101010101010

Example 5:

Words over alphabet $\Sigma_5 = \{\mathtt{a}, \mathtt{b}\}$: $\mathtt{aababb}$, $\mathtt{abbabbba}$, $\mathtt{aaab}$

Language

The **set of all words** over alphabet Σ is denoted Σ^*.

Definition

A **(formal) language** L over an alphabet Σ is a subset of Σ^*, i.e., $L \subseteq \Sigma^*$.

Example 1: The set $\{00, 01001, 1101\}$ is a language over alphabet $\{0, 1\}$.

Example 2: The set of all syntactically correct programs in the C programming language is a language over the alphabet consisting of all ASCII characters.

Example 3: The set of all texts containing the sequence `hello` is a language over alphabet consisting of all ASCII characters.

Some Basic Concepts

The **length of a word** is the number of symbols of the word.

For example, the length of word $\texttt{abaab}$ is 5.

The length of a word w is denoted $|w|$.

For example, if $w = \texttt{abaab}$ then $|w| = 5$.

We denote the number of occurrences of a symbol a in a word w by $|w|_a$.

Example: If $w = \texttt{cabcbba}$ then $|w| = 7$, $|w|_{\texttt{a}} = 2$, $|w|_{\texttt{b}} = 3$, $|w|_{\texttt{c}} = 2$, $|w|_{\texttt{d}} = 0$.

An **empty word** is a word of length 0, i.e., the word containing no symbols.

The empty word is denoted by the letter ε (epsilon) of the Greek alphabet.

$$|\varepsilon| = 0$$

Concatenation of Words

One of operations we can do on words is the operation of **concatenation**:

For example, the concatenation of words `cabc` and `bba` is the word `cabcbba`.

The operation of concatenation is denoted by symbol $\cdot$ (it is similar to multiplication). This symbol can be omitted.

So, for $u, v \in \Sigma^*$, the concatenation of words u and v is written as $u \cdot v$ or just uv.

Example: If $u = \texttt{cabc}$ and $v = \texttt{bba}$, then

$$u \cdot v = \texttt{cabcbba}$$

Remark: Formally, the concatenation of words over alphabet Σ is a fuction of type

$$\Sigma^* \times \Sigma^* \to \Sigma^*$$

Concatenation of Words

Concatenation is **associative**, i.e., for every three words u, v, and w we have

$$(u \cdot v) \cdot w = u \cdot (v \cdot w)$$

which means that we can omit parenthesis when we write multiple concatenations. For example, we can write $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5$ instead of $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$.

Word ε is a neutral element for the operation of concatenation, so for every word w we also have:

$$\varepsilon \cdot w = w \cdot \varepsilon = w$$

Remark: It is obvious that if the given alphabet contains at least two different symbols, the operation of concatenation is not commutative, e.g.,

$$\mathtt{a} \cdot \mathtt{b} \neq \mathtt{b} \cdot \mathtt{a}$$

Power of a Word

For arbitrary word $w \in \Sigma^*$ and arbitrary $k \in \mathbb{N}$ we can define word w^k as the word obtained by concatenating k copies of the word w.

Example: For $w = \texttt{abb}$ it is $w^4 = \texttt{abbabbabbabb}$.

Example: Notation $\texttt{a}^5\texttt{b}^3\texttt{a}^4$ denotes word $\texttt{aaaaabbbaaaa}$.

A little bit more formal definition looks as follows:

$$w^0 = \varepsilon, \qquad w^{k+1} = w^k \cdot w \quad \text{for } k \in \mathbb{N}$$

This means

$$\begin{array}{rcl} w^0 & = & \varepsilon \\ w^1 & = & w \\ w^2 & = & w \cdot w \\ w^3 & = & w \cdot w \cdot w \\ w^4 & = & w \cdot w \cdot w \cdot w \\ w^5 & = & w \cdot w \cdot w \cdot w \cdot w \\ & & \cdots \end{array}$$

Reverse of a Word

The **reverse** of a word w is the word w written from backwards (in the opposite order).

The reverse of a word w is denoted w^R.

Example: $w = \mathtt{abbab}$ $w^R = \mathtt{babba}$

So if $w = a_1 a_2 \cdots a_n$ (where $a_i \in \Sigma$) then $w^R = a_n a_{n-1} \cdots a_1$.

We can define w^R using the following inductively defined function $rev : \Sigma^* \to \Sigma^*$ as the value $rev(w)$.

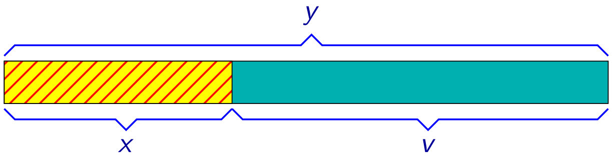
The function rev is defined as follows:

- $rev(\varepsilon) = \varepsilon$
- for $a \in \Sigma$ and $w \in \Sigma^*$ it holds that $rev(a \cdot w) = rev(w) \cdot a$

Prefix of a Word

Definition

A word x is a **prefix** of a word y if there exists a word v such that $y = xv$.

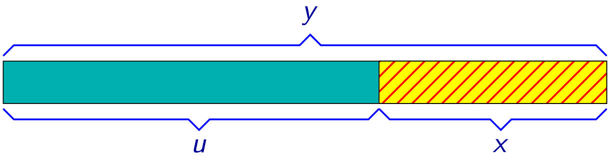


Example: Prefixes of the word `abaab` are ε, `a`, `ab`, `aba`, `abaa`, `abaab`.

Suffix of a Word

Definition

A word x is a **suffix** of a word y if there exists a word u such that $y = ux$.

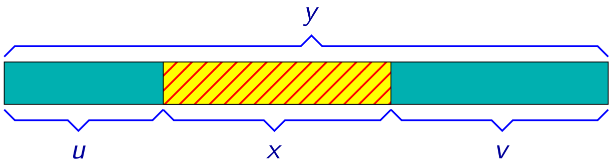


Example: Suffixes of the word `abaab` are ε, `b`, `ab`, `aab`, `baab`, `abaab`.

Subword

Definition

A word x is a **subword** of a word y if there exist words u and v such that $y = uxv$.

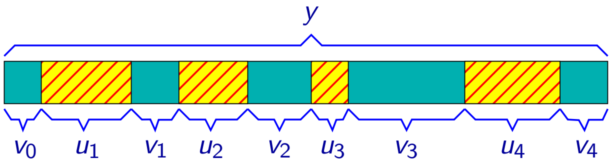


Example: Subwords of the word `abaab` are ε, `a`, `b`, `ab`, `ba`, `aa`, `aba`, `baa`, `aab`, `abaa`, `baab`, `abaab`.

Subsequence

Definition

A word x is a **subsequence** of a word y if there is a number n and words $u_1, u_2, \ldots, u_n$ and $v_0, v_1, \ldots, v_n$ such that $x = u_1 u_2 \cdots u_n$ and $y = v_0 u_1 v_1 u_2 v_2 \cdots u_n v_n$.



Example: Word `cbab` is a subsequence of word `acabccabbaa`.

Order on Words

Let us assume some (linear) order $<$ on the symbols of alphabet Σ, i.e., if $\Sigma = \{a_1, a_2, \ldots, a_n\}$ then

$$a_1 < a_2 < \ldots < a_n \,.$$

Example: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ with $\mathtt{a} < \mathtt{b} < \mathtt{c}$.

The following (linear) order $<_L$ can be defined on Σ^*:
$x <_L y$ iff:

- $|x| < |y|$, or
- $|x| = |y|$ there exist words $u, v, w \in \Sigma^*$ and symbols $a, b \in \Sigma$ such that

$$x = uav \qquad y = ubw \qquad a < b$$

Informally, we can say that in order $<_L$ we order words according to their length, and in case of the same length we order them lexicographically.

Order on Words

All words over alphabet Σ can be ordered by $<_L$ into a sequence

$$w_0, w_1, w_2, \ldots$$

where every word $w \in \Sigma^*$ occurs exactly once, and where for each $i, j \in \mathbb{N}$ it holds that $w_i <_L w_j$ iff $i < j$.

Example: For alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ (where $\mathtt{a} < \mathtt{b} < \mathtt{c}$) , the initial part of the sequence looks as follows:

$$\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{ab}, \mathtt{ac}, \mathtt{ba}, \mathtt{bb}, \mathtt{bc}, \mathtt{ca}, \mathtt{cb}, \mathtt{cc}, \mathtt{aaa}, \mathtt{aab}, \mathtt{aac}, \mathtt{aba}, \mathtt{abb}, \mathtt{abc}, \ldots$$

For example, when we talk about the first ten words of a language $L \subseteq \Sigma^*$, we mean ten words that belong to language L and that are smallest of all words of L according to order $<_L$.

Order on Words

$\boxed{\varepsilon}$
$\boxed{\mathtt{a}}$
$\mathtt{b}$
$\boxed{\mathtt{c}}$
$\boxed{\mathtt{aa}}$
$\mathtt{ab}$
$\boxed{\mathtt{ac}}$
$\mathtt{ba}$
$\boxed{\mathtt{bb}}$
$\mathtt{bc}$
$\boxed{\mathtt{ca}}$
$\mathtt{cb}$
$\boxed{\mathtt{cc}}$
$\boxed{\mathtt{aaa}}$
$\mathtt{aab}$
$\boxed{\mathtt{aac}}$
$\mathtt{aba}$
$\boxed{\mathtt{abb}}$
$\mathtt{abc}$
$\vdots$

Example:

Language
$L = \{\, w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid |w|_{\mathtt{b}} \bmod 2 = 0 \,\}$

Order on Words

ε	1
a	2
b	
c	3
aa	4
ab	
ac	5
ba	
bb	6
bc	
ca	7
cb	
cc	8
aaa	9
aab	
aac	10
aba	
abb	11
abc	
$\vdots$	

Example:

Language
$L = \{\, w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid |w|_{\mathtt{b}} \bmod 2 = 0 \,\}$

Operations on Languages

Let us say we have already described some languages. We can create new languages from these languages using different **operations on languages**.

So a description of a complicated language can be decomposed in such a way that it is described a result of an application of some operations on some simpler languages.

Examples of important operations on languages:

- union
- intersection
- complement
- concatenation
- iteration
- ...

Remark: It is assumed the languages involved in these operations use the same alphabet Σ.

Set Operations on Languages

Since languages are sets, we can apply any set operations to them:

Union – $L_1 \cup L_2$ is the language consisting of the words belonging to language L_1 or to language L_2 (or to both of them).

Intersection – $L_1 \cap L_2$ is the language consisting of the words belonging to language L_1 and also to language L_2.

Complement – $\overline{L_1}$ is the language containing those words from Σ^* that do not belong to L_1.

Difference – $L_1 - L_2$ is the language containing those words of L_1 that do not belong to L_2.

Remark: We assume that $L_1, L_2 \subseteq \Sigma^*$ for some given alphabet Σ.

Set Operations on Languages

Formally:

Union: $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$

Intersection: $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$

Complement: $\overline{L_1} = \{w \in \Sigma^* \mid w \notin L_1\}$

Difference: $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \notin L_2\}$

Set Operations on Languages

Example:

Consider languages over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

- L_1 — the set of all words containing subword `baa`
- L_2 — the set of all words with an even number of occurrences of symbol `b`

Then

- $L_1 \cup L_2$ — the set of all words containing subword `baa` or an even number of occurrences of `b`
- $L_1 \cap L_2$ — the set of all words containing subword `baa` and an even number of occurrences of `b`
- $\overline{L_1}$ — the set of all words that do not contain subword `baa`
- $L_1 - L_2$ — the set of all words that contain subword `baa` but do not contain an even number of occurrences of `b`

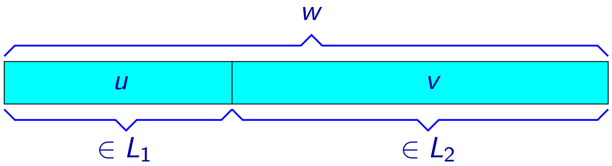
Concatenation of Languages

Definition

Concatenation of languages L_1 and L_2, where $L_1, L_2 \subseteq \Sigma^*$, is the language $L \subseteq \Sigma^*$ such that for each $w \in \Sigma^*$ it holds that

$$w \in L \iff (\exists u \in L_1)(\exists v \in L_2)(w = u \cdot v)$$

The concatenation of languages L_1 and L_2 is denoted $L_1 \cdot L_2$.



Concatenation of Languages

Example:

$$\begin{aligned} L_1 &= \{\mathtt{abb}, \mathtt{ba}\} \\ L_2 &= \{\mathtt{a}, \mathtt{ab}, \mathtt{bbb}\} \end{aligned}$$

The language $L_1 \cdot L_2$ contains the following words:

`abba`   `abbab`   `abbbbb`   `baa`   `baab`   `babbb`

Remark: Note that the concatenation of languages is associative, i.e., for arbitrary languages L_1, L_2, L_3 it holds that:

$$L_1 \cdot (L_2 \cdot L_3) = (L_1 \cdot L_2) \cdot L_3$$

Power of a Language

Notation L^k, where $L \subseteq \Sigma^*$ and $k \in \mathbb{N}$, denotes the concatenation of the form

$$L \cdot L \cdot \quad \cdots \quad \cdot L$$

where the language L occurs k times, i.e.,

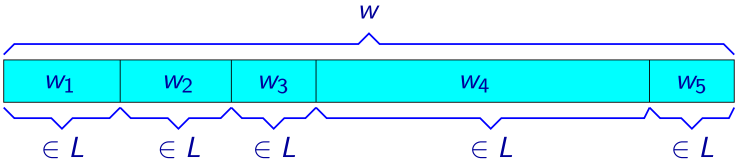
$$\begin{array}{lcl} L^0 & = & \{\varepsilon\} \\ L^1 & = & L \\ L^2 & = & L \cdot L \\ L^3 & = & L \cdot L \cdot L \\ L^4 & = & L \cdot L \cdot L \cdot L \\ L^5 & = & L \cdot L \cdot L \cdot L \cdot L \\ & & \ldots \end{array}$$

Example: For $L = \{\mathtt{aa}, \mathtt{b}\}$, the language L^3 contains the following words:

aaaaaa aaaab aabaa aabb baaaa baab bbaa bbb

Power of a Language

Example: A word in language L^5 is created by concatenating five words from language L:



Formally, the k-th power of a language L, denoted L^k can be defined using the following inductive definition:

$$L^0 = \{\varepsilon\}, \qquad L^{k+1} = L^k \cdot L \quad \text{for } k \in \mathbb{N}$$

Iteration of a Language

The **iteration of a language** L, denoted L^*, is the language consisting of words created by concatenation of some arbitrary number of words from language L.

I.e., a word w belongs to L^* iff there exists a sequence $w_1, w_2, \ldots, w_n$ of words from language L such that

$$w = w_1 w_2 \cdots w_n \,.$$

Example: $L = \{\mathtt{aa}, \mathtt{b}\}$

$$L^* = \{\varepsilon, \mathtt{aa}, \mathtt{b}, \mathtt{aaaa}, \mathtt{aab}, \mathtt{baa}, \mathtt{bb}, \mathtt{aaaaaa}, \mathtt{aaaab}, \mathtt{aabaa}, \mathtt{aabb}, \ldots\}$$

Remark: The number of concatenated words can be 0, which means that $\varepsilon \in L^*$ always holds (it does not matter if $\varepsilon \in L$ or not).

Iteration of a Language

Formally, the language L^* can be defined as the union of all powers of language L. I.e., a word w belongs to the language L^* iff if there exists $k \in \mathbb{N}$ such that $w \in L^k$:

Definition

The **iteration of a language** L is the language

$$L^* = \bigcup_{k \geq 0} L^k$$

Remark:

$$\bigcup_{k \geq 0} L^k = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

Iteration of a Language

Notation L^+ denotes the language consinsting of those words that can be created as a concatenation of a non-zero number of words from language L.

So it holds that

$$L^+ = \bigcup_{k \geq 1} L^k$$

i.e.

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots$$

Formally, the language L^+ can be defined also as follows:

$$L^+ = L \cdot L^*$$

Reverse

The **reverse** of a language L is the language consisting of reverses of all words of L.

Reverse of a language L is denoted L^R.

$$L^R = \{w^R \mid w \in L\}$$

Example: $L = \{\mathtt{ab}, \mathtt{baaba}, \mathtt{aaab}\}$

$L^R = \{\mathtt{ba}, \mathtt{abaab}, \mathtt{baaa}\}$

Some Properties of Operations on Languages

$$\begin{aligned}
L_1 \cup (L_2 \cup L_3) &= (L_1 \cup L_2) \cup L_3 \\
L_1 \cup L_2 &= L_2 \cup L_1 \\
L_1 \cup L_1 &= L_1 \\
L_1 \cup \emptyset &= L_1
\end{aligned}$$

$$\begin{aligned}
L_1 \cap (L_2 \cap L_3) &= (L_1 \cap L_2) \cap L_3 \\
L_1 \cap L_2 &= L_2 \cap L_1 \\
L_1 \cap L_1 &= L_1 \\
L_1 \cap \emptyset &= \emptyset
\end{aligned}$$

$$\begin{aligned}
L_1 \cdot (L_2 \cdot L_3) &= (L_1 \cdot L_2) \cdot L_3 \\
L_1 \cdot \{\varepsilon\} &= L_1 \\
\{\varepsilon\} \cdot L_1 &= L_1 \\
L_1 \cdot \emptyset &= \emptyset \\
\emptyset \cdot L_1 &= \emptyset
\end{aligned}$$

Some Properties of Operations on Languages

$$\begin{aligned}
L_1 \cdot (L_2 \cup L_3) &= (L_1 \cdot L_2) \cup (L_1 \cdot L_3) \\
(L_1 \cup L_2) \cdot L_3 &= (L_1 \cdot L_3) \cup (L_2 \cdot L_3) \\
(L_1^*)^* &= L_1^* \\
\emptyset^* &= \{\varepsilon\} \\
L_1^* &= \{\varepsilon\} \cup (L_1 \cdot L_1^*) \\
L_1^* &= \{\varepsilon\} \cup (L_1^* \cdot L_1) \\
(L_1 \cup L_2)^* &= L_1^* \cdot (L_2 \cdot L_1^*)^* \\
(L_1 \cdot L_2)^R &= L_2^R \cdot L_1^R
\end{aligned}$$

Regular Expressions

Regular Expressions

Regular expressions describing languages over an alphabet Σ:

- $\emptyset$, ε, a (where $a \in \Sigma$) are regular expressions:

  $\emptyset$ . . . denotes the empty language
  ε . . . denotes the language $\{\varepsilon\}$
  a . . . denotes the language $\{a\}$

- If α, β are regular expressions then also $(\alpha + \beta)$, $(\alpha \cdot \beta)$, (α^*) are regular expressions:

  $(\alpha + \beta)$ . . . denotes the union of languages denoted α and β
  $(\alpha \cdot \beta)$ . . . denotes the concatenation of languages denoted α and β
  (α^*) . . . denotes the iteration of a language denoted α

- There are no other regular expressions except those defined in the two points mentioned above.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.
- Since $(0 + 1)$ and (0^*) are regular expressions, $((0 + 1) \cdot (0^*))$ is also a regular expression.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

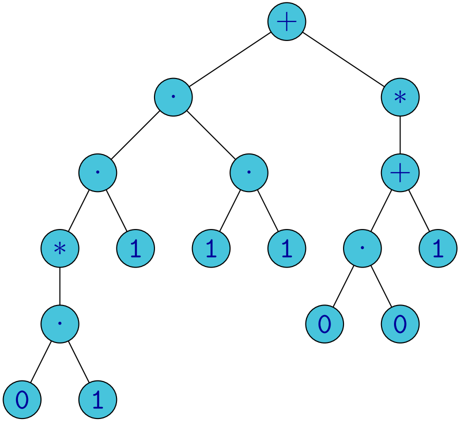
- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.
- Since $(0 + 1)$ and (0^*) are regular expressions, $((0 + 1) \cdot (0^*))$ is also a regular expression.

Remark: If α is a regular expression, by $\mathcal{L}(\alpha)$ we denote the language defined by the regular expression α.

$$\mathcal{L}(((0+1) \cdot (0^*))) = \{0,\ 1,\ 00,\ 10,\ 000,\ 100,\ 0000,\ 1000,\ 00000,\ \ldots\}$$

Regular Expressions

The structure of a regular expression can be represented by an abstract syntax tree:



$$(((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*))$$

Regular Expressions

The formal definition of semantics of regular expressions:

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\mathcal{L}(a) = \{a\}$
- $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$
- $\mathcal{L}(\alpha \cdot \beta) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta)$
- $\mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$

Regular Expressions

To make regular expressions more lucid and succinct, we use the following conventions:

- The outward pair of parentheses can be omitted.
- We can omit parentheses that are superflous due to associativity of operations of union $(+)$ and concatenation $(\cdot)$.
- We can omit parentheses that are superflous due to the defined priority of operators (iteration $({}^*)$ has the highest priority, concatenation $(\cdot)$ has lower priority, and union $(+)$ has the lowest priority).
- A dot denoting concatenation can be omitted.

Example: Instead of

$$(((((0\cdot 1)^*)\cdot 1)\cdot(1\cdot 1))+(((0\cdot 0)+1)^*))$$

we usually write

$$(01)^*111+(00+1)^*$$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$ . . . the language containing the only word $\mathtt{a}$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

`a`	. . . the language containing the only word `a`
`ab`	. . . the language containing the only word `ab`

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$	... the language containing the only word $\mathtt{a}$
$\mathtt{ab}$	... the language containing the only word $\mathtt{ab}$
$\mathtt{a} + \mathtt{b}$	... the language containing two words $\mathtt{a}$ and $\mathtt{b}$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$	… the language containing the only word $\mathtt{a}$
$\mathtt{ab}$	… the language containing the only word $\mathtt{ab}$
$\mathtt{a} + \mathtt{b}$	… the language containing two words $\mathtt{a}$ and $\mathtt{b}$
$\mathtt{a}^*$	… the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, …

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$	. . . the language containing the only word $\mathtt{a}$
$\mathtt{ab}$	. . . the language containing the only word $\mathtt{ab}$
$\mathtt{a} + \mathtt{b}$	. . . the language containing two words $\mathtt{a}$ and $\mathtt{b}$
$\mathtt{a}^*$	. . . the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, . . .
$(\mathtt{ab})^*$	. . . the language containing words ε, $\mathtt{ab}$, $\mathtt{abab}$, $\mathtt{ababab}$, . . .

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$	... the language containing the only word $\mathtt{a}$
$\mathtt{ab}$	... the language containing the only word $\mathtt{ab}$
$\mathtt{a} + \mathtt{b}$	... the language containing two words $\mathtt{a}$ and $\mathtt{b}$
$\mathtt{a}^*$	... the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, ...
$(\mathtt{ab})^*$	... the language containing words ε, $\mathtt{ab}$, $\mathtt{abab}$, $\mathtt{ababab}$, ...
$(\mathtt{a} + \mathtt{b})^*$	... the language containing all words over the alphabet $\{\mathtt{a}, \mathtt{b}\}$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$	... the language containing the only word $\mathtt{a}$
$\mathtt{ab}$	... the language containing the only word $\mathtt{ab}$
$\mathtt{a} + \mathtt{b}$	... the language containing two words $\mathtt{a}$ and $\mathtt{b}$
$\mathtt{a}^*$	... the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, ...
$(\mathtt{ab})^*$	... the language containing words ε, $\mathtt{ab}$, $\mathtt{abab}$, $\mathtt{ababab}$, ...
$(\mathtt{a} + \mathtt{b})^*$	... the language containing all words over the alphabet $\{\mathtt{a}, \mathtt{b}\}$
$(\mathtt{a} + \mathtt{b})^*\mathtt{aa}$	... the language containing all words ending with $\mathtt{aa}$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$	... the language containing the only word $\mathtt{a}$
$\mathtt{ab}$	... the language containing the only word $\mathtt{ab}$
$\mathtt{a} + \mathtt{b}$	... the language containing two words $\mathtt{a}$ and $\mathtt{b}$
$\mathtt{a}^*$	... the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, ...
$(\mathtt{ab})^*$	... the language containing words ε, $\mathtt{ab}$, $\mathtt{abab}$, $\mathtt{ababab}$, ...
$(\mathtt{a} + \mathtt{b})^*$	... the language containing all words over the alphabet $\{\mathtt{a}, \mathtt{b}\}$
$(\mathtt{a} + \mathtt{b})^*\mathtt{aa}$	... the language containing all words ending with $\mathtt{aa}$
$(\mathtt{ab})^*\mathtt{bbb}(\mathtt{ab})^*$	... the language containing all words that contain a subword $\mathtt{bbb}$ preceded and followed by an arbitrary number of copies of the word $\mathtt{ab}$

Regular Expressions

$(a + b)^* aa + (ab)^* bbb (ab)^*$ ... the language containing all words that either end with aa or contain a subwords bbb preceded and followed with some arbitrary number of words ab

Regular Expressions

$(\mathtt{a}+\mathtt{b})^*\mathtt{aa}+(\mathtt{ab})^*\mathtt{bbb}(\mathtt{ab})^*$ . . . the language containing all words that either end with aa or contain a subwords bbb preceded and followed with some arbitrary number of words ab

$(\mathtt{a}+\mathtt{b})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$ . . . the language of all words that contain at least one occurrence of symbol b

Regular Expressions

$(\mathtt{a}+\mathtt{b})^*\mathtt{aa}+(\mathtt{ab})^*\mathtt{bbb}(\mathtt{ab})^*$ ... the language containing all words that either end with aa or contain a subwords bbb preceded and followed with some arbitrary number of words ab

$(\mathtt{a}+\mathtt{b})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$ ... the language of all words that contain at least one occurrence of symbol b

$\mathtt{a}^*(\mathtt{b}\mathtt{a}^*\mathtt{b}\mathtt{a}^*)^*$ ... the language containg all words with an even number of occurrences of symbol b

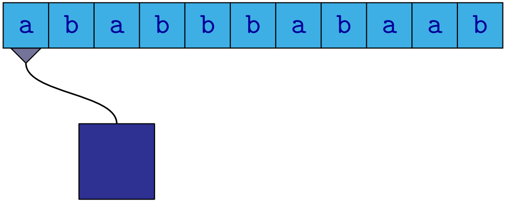
Finite Automata

Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

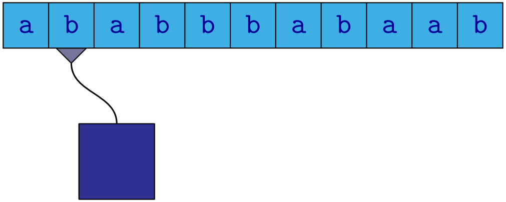


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

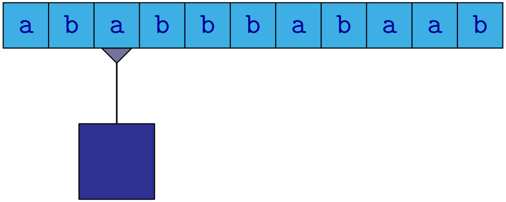


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

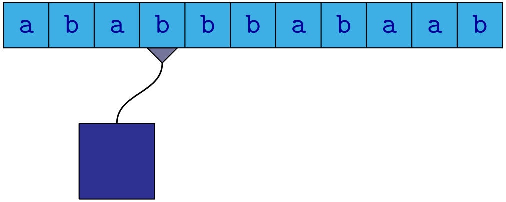


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

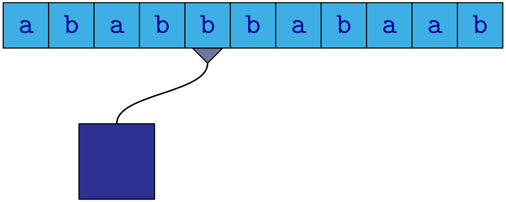


Recognition of a Language

Example: Consider words over alphabet $\{\texttt{a}, \texttt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\texttt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

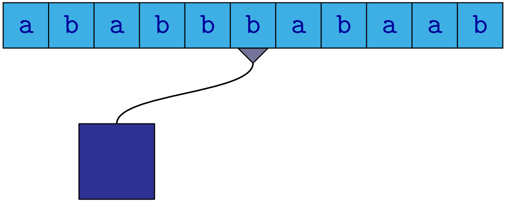


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols b.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

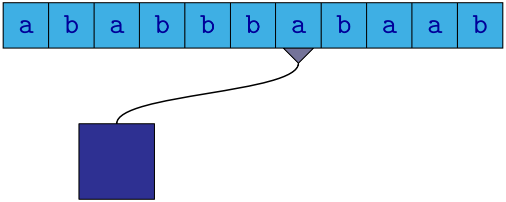


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

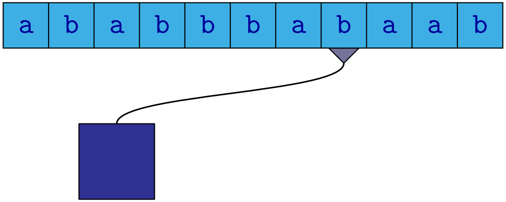


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

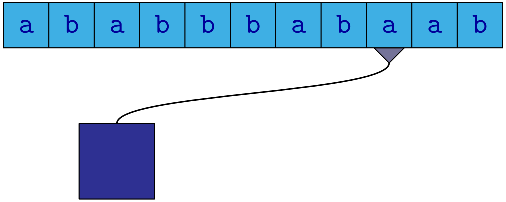


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

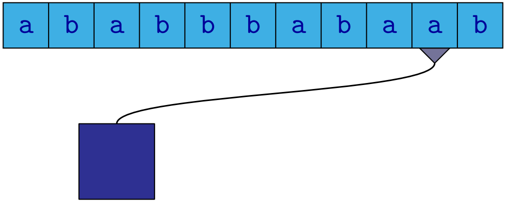


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols $\mathtt{b}$.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

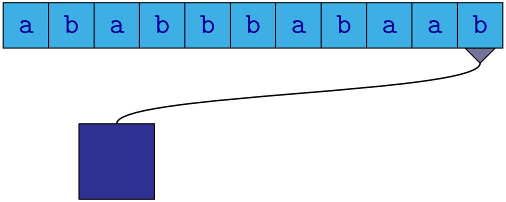


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We would like to recognize a language L consisting of words with even number of symbols b.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.

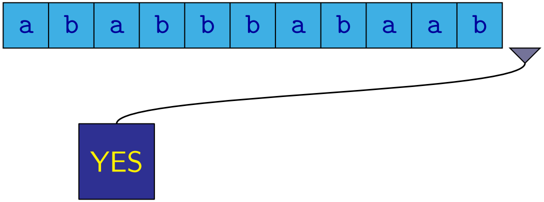


Recognition of a Language

Example: Consider words over alphabet $\{\mathtt{a}, \mathtt{b}\}$.

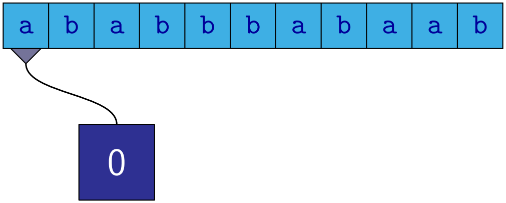
We would like to recognize a language L consisting of words with even number of symbols b.

We want to design a device that reads a word and then tells us if the word belongs to the language L or not.



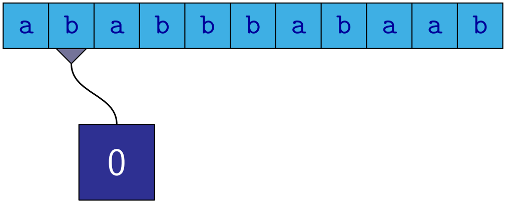
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



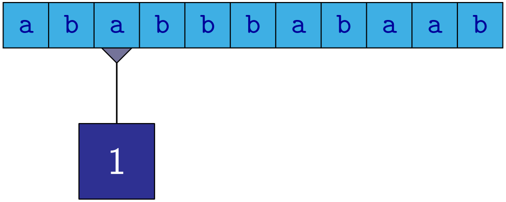
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



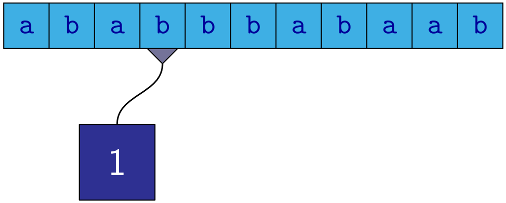
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



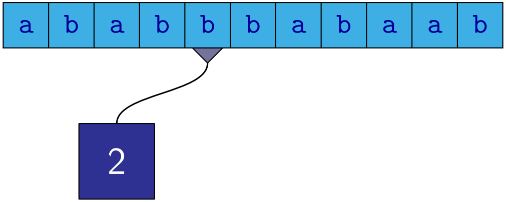
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



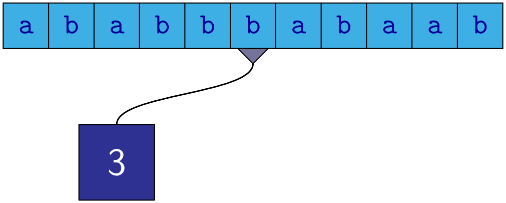
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



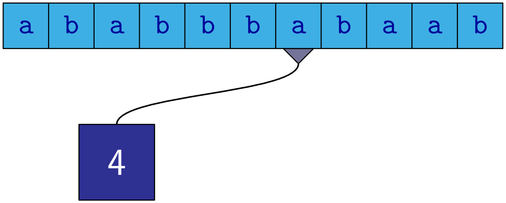
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



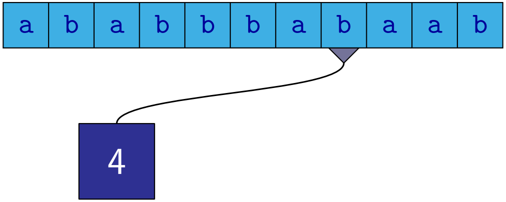
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



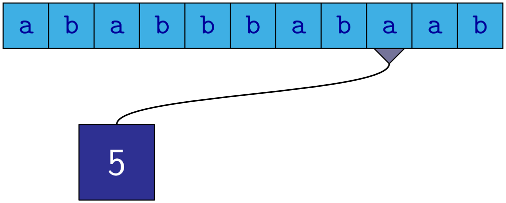
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



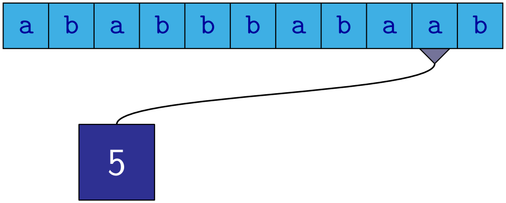
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



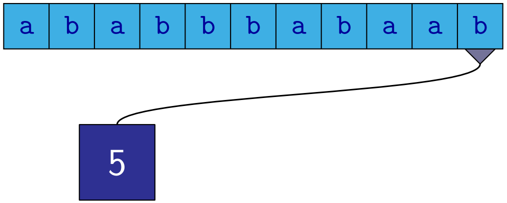
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



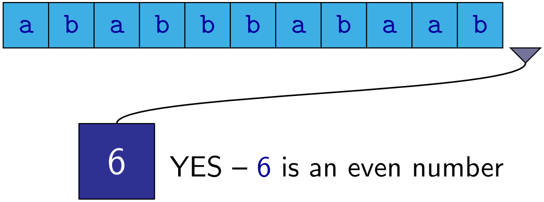
Recognition of a Language

The first idea: To count the number of occurrences of symbol b.



Recognition of a Language

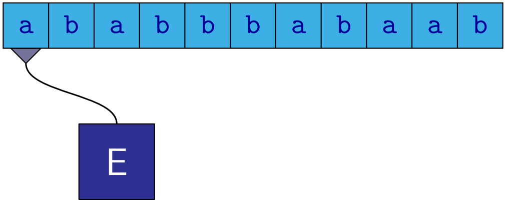
The first idea: To count the number of occurrences of symbol b.



YES – 6 is an even number

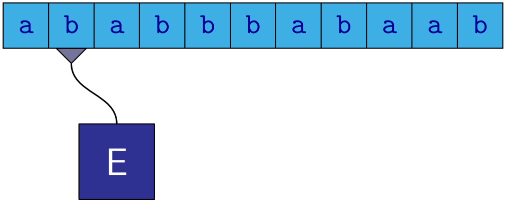
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



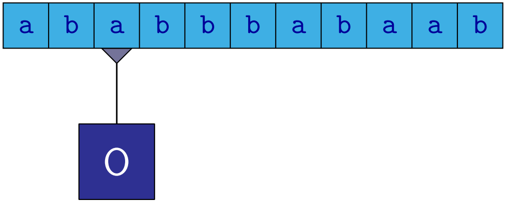
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



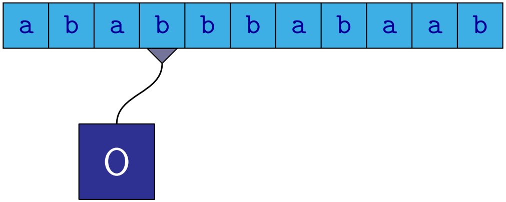
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



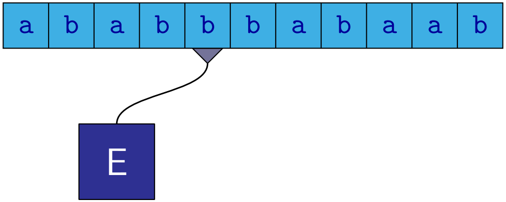
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



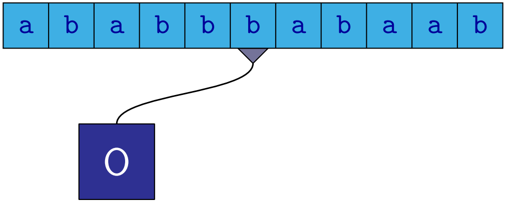
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



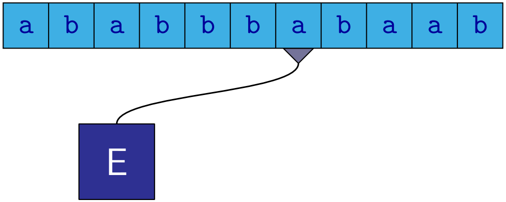
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



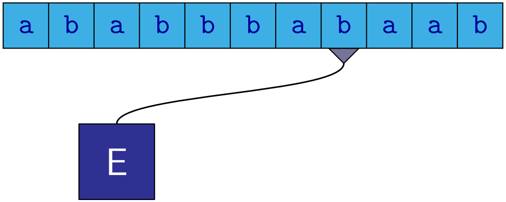
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols `b` read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



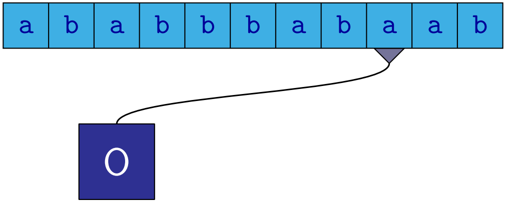
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



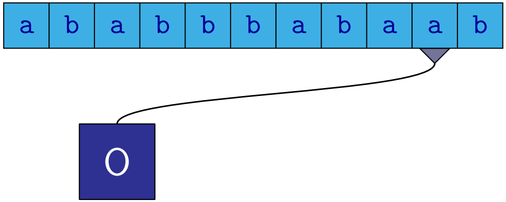
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



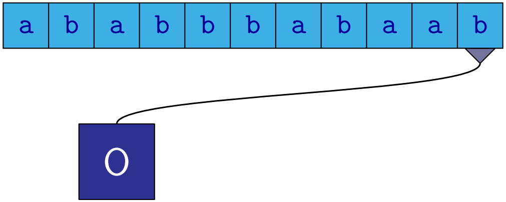
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



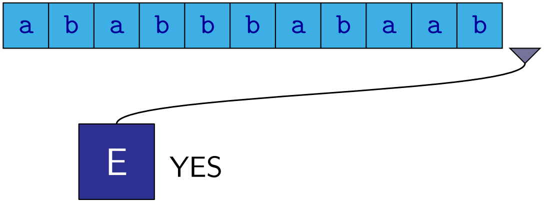
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols b read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).

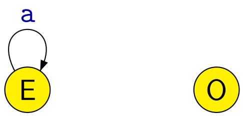


Recognition of a Language

The behaviour of the device can be described by the following graph:

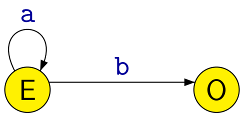
Recognition of a Language

The behaviour of the device can be described by the following graph:



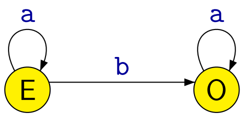
Recognition of a Language

The behaviour of the device can be described by the following graph:



Recognition of a Language

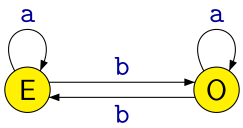
The behaviour of the device can be described by the following graph:



$$\begin{tikzcd}
E \arrow[loop above, "a"] \arrow[r, "b"] & O \arrow[loop above, "a"]
\end{tikzcd}$$

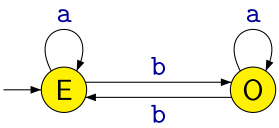
Recognition of a Language

The behaviour of the device can be described by the following graph:


$$\begin{tikzcd}
E \arrow[loop above, "a"] \arrow[r, bend left, "b"] & O \arrow[loop above, "a"] \arrow[l, bend left, "b"]
\end{tikzcd}$$

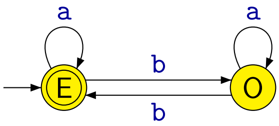
Recognition of a Language

The behaviour of the device can be described by the following graph:



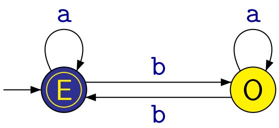
Recognition of a Language

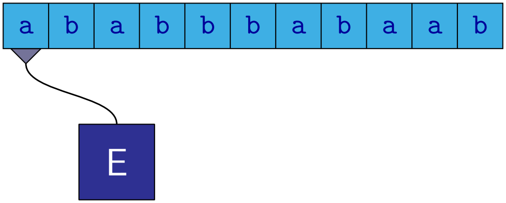
The behaviour of the device can be described by the following graph:


$$\begin{tikzcd}
\arrow[r] & E \arrow[loop above, "a"] \arrow[r, shift left, "b"] & O \arrow[loop above, "a"] \arrow[l, shift left, "b"]
\end{tikzcd}$$

Recognition of a Language

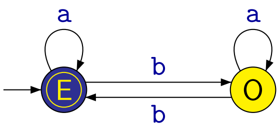
The behaviour of the device can be described by the following graph:

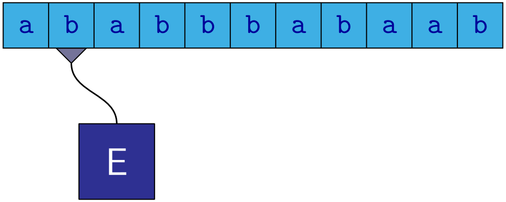




Recognition of a Language

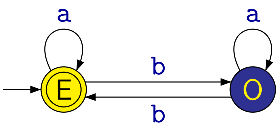
The behaviour of the device can be described by the following graph:

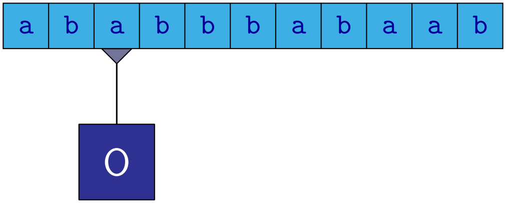

$$\begin{tikzcd}
\arrow[r] & E \arrow[loop above, "a"] \arrow[rr, bend left=10, "b"] & & O \arrow[loop above, "a"] \arrow[ll, bend left=10, "b"]
\end{tikzcd}$$



Recognition of a Language

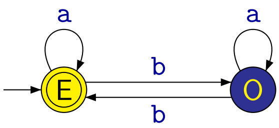
The behaviour of the device can be described by the following graph:

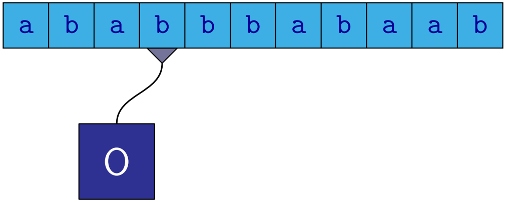




Recognition of a Language

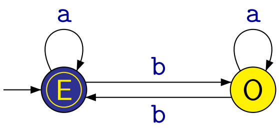
The behaviour of the device can be described by the following graph:

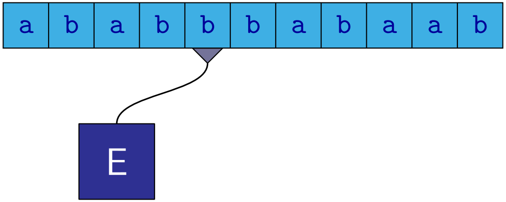

$$\begin{tikzcd} \arrow[r] & E \arrow[loop above, "a"] \arrow[r, shift left, "b"] & O \arrow[loop above, "a"] \arrow[l, shift left, "b"] \end{tikzcd}$$



Recognition of a Language

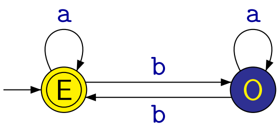
The behaviour of the device can be described by the following graph:

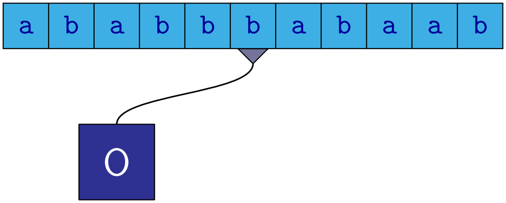




Recognition of a Language

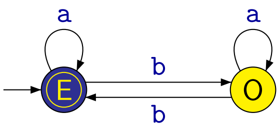
The behaviour of the device can be described by the following graph:

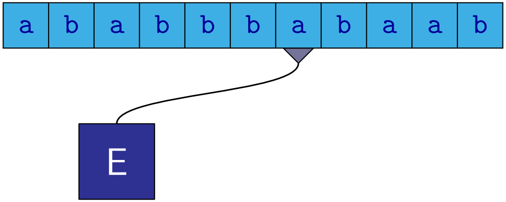




Recognition of a Language

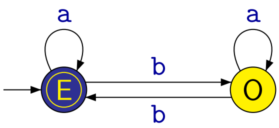
The behaviour of the device can be described by the following graph:


$$\begin{tikzcd}
\arrow[r] & E \arrow[loop above, "a"] \arrow[r, bend left, "b"] & O \arrow[loop above, "a"] \arrow[l, bend left, "b"]
\end{tikzcd}$$

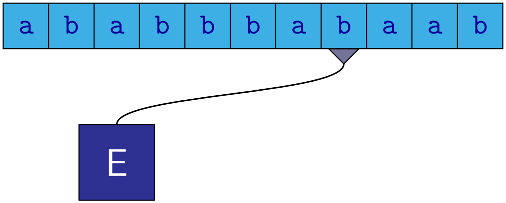


Recognition of a Language

The behaviour of the device can be described by the following graph:

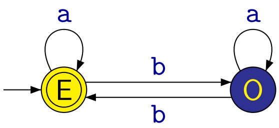


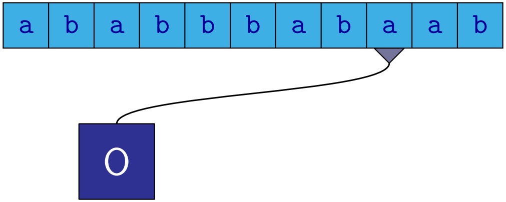
$$\begin{tikzcd}
\arrow[r] & E \arrow[loop above, "a"] \arrow[r, bend left, "b"] & O \arrow[loop above, "a"] \arrow[l, bend left, "b"]
\end{tikzcd}$$



Recognition of a Language

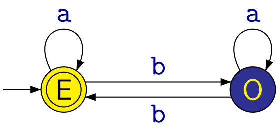
The behaviour of the device can be described by the following graph:

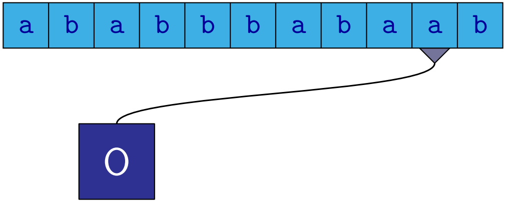




Recognition of a Language

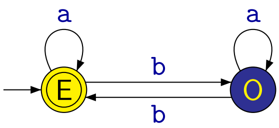
The behaviour of the device can be described by the following graph:

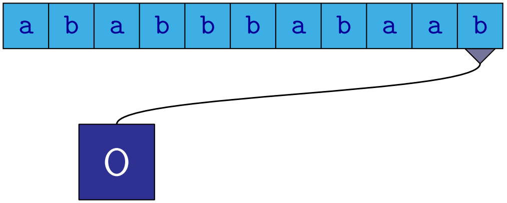




Recognition of a Language

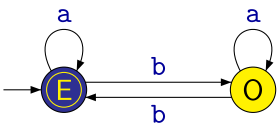
The behaviour of the device can be described by the following graph:

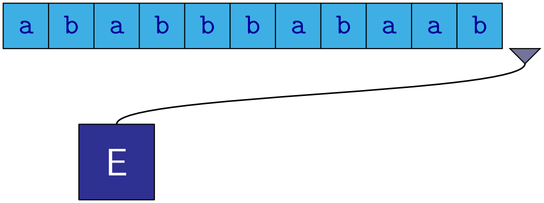

$$\begin{tikzcd}
\arrow[r] & E \arrow[loop above, "a"] \arrow[r, bend left=10, "b"] & O \arrow[loop above, "a"] \arrow[l, bend left=10, "b"]
\end{tikzcd}$$



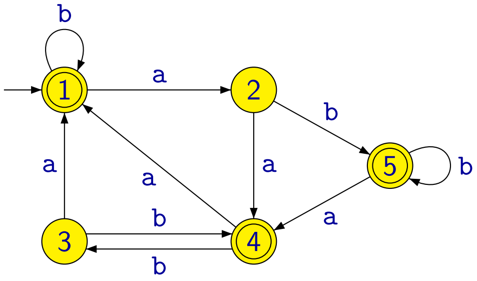
Recognition of a Language

The behaviour of the device can be described by the following graph:


$$\begin{tikzcd}
\arrow[r] & E \arrow[loop above, "a"] \arrow[rr, bend left, "b"] & & O \arrow[loop above, "a"] \arrow[ll, bend left, "b"]
\end{tikzcd}$$



Deterministic Finite Automaton



A **deterministic finite automaton** consists of **states** and **transitions**. One of the states is denoted as an **initial state** and some of states are denoted as **accepting**.

Deterministic Finite Automaton

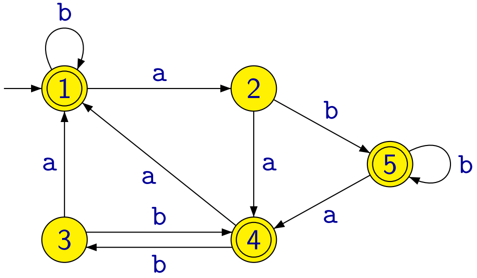
Formally, a **deterministic finite automaton** (**DFA**) is defined as a tuple

$$(Q, \Sigma, \delta, q_0, F)$$

where:

- Q is a nonempty finite set of **states**
- Σ is an **alphabet** (a nonempty finite set of symbols)
- $\delta : Q \times \Sigma \rightarrow Q$ is a **transition function**
- $q_0 \in Q$ is an **initial state**
- $F \subseteq Q$ is a set of **accepting states**

Deterministic Finite Automaton



- $Q = \{1, 2, 3, 4, 5\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $q_0 = 1$
- $F = \{1, 4, 5\}$

$\delta(1, \mathtt{a}) = 2$	$\delta(1, \mathtt{b}) = 1$
$\delta(2, \mathtt{a}) = 4$	$\delta(2, \mathtt{b}) = 5$
$\delta(3, \mathtt{a}) = 1$	$\delta(3, \mathtt{b}) = 4$
$\delta(4, \mathtt{a}) = 1$	$\delta(4, \mathtt{b}) = 3$
$\delta(5, \mathtt{a}) = 4$	$\delta(5, \mathtt{b}) = 5$

Deterministic Finite Automaton

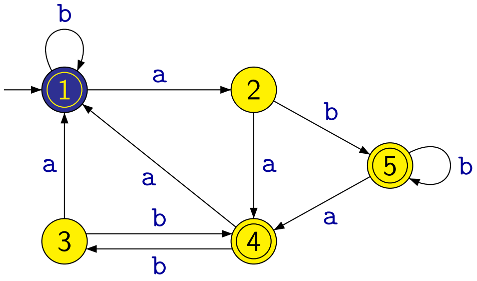
Instead of

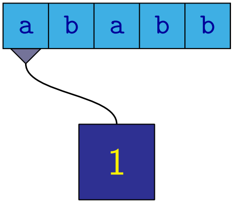
$$\begin{array}{ll}
\delta(1,\mathtt{a}) = 2 & \delta(1,\mathtt{b}) = 1 \\
\delta(2,\mathtt{a}) = 4 & \delta(2,\mathtt{b}) = 5 \\
\delta(3,\mathtt{a}) = 1 & \delta(3,\mathtt{b}) = 4 \\
\delta(4,\mathtt{a}) = 1 & \delta(4,\mathtt{b}) = 3 \\
\delta(5,\mathtt{a}) = 4 & \delta(5,\mathtt{b}) = 5
\end{array}$$

we rather use a more succinct representation as a table or a depicted graph:

δ	a	b
$\leftrightarrow$ 1	2	1
2	4	5
3	1	4
$\leftarrow$ 4	1	3
$\leftarrow$ 5	4	5

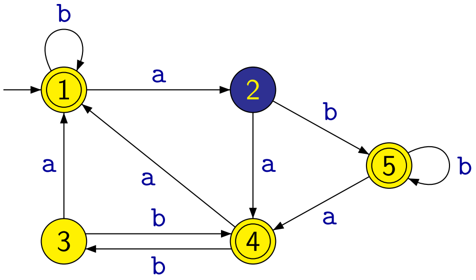
Deterministic Finite Automaton

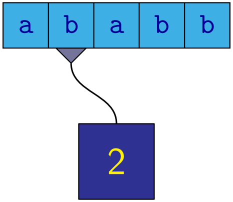




1

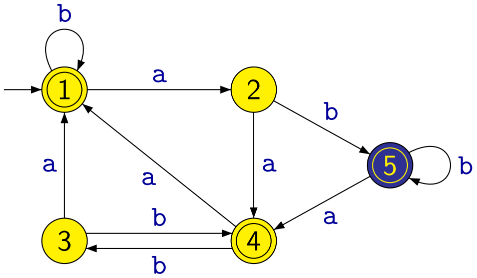
Deterministic Finite Automaton

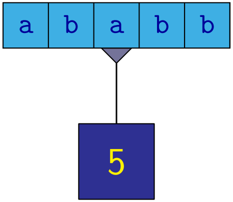




$1 \xrightarrow{a} 2$

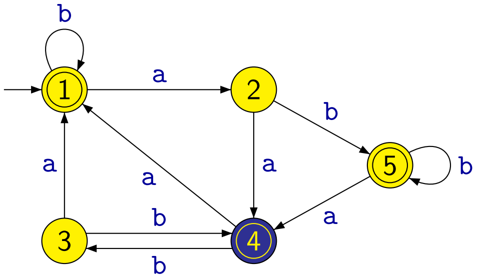
Deterministic Finite Automaton

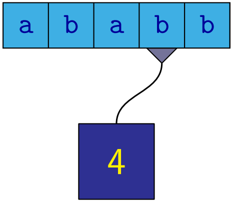




$1 \xrightarrow{a} 2 \xrightarrow{b} 5$

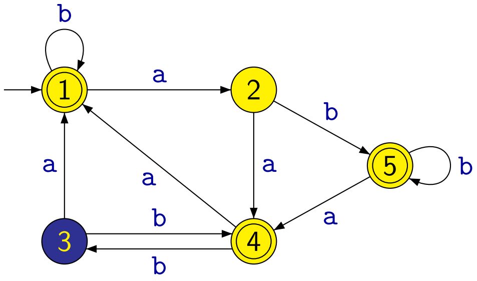
Deterministic Finite Automaton

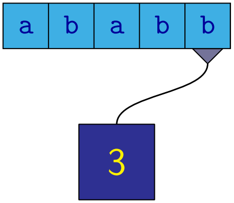




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4$$

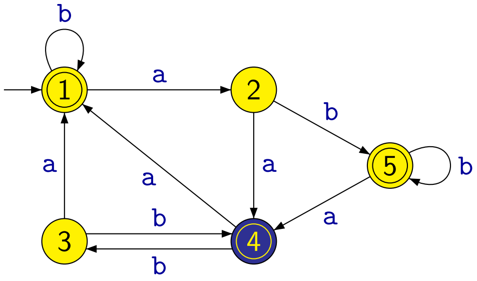
Deterministic Finite Automaton

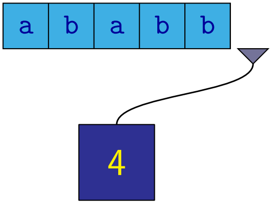




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3$$

Deterministic Finite Automaton





$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3 \xrightarrow{b} 4$$

Deterministic Finite Automaton

Definition

Let us have a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$.

By $q \xrightarrow{w} q'$, where $q, q' \in Q$ and $w \in \Sigma^*$, we denote the fact that the automaton, starting in state q goes to state q' by reading word w.

Remark: $\longrightarrow \subseteq Q \times \Sigma^* \times Q$ is a ternary relation.

Instead of $(q, w, q') \in \longrightarrow$ we write $q \xrightarrow{w} q'$.

It holds for a DFA that for each state q and each word w there is exactly one state q' such that $q \xrightarrow{w} q'$.

Deterministic Finite Automaton

Relation $\longrightarrow$ can be formally defined by the following inductive definition:

- $q \xrightarrow{\varepsilon} q$ for each $q \in Q$
- For $w \in \Sigma^*$ and $a \in \Sigma$:

  $q \xrightarrow{wa} q'$ iff there is $q'' \in Q$ such that

  $$q \xrightarrow{w} q'' \text{ and } \delta(q'', a) = q'$$

Deterministic Finite Automaton

$$\begin{tikzcd}[column sep=tiny]
1 \xrightarrow{\varepsilon} 1 \arrow[dr] & & \delta(1,\mathtt{a}) = 2 \arrow[dl] & & & & \\
& 1 \xrightarrow{\mathtt{a}} 2 \arrow[dr] & & \delta(2,\mathtt{b}) = 5 \arrow[dl] & & & \\
& & 1 \xrightarrow{\mathtt{ab}} 5 \arrow[dr] & & \delta(5,\mathtt{a}) = 4 \arrow[dl] & & \\
& & & 1 \xrightarrow{\mathtt{aba}} 4 \arrow[dr] & & \delta(4,\mathtt{b}) = 3 \arrow[dl] & \\
& & & & 1 \xrightarrow{\mathtt{abab}} 3 \arrow[dr] & & \delta(3,\mathtt{b}) = 4 \arrow[dl] \\
& & & & & 1 \xrightarrow{\mathtt{ababb}} 4 &
\end{tikzcd}$$

	a	b
$\leftrightarrow$ 1	2	1
2	4	5
3	1	4
$\leftarrow$ 4	1	3
$\leftarrow$ 5	4	5

Deterministic Finite Automaton

A word $w \in \Sigma^*$ is **accepted** by a deterministic finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ iff there exists a state $q \in F$ such that $q_0 \xrightarrow{w} q$.

Definition

A **language** accepted by a given deterministic finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, denoted $\mathcal{L}(\mathcal{A})$, is the set of all words accepted by the automaton, i.e.,

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists q \in F : q_0 \xrightarrow{w} q\}$$

Regular languages

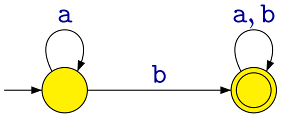
Definition

A language L is **regular** iff there exists some deterministic finite automaton accepting L, i.e., DFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = L$.

Examples of Deterministic Finite Automata

Example: An automaton recognizing the language L over alphabet $\{\mathtt{a}, \mathtt{b}\}$ consisting of those words that contain at least one occurrence of symbol $\mathtt{b}$, i.e.,

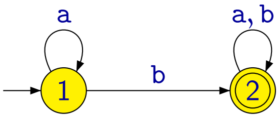
$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w|_{\mathtt{b}} \geq 1\}$$



Examples of Deterministic Finite Automata

Example: An automaton recognizing the language L over alphabet $\{\mathtt{a}, \mathtt{b}\}$ consisting of those words that contain at least one occurrence of symbol $\mathtt{b}$, i.e.,

$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w|_{\mathtt{b}} \geq 1\}$$

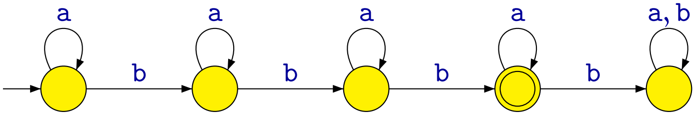


	a	b
$\rightarrow 1$	1	2
$\leftarrow 2$	2	2

Examples of Deterministic Finite Automata

Example: An automaton recognizing the language L over alphabet $\{\mathtt{a},\mathtt{b}\}$ consisting of those words that contain exactly three occurrences of symbol $\mathtt{b}$, i.e.,

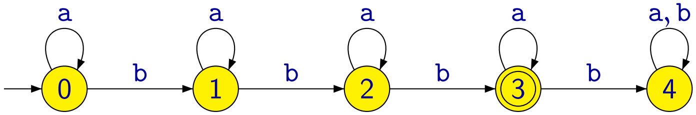
$$L = \{w \in \{\mathtt{a},\mathtt{b}\}^* \mid |w|_{\mathtt{b}} = 3\}$$



Examples of Deterministic Finite Automata

Example: An automaton recognizing the language L over alphabet $\{\mathtt{a}, \mathtt{b}\}$ consisting of those words that contain exactly three occurrences of symbol $\mathtt{b}$, i.e.,

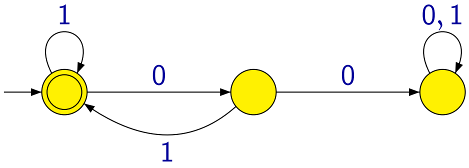
$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w|_{\mathtt{b}} = 3\}$$



	a	b
$\rightarrow 0$	0	1
1	1	2
2	2	3
$\leftarrow 3$	3	4
4	4	4

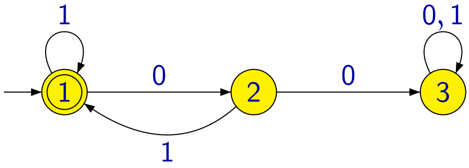
Examples of Deterministic Finite Automata

Example: An automaton recognizing the language over alphabet $\{0, 1\}$ consisting of those words where every occurrence of symbol 0 is immediately followed with symbol 1.



Examples of Deterministic Finite Automata

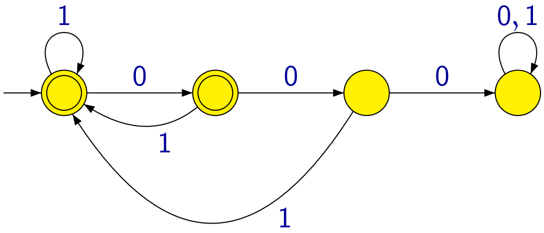
Example: An automaton recognizing the language over alphabet $\{0, 1\}$ consisting of those words where every occurrence of symbol 0 is immediately followed with symbol 1.



	0	1
$\leftrightarrow 1$	2	1
2	3	1
3	3	3

Examples of Deterministic Finite Automata

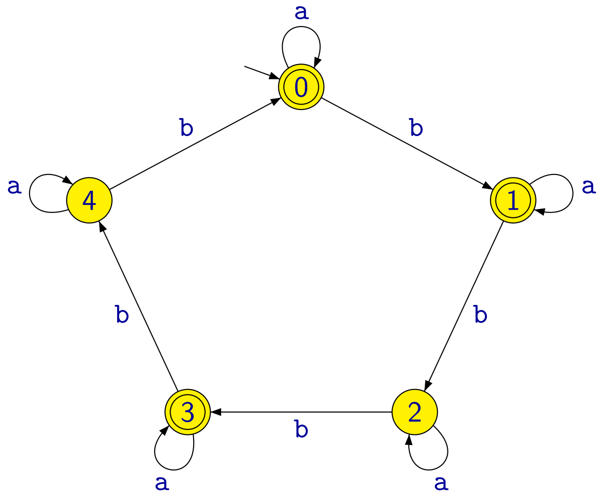
Example: An automaton recognizing the language over alphabet $\{0, 1\}$ consisting of those words where every pair of consecutive symbols 0 is immediately followed with symbol 1.



Examples of Deterministic Finite Automata

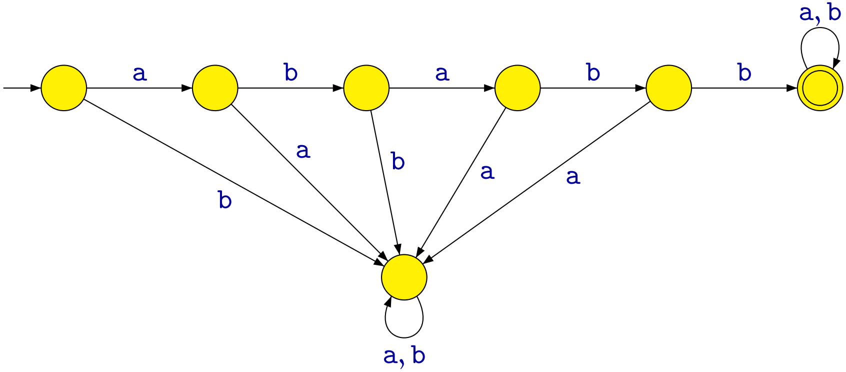
Example: An automaton recognizing the language

$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid (|w|_{\mathtt{b}} \bmod 5) \in \{0, 1, 3\}\}$$



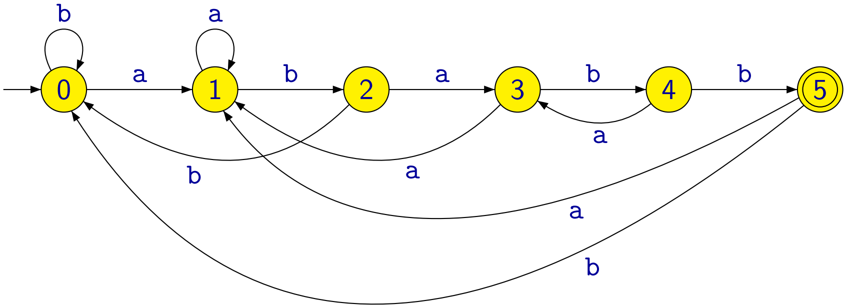
Examples of Deterministic Finite Automata

Example: An automaton recognizing the language over alphabet $\{a, b\}$ consisting of those words that start with the **prefix** ababb.



Examples of Deterministic Finite Automata

Example: An automaton recognizing the language over alphabet $\{\mathtt{a}, \mathtt{b}\}$ of those words that end with **suffix** `ababb`.



Examples of Deterministic Finite Automata

The construction of this automaton is based on the following idea:

- Let us assume that we want to search for a word u of length n (i.e., $|u| = n$).
  The states of the automaton are denoted with numbers $0, 1, \ldots, n$.
- A state with number i corresponds to the situation when i is the length of the longest word that is at the same time:
  - a prefix of the pattern u we are searching for
  - a suffix of the part of the input word that the automaton has read so far

For example, for the searched pattern `ababb` the states of the automaton correspond to the following words:

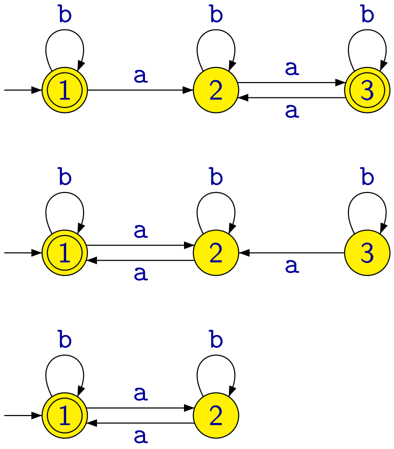
State 0	...	ε
State 1	...	`a`
State 2	...	`ab`
State 3	...	`aba`
State 4	...	`abab`
State 5	...	`ababb`

Examples of Deterministic Finite Automata

Example: An automaton recognizing the language over alphabet $\{\mathtt{a}, \mathtt{b}\}$ consisting of those words that contain **subword** `ababb`.

$$\begin{tikzcd}
{} \arrow[r] & 0 \arrow[loop above, "\mathtt{b}"] \arrow[r, "\mathtt{a}"] & 1 \arrow[loop above, "\mathtt{a}"] \arrow[r, "\mathtt{b}"] & 2 \arrow[r, "\mathtt{a}"] \arrow[ll, bend left, "\mathtt{b}"] & 3 \arrow[r, "\mathtt{b}"] \arrow[ll, bend left, "\mathtt{a}"] & 4 \arrow[r, "\mathtt{b}"] \arrow[l, bend left, "\mathtt{a}"] & 5 \arrow[loop above, "{\mathtt{a}, \mathtt{b}}"]
\end{tikzcd}$$

Equivalence of Automata



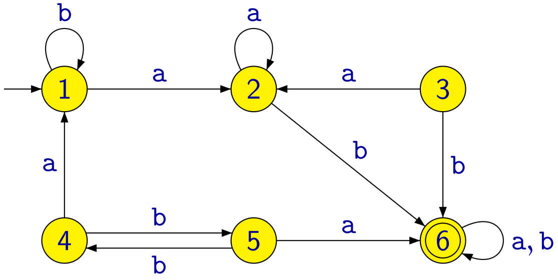
All three automata accept the language of all words with an even number of a's.

Equivalence of Automata

Definition

We say automata $\mathcal{A}_1$, $\mathcal{A}_2$ are **equivalent** if $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.

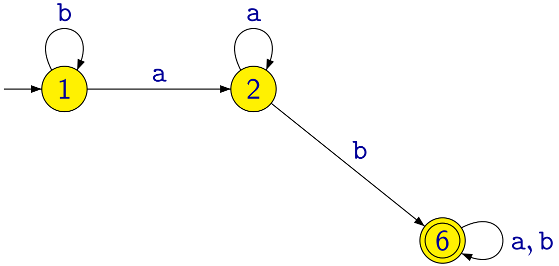
Unreachable States of an Automaton



- The automaton accepts the language
  $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ contains subword } \mathtt{ab}\}$
- There is no input sequence such that after reading it, the automaton gets to states 3, 4, or 5.

.

Unreachable States of an Automaton


$$\begin{tikzcd}
\arrow[r] & 1 \arrow[loop above, "b"] \arrow[r, "a"] & 2 \arrow[loop above, "a"] \arrow[rd, "b"] & \\
& & & 6 \arrow[loop right, "{a, b}"]
\end{tikzcd}$$

- The automaton accepts the language
  $L = \{w \in \{a, b\}^* \mid w \text{ contains subword } ab\}$
- There is no input sequence such that after reading it, the automaton gets to states 3, 4, or 5.
- If we remove these states, the automaton still accepts the same language L.

Unreachable States of an Automaton

Definition

A state q of a finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is **reacheable** if there exists a word w such that $q_0 \xrightarrow{w} q$.

Otherwise the state is **unreachable**.

- There is no path in a graph of an automaton going from the initial state to some unreachable state.
- Unreachable states can be removed from an automaton (together with all transitions going to them and from them). The language accepted by the automaton is not affected.

Automaton and Operations on Languages

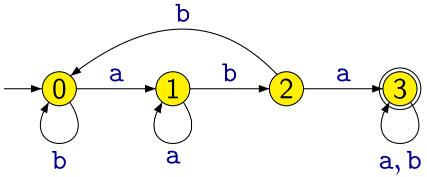
When we construct automata, it can be difficult to construct an automaton for a given language L directly.

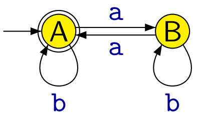
If it is possible to describe the language L as a result of some language operations (intersection, union, concatenation, iteration, ...) applied to some simpler languages L_1 and L_2, then it can be easier to proceed in a modular manner:

- To construct automata for languages L_1 and L_2.
- Then to use some of general constructions that allow to algorithmically construct an automaton for language L, which is a result of applying a given language operation on languages L_1 and L_2, from automata for languages L_1 and L_2.

An Automaton for Intersection of Languages

Let us have the following two automata:

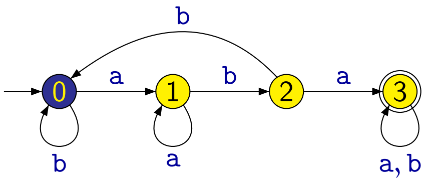


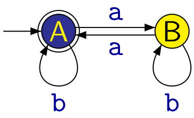


Do both of them accept the word abbaaba?

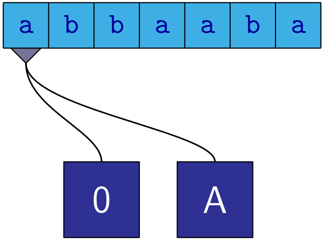
An Automaton for Intersection of Languages

Let us have the following two automata:



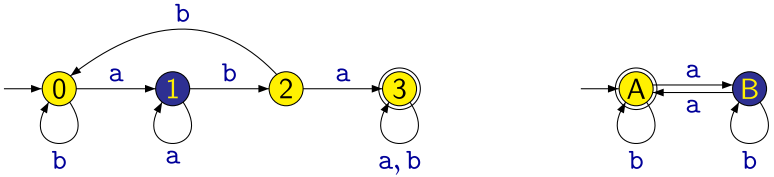


Do both of them accept the word abbaaba?

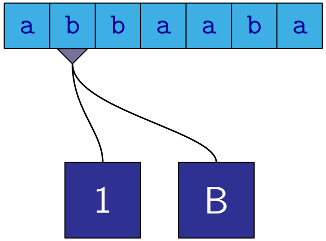


An Automaton for Intersection of Languages

Let us have the following two automata:

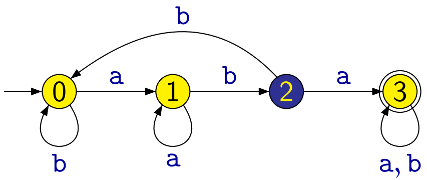


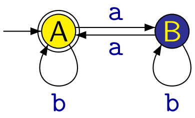
Do both of them accept the word abbaaba?



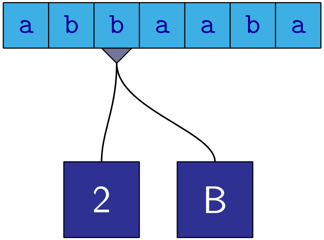
An Automaton for Intersection of Languages

Let us have the following two automata:



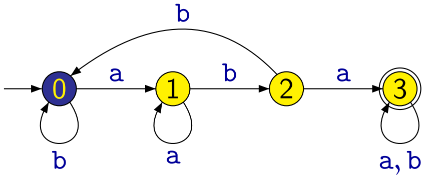


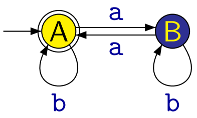
Do both of them accept the word abbaaba?



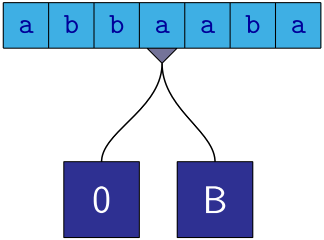
An Automaton for Intersection of Languages

Let us have the following two automata:



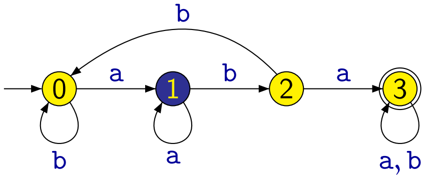


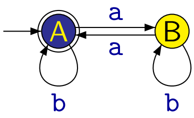
Do both of them accept the word abbaaba?



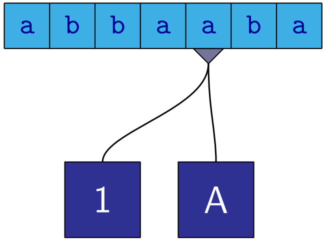
An Automaton for Intersection of Languages

Let us have the following two automata:



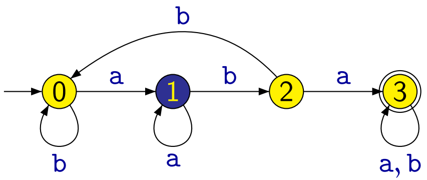


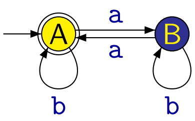
Do both of them accept the word abbaaba?



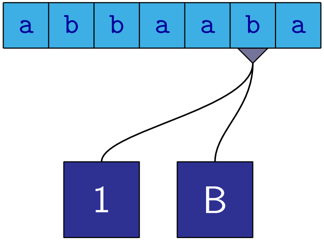
An Automaton for Intersection of Languages

Let us have the following two automata:


$$\begin{tikzcd}
\arrow[r] & 0 \arrow[r, "a"] \arrow[loop below, "b"] & 1 \arrow[r, "b"] \arrow[loop below, "a"] & 2 \arrow[r, "a"] \arrow[ll, bend right, "b"'] & 3 \arrow[loop below, "{a, b}"]
\end{tikzcd}$$

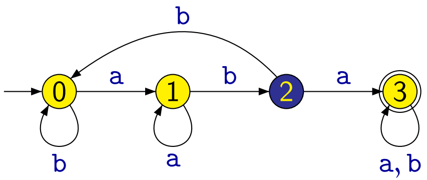

$$\begin{tikzcd}
\arrow[r] & A \arrow[r, shift left, "a"] \arrow[loop below, "b"] & B \arrow[l, shift left, "a"] \arrow[loop below, "b"]
\end{tikzcd}$$

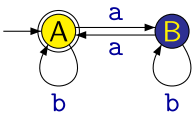
Do both of them accept the word abbaaba?



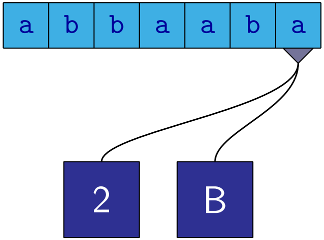
An Automaton for Intersection of Languages

Let us have the following two automata:



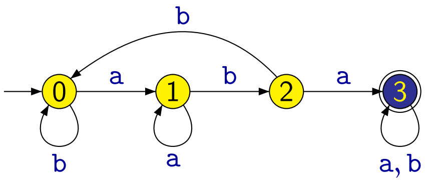


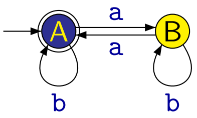
Do both of them accept the word abbaaba?



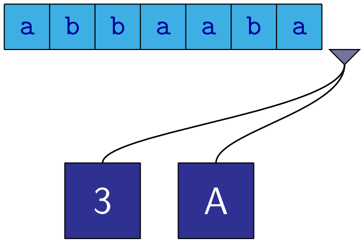
An Automaton for Intersection of Languages

Let us have the following two automata:

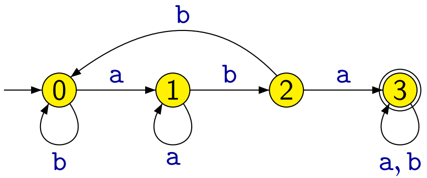


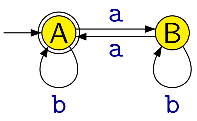


Do both of them accept the word abbaaba?

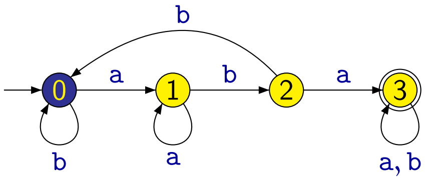


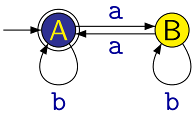
An Automaton for Intersection of Languages

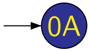




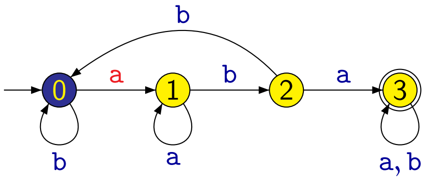
An Automaton for Intersection of Languages

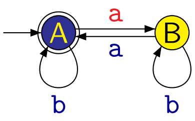


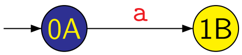




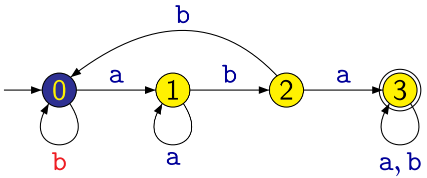
An Automaton for Intersection of Languages

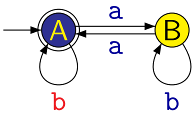


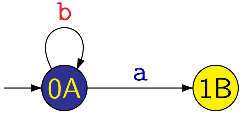




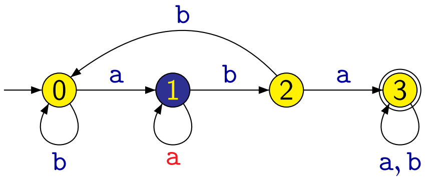
An Automaton for Intersection of Languages

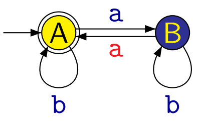


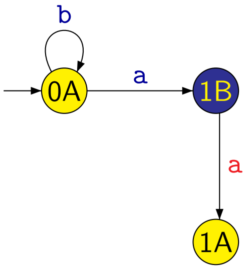




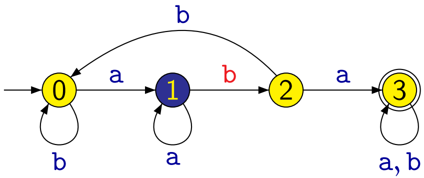
An Automaton for Intersection of Languages

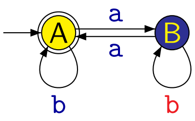


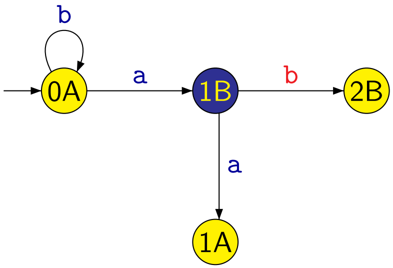




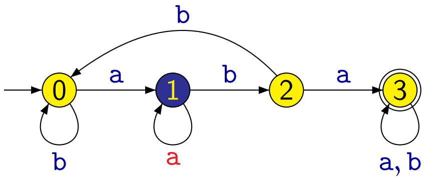
An Automaton for Intersection of Languages

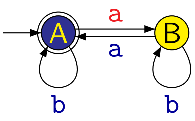


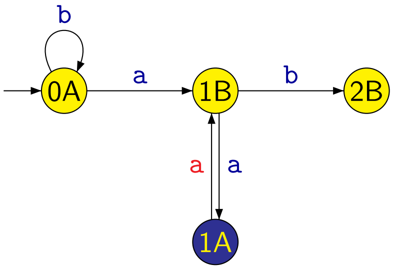




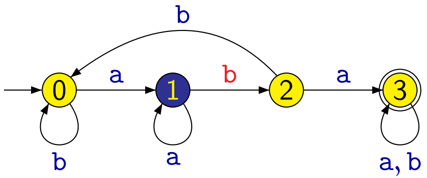
An Automaton for Intersection of Languages

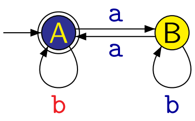


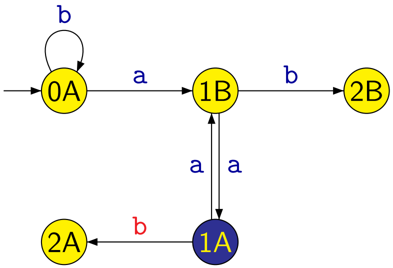




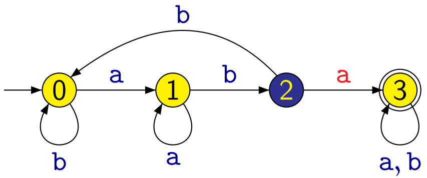
An Automaton for Intersection of Languages

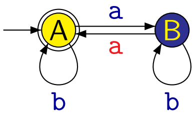


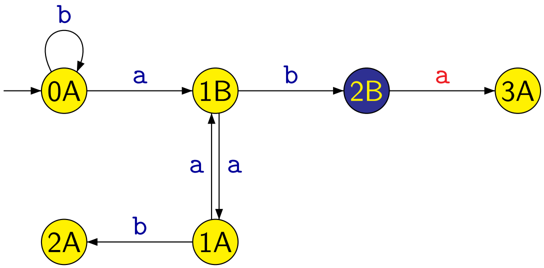




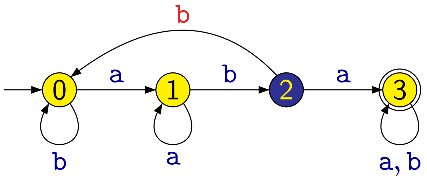
An Automaton for Intersection of Languages

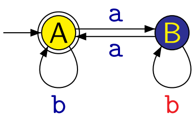


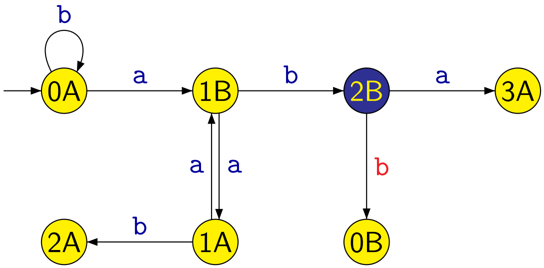




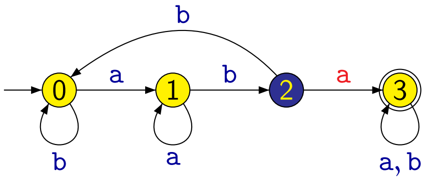
An Automaton for Intersection of Languages

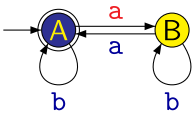


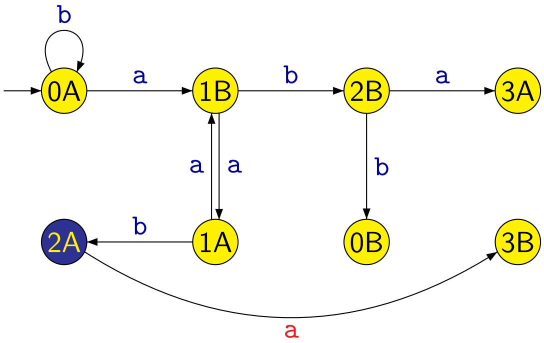




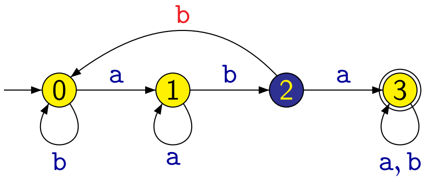
An Automaton for Intersection of Languages

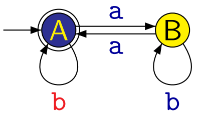


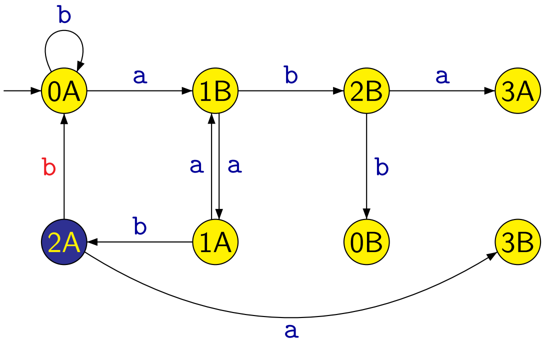




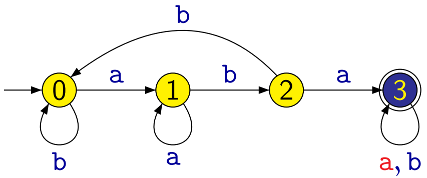
An Automaton for Intersection of Languages

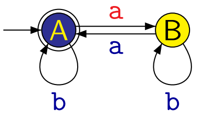


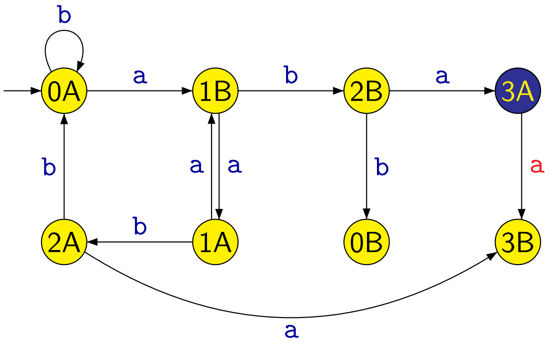




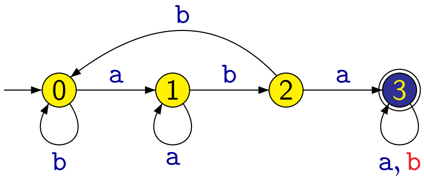
An Automaton for Intersection of Languages

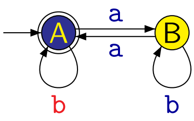


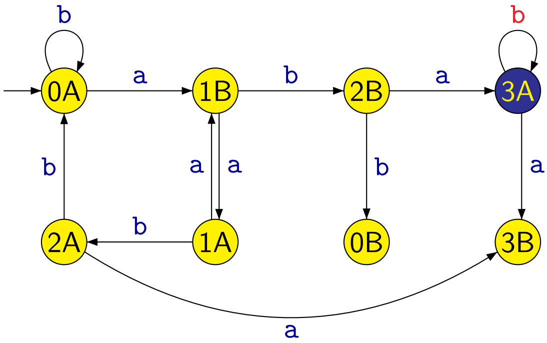




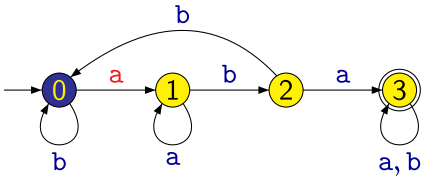
An Automaton for Intersection of Languages

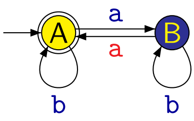


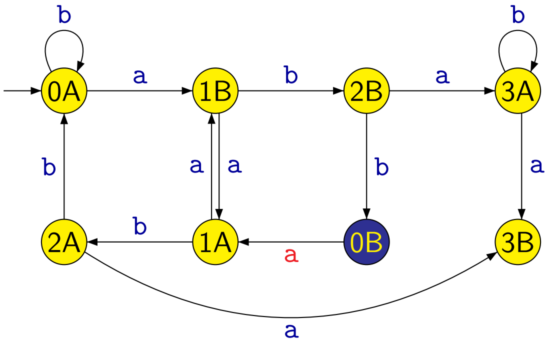




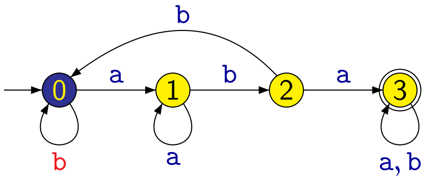
An Automaton for Intersection of Languages

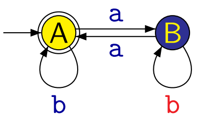


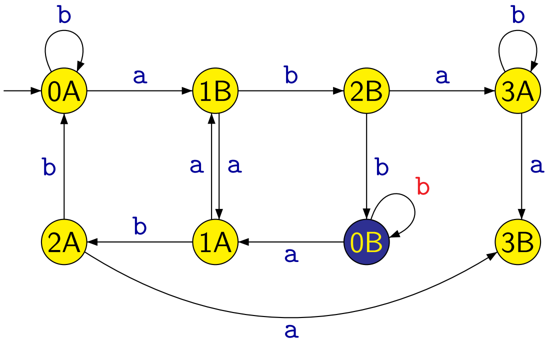




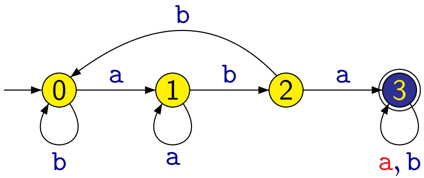
An Automaton for Intersection of Languages

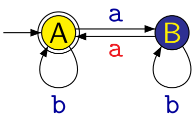


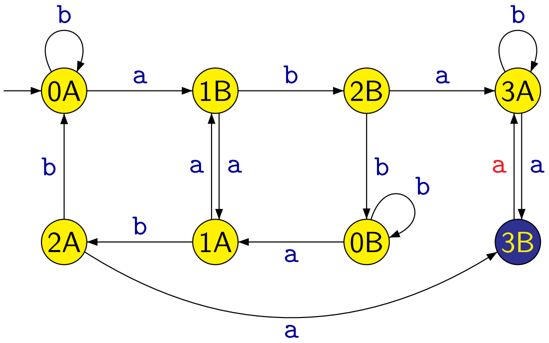




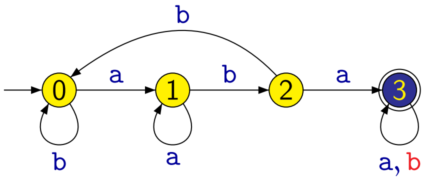
An Automaton for Intersection of Languages

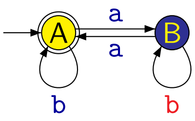


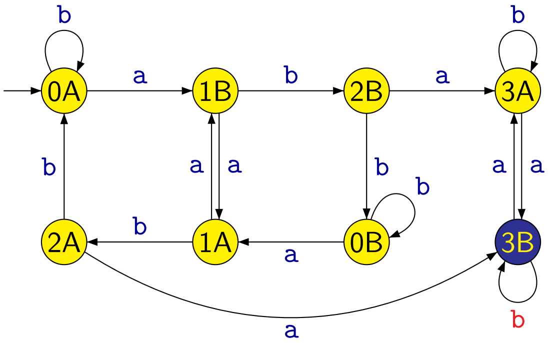




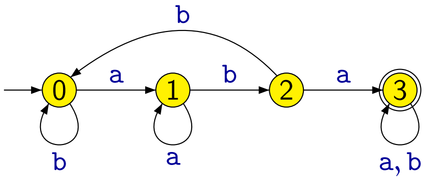
An Automaton for Intersection of Languages

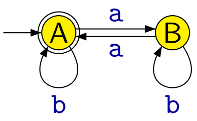


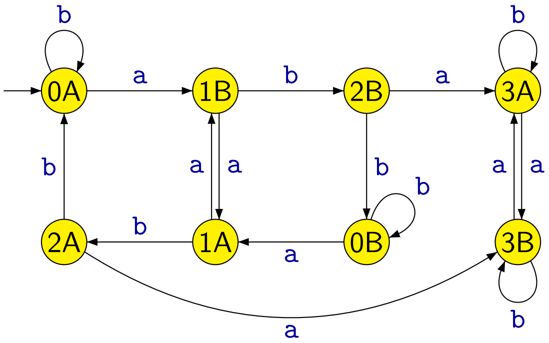




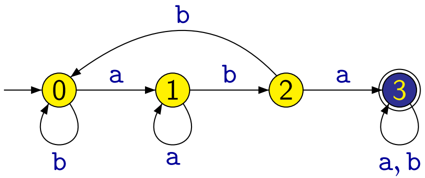
An Automaton for Intersection of Languages

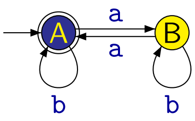


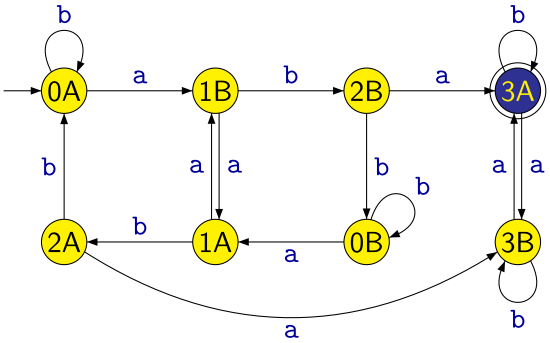




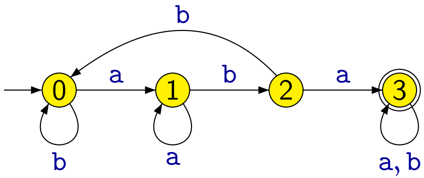
An Automaton for Intersection of Languages

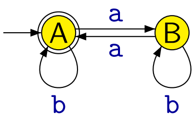


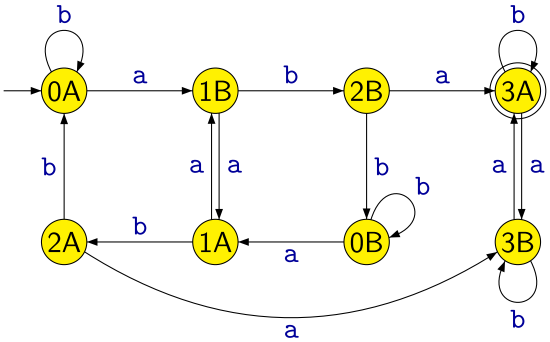




An Automaton for Intersection of Languages







An Automaton for Intersection of Languages

Formally, the construction can be described as follows:

We assume we have two deterministic finite automata $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$.

We construct DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Q_1 \times Q_2$
- $\delta(\,(q_1, q_2), a\,) = (\,\delta_1(q_1, a), \delta_2(q_2, a)\,)$ for each $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \Sigma$
- $q_0 = (q_{01}, q_{02})$
- $F = F_1 \times F_2$

It is not difficult to check that for each word $w \in \Sigma^*$ we have $w \in \mathcal{L}(\mathcal{A})$ iff $w \in \mathcal{L}(\mathcal{A}_1)$ and $w \in \mathcal{L}(\mathcal{A}_2)$, i.e.,

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$$

Intersection of Regular Languages

Theorem

If languages $L_1, L_2 \subseteq \Sigma^*$ are regular then also the language $L_1 \cap L_2$ is regular.

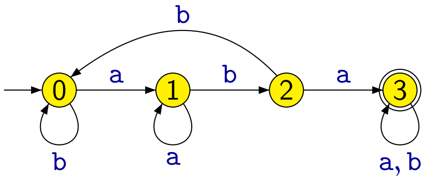
Proof: Let us assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic finite automata such that

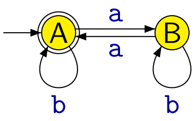
$$L_1 = \mathcal{L}(\mathcal{A}_1) \qquad\qquad L_2 = \mathcal{L}(\mathcal{A}_2)$$

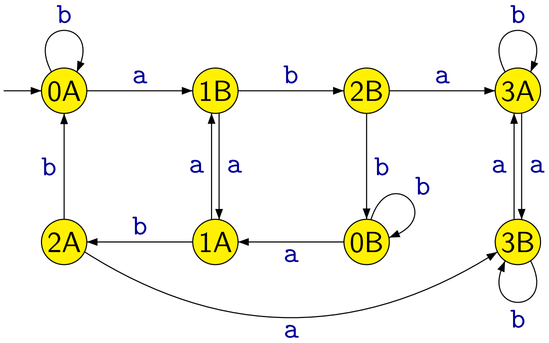
Using the described construction, we can construct a deterministic finite automaton $\mathcal{A}$ such that

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = L_1 \cap L_2$$

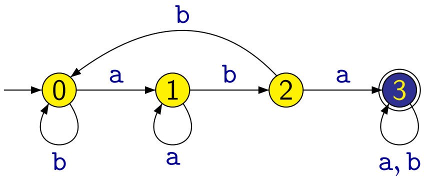
An Automaton for the Union of Languages

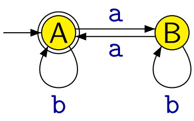


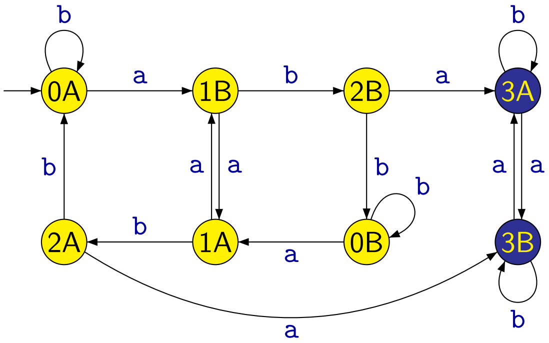




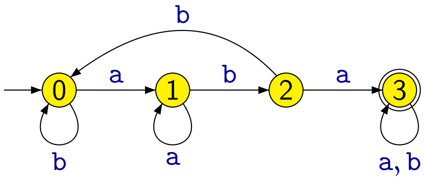
An Automaton for the Union of Languages

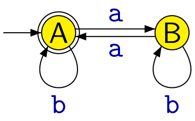


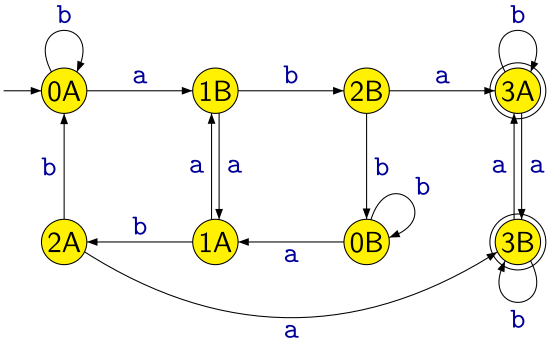




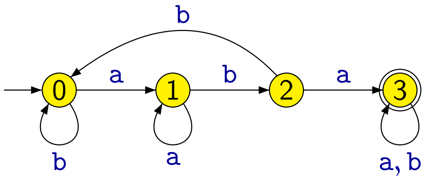
An Automaton for the Union of Languages

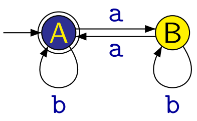


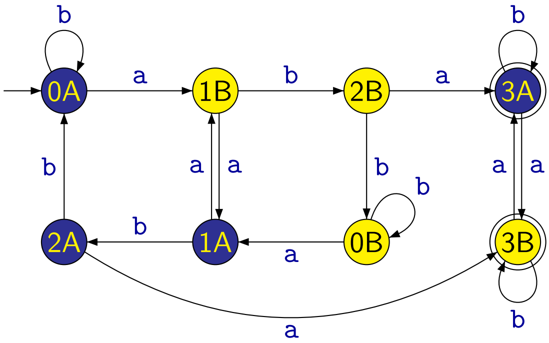




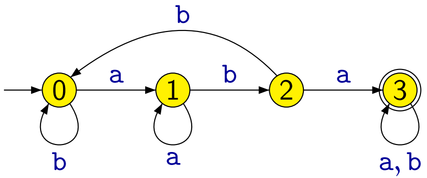
An Automaton for the Union of Languages

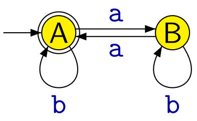


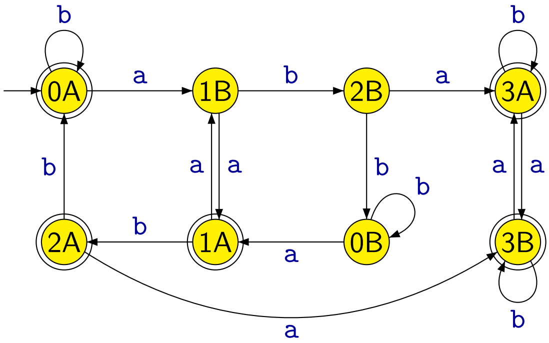




An Automaton for the Union of Languages







Union of Regular Languages

The construction of an automaton $\mathcal{A}$ that accepts the **union** of languages accepted by automata $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., the language

$$\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_1)$$

is almost identical as in the case of the automaton accepting $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

The only difference is the set of accepting states:

- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Union of Regular Languages

The construction of an automaton $\mathcal{A}$ that accepts the **union** of languages accepted by automata $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., the language

$$\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_1)$$

is almost identical as in the case of the automaton accepting $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

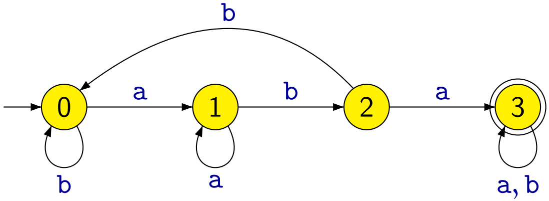
The only difference is the set of accepting states:

- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

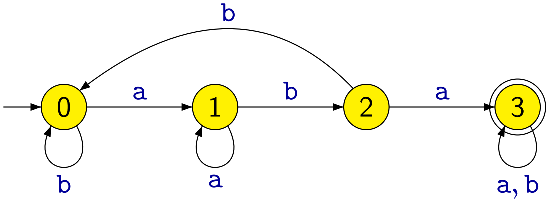
Theorem

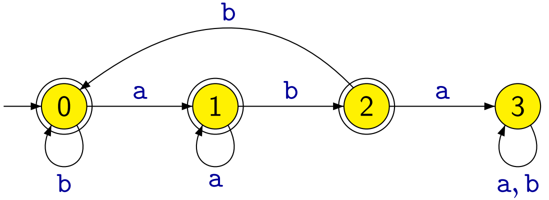
If languages $L_1, L_2 \subseteq \Sigma^*$ are regular then also the language $L_1 \cup L_2$ is regular.

An Automaton for the Complement of a Language


$$\begin{tikzcd}
\arrow[r] & 0 \arrow[r, "a"] \arrow[loop below, "b"] & 1 \arrow[r, "b"] \arrow[loop below, "a"] & 2 \arrow[r, "a"] \arrow[ll, bend right, "b"'] & 3 \arrow[loop below, "{a, b}"]
\end{tikzcd}$$

An Automaton for the Complement of a Language





Complement of a Regular Language

Given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ we construct DFA $\mathcal{A}' = (Q, \Sigma, \delta, q_0, Q - F)$.

It is obvious that for each word $w \in \Sigma^*$ we have $w \in \mathcal{L}(\mathcal{A}')$ iff $w \notin \mathcal{L}(\mathcal{A})$, i.e.,

$$\mathcal{L}(\mathcal{A}') = \overline{\mathcal{L}(\mathcal{A})}$$

Complement of a Regular Language

Given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ we construct DFA $\mathcal{A}' = (Q, \Sigma, \delta, q_0, Q - F)$.

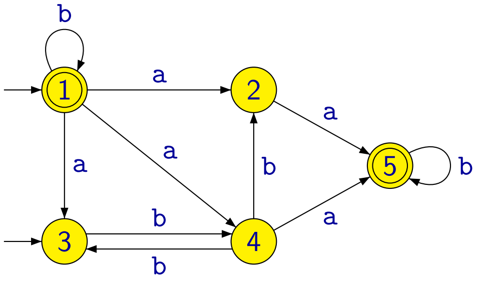
It is obvious that for each word $w \in \Sigma^*$ we have $w \in \mathcal{L}(\mathcal{A}')$ iff $w \notin \mathcal{L}(\mathcal{A})$, i.e.,

$$\mathcal{L}(\mathcal{A}') = \overline{\mathcal{L}(\mathcal{A})}$$

Theorem

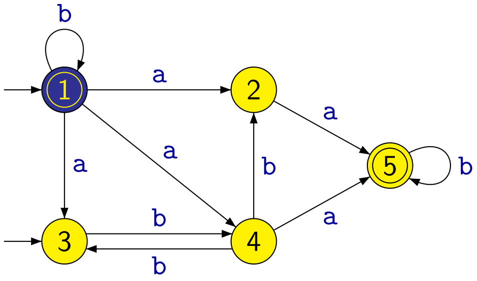
If a language L is regular then also its complement $\overline{L}$ is regular.

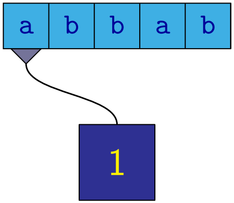
Nondeterministic Finite Automaton



- The number of transitions going from one state and labelled with the same symbol can be arbitrary (including zero).
- There can be more than one initial state in the automaton.

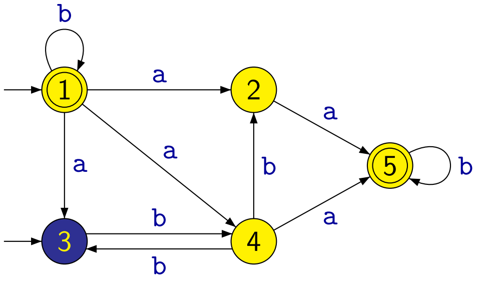
Nondeterministic Finite Automaton

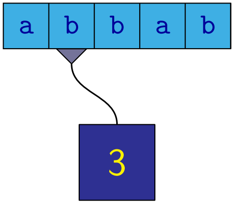




1

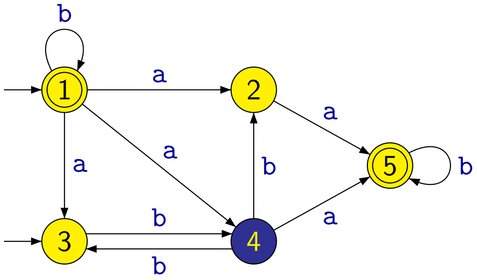
Nondeterministic Finite Automaton

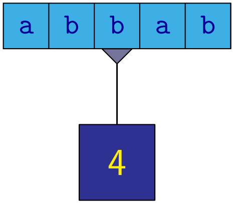




$1 \xrightarrow{a} 3$

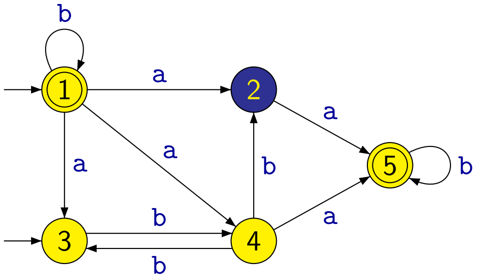
Nondeterministic Finite Automaton

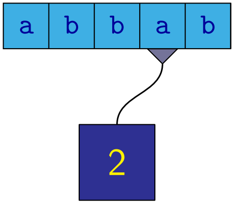




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

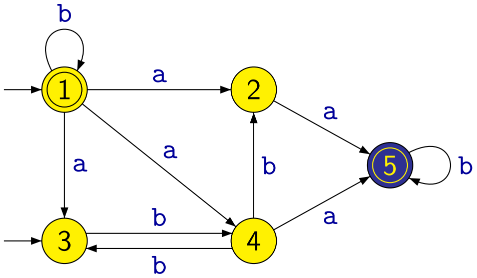
Nondeterministic Finite Automaton

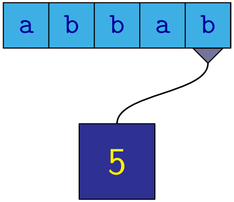




$$1 \xrightarrow{\texttt{a}} 3 \xrightarrow{\texttt{b}} 4 \xrightarrow{\texttt{b}} 2$$

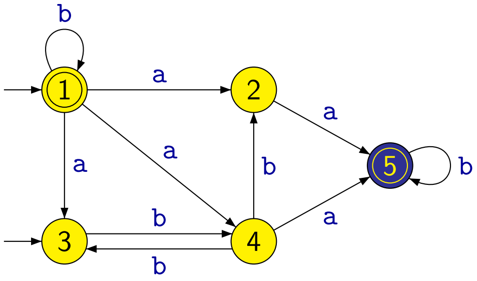
Nondeterministic Finite Automaton

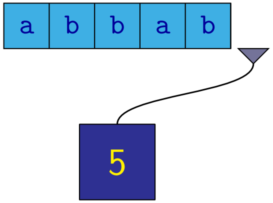




$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{b} 2 \xrightarrow{a} 5$

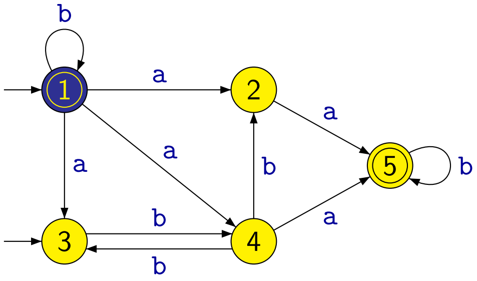
Nondeterministic Finite Automaton

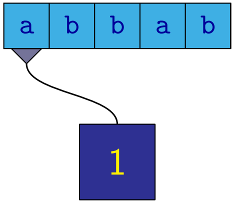




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{b} 2 \xrightarrow{a} 5 \xrightarrow{b} 5$$

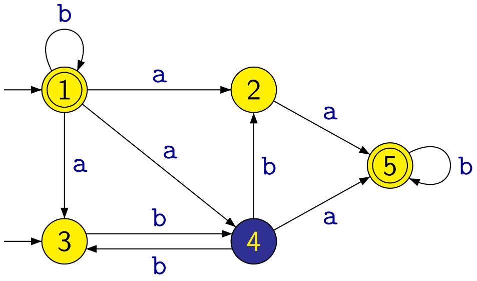
Nondeterministic Finite Automaton

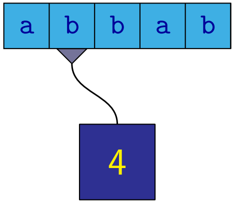




1

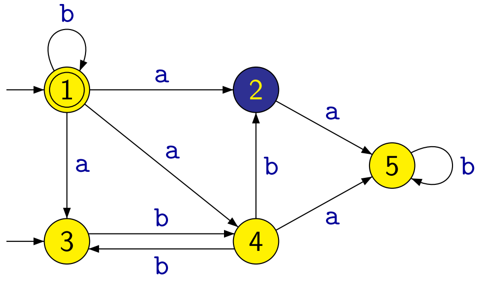
Nondeterministic Finite Automaton

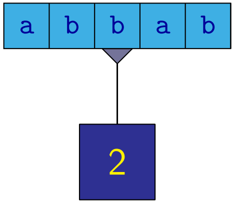




$1 \xrightarrow{a} 4$

Nondeterministic Finite Automaton

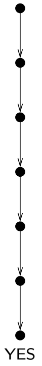


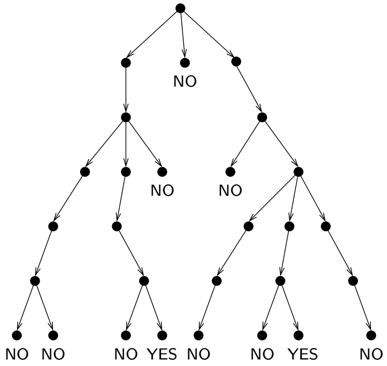


$1 \xrightarrow{a} 4 \xrightarrow{b} 2$

Nondeterministic Finite Automaton

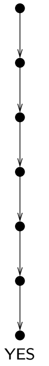
A nondeterministic finite automaton accepts a given word if there **exists** at least one computation of the automaton that accepts the word.

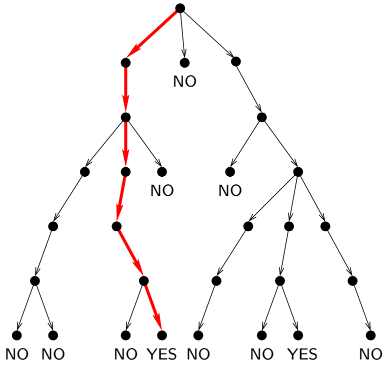




Nondeterministic Finite Automaton

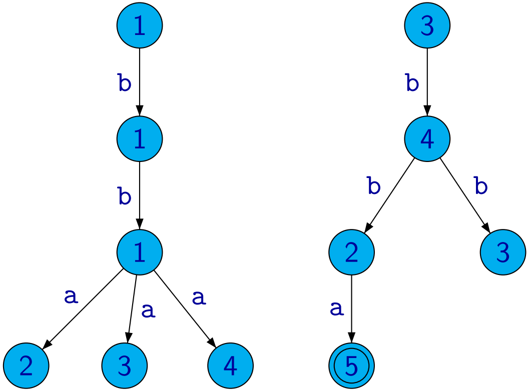
A nondeterministic finite automaton accepts a given word if there **exists** at least one computation of the automaton that accepts the word.





Nondeterministic Finite Automaton

	a	b
$\leftrightarrow 1$	$2, 3, 4$	1
2	5	—
$\rightarrow 3$	—	4
4	5	$2, 3$
$\leftarrow 5$	—	5



Example: A forest representing all possible computations over the word `bba`.

Nondeterministic Finite Automaton

Formally, a **nondeterministic finite automaton** (**NFA**) is defined as a tuple

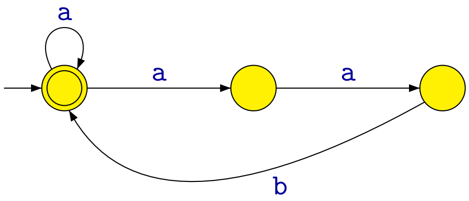
$$(Q, \Sigma, \delta, I, F)$$

where:

- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is a **transition fuction**
- $I \subseteq Q$ is a set of **initial states**
- $F \subseteq Q$ is a set of **accepting states**

Examples of Nondeterministic Finite Automata

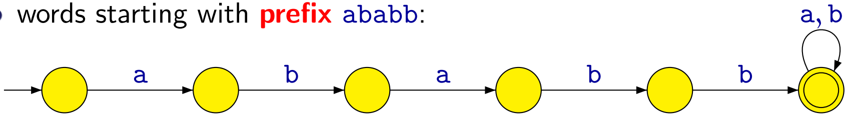
Example: An automaton recognizing the language over alphabet $\{\mathtt{a}, \mathtt{b}\}$ consisting of those words where every occurrence of symbol $\mathtt{b}$ is immediately preceded with two symbols $\mathtt{a}$.



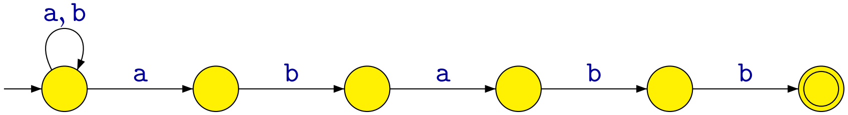
Examples of Nondeterministic Finite Automata

Example: An automaton recognizing the language over alphabet $\{a, b\}$:

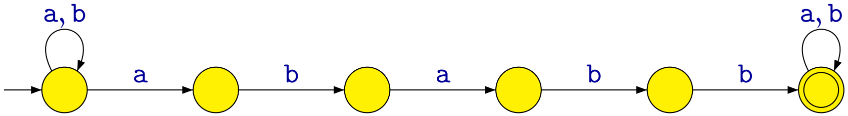
- words starting with **prefix** ababb:



- words ending with **suffix** ababb:

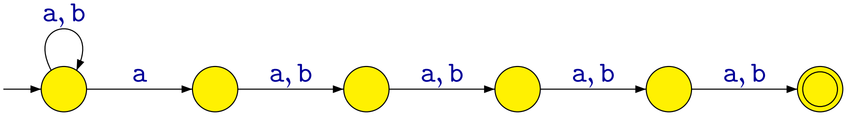


- words containing **subword** ababb:

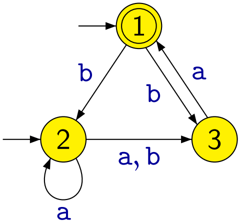


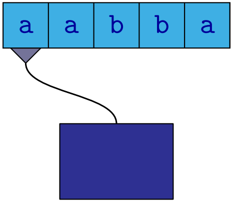
Examples of Nondeterministic Finite Automata

Example: An automaton recognizing the language over alphabet $\{a, b\}$ consisting of those words where the fifth symbol from the end is a.

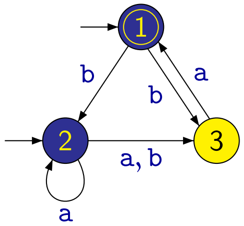


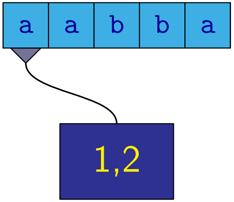
Transformation of NFA to DFA



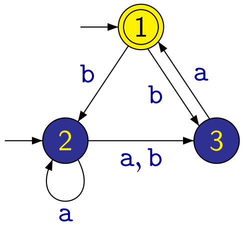


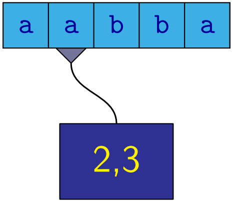
Transformation of NFA to DFA



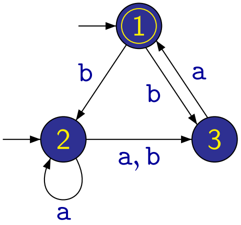


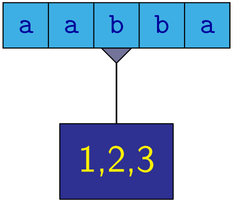
Transformation of NFA to DFA



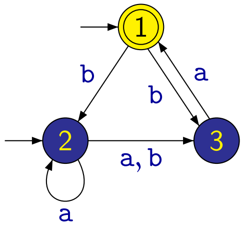


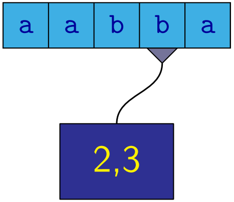
Transformation of NFA to DFA



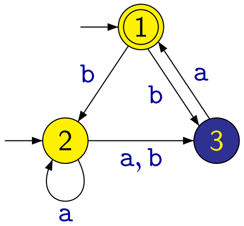


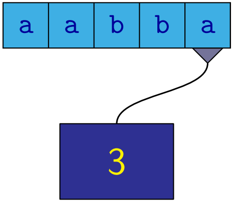
Transformation of NFA to DFA



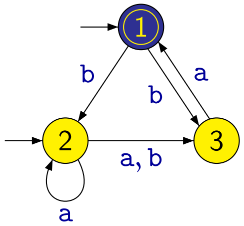


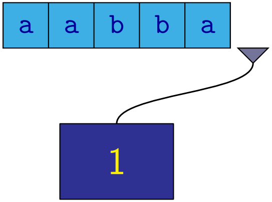
Transformation of NFA to DFA



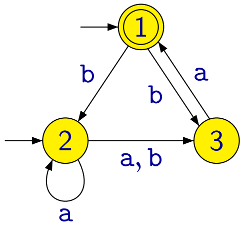


Transformation of NFA to DFA

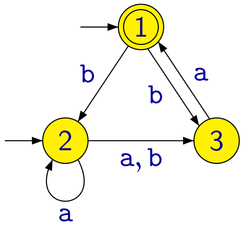


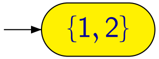


Transformation of NFA to DFA

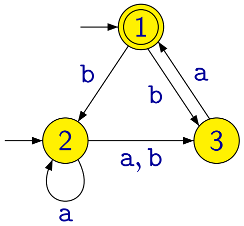


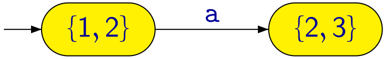
Transformation of NFA to DFA



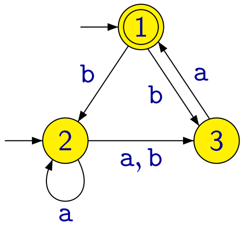


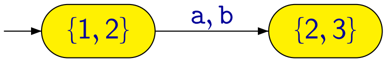
Transformation of NFA to DFA



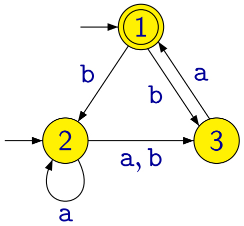


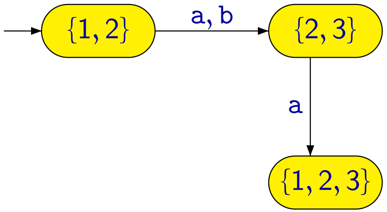
Transformation of NFA to DFA



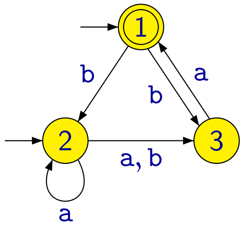


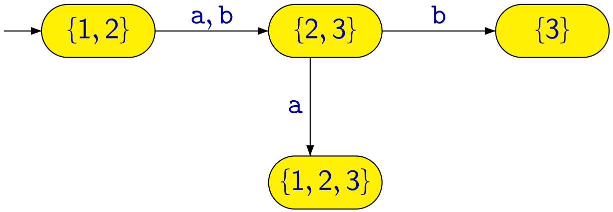
Transformation of NFA to DFA



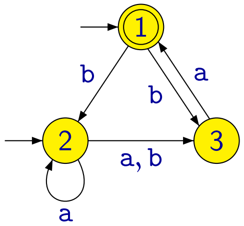


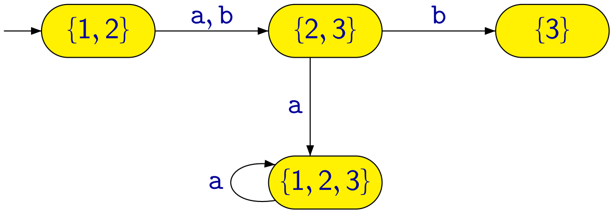
Transformation of NFA to DFA



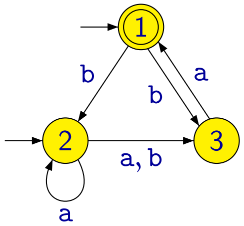


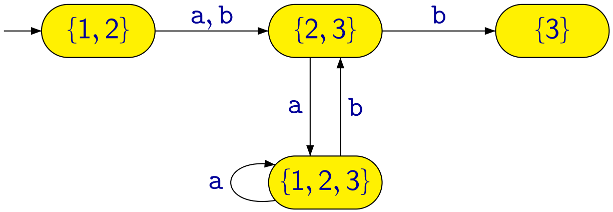
Transformation of NFA to DFA



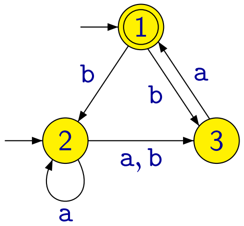


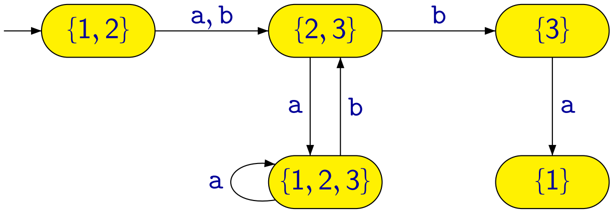
Transformation of NFA to DFA



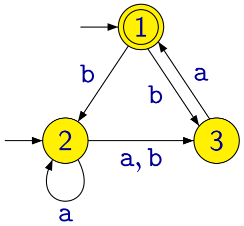


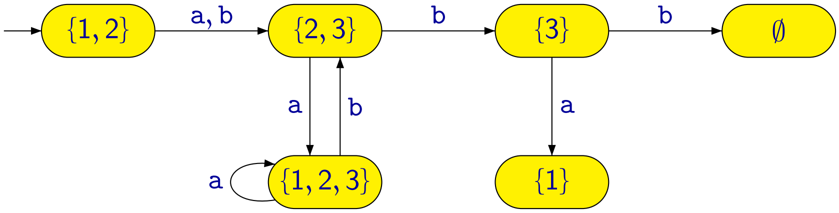
Transformation of NFA to DFA



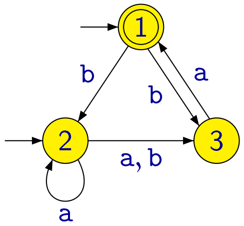


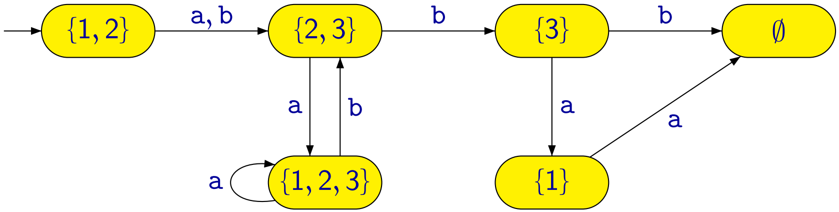
Transformation of NFA to DFA



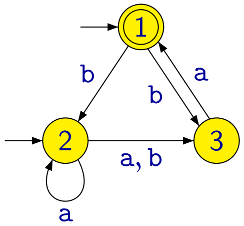


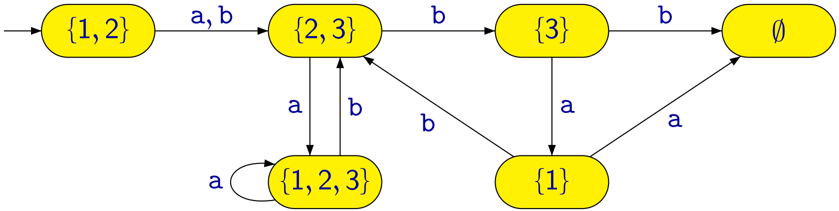
Transformation of NFA to DFA



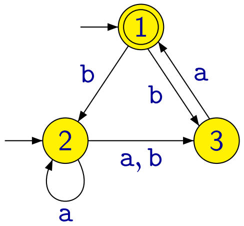


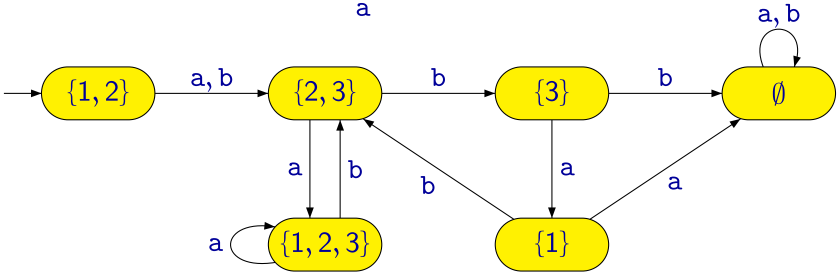
Transformation of NFA to DFA



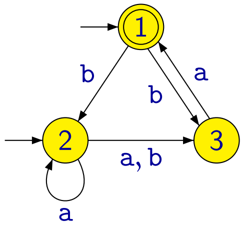


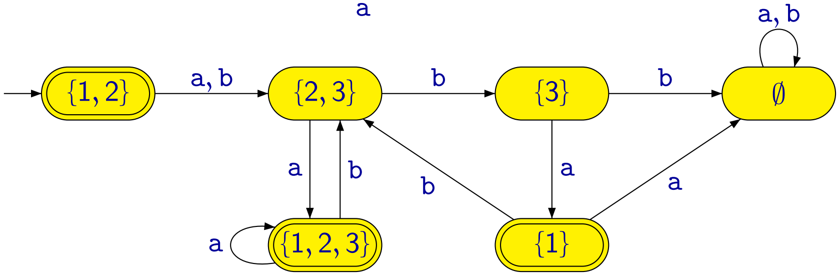
Transformation of NFA to DFA





Transformation of NFA to DFA





Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	2, 3
$\rightarrow 2$	2, 3	3
3	1	—

	a	b

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	$-$	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	$-$

	a	b
$\leftrightarrow \{1, 2\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	$-$	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	$-$

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	
$\{2,3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	2, 3
$\rightarrow 2$	2, 3	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	
$\leftarrow \{1,2,3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$		
$\{3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	2, 3
$\rightarrow 2$	2, 3	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	
$\{3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	$-$	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	$-$

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	
$\leftarrow \{1\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$		
$\emptyset$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	
$\emptyset$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

	a	b
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2, 3\}$
$\emptyset$		

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	–	2, 3
$\rightarrow 2$	2, 3	3
3	1	–

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$

Transformation of NFA to DFA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$

	a	b
$\leftrightarrow 1$	2	2
2	3	4
$\leftarrow 3$	3	2
4	5	6
$\leftarrow 5$	6	2
6	6	6

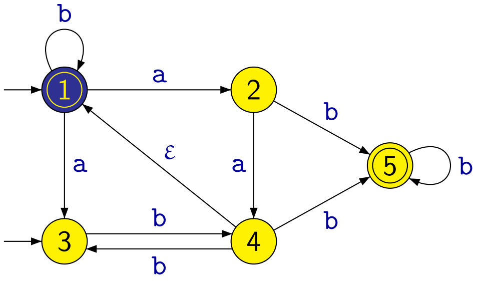
Transformation of NFA to DFA

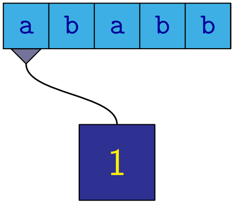
Remark: When a nondeterministic automaton with n states is transformed into a deterministic one, the resulting automaton can have 2^n states.

For example when we transform an automaton with 20 states, the resulting automaton can have $2^{20} = 1048576$ states.

It is often the case that the resulting automaton has far less than 2^n states. However, the worst cases are possible.

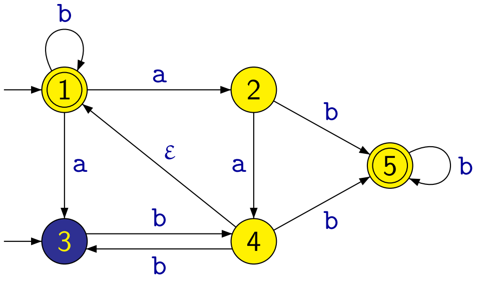
Generalized Nondeterministic Finite Automaton

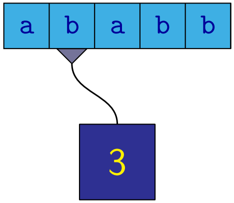




1

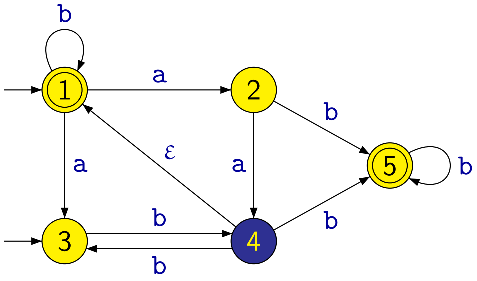
Generalized Nondeterministic Finite Automaton

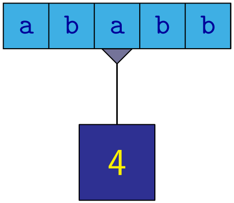




$1 \xrightarrow{a} 3$

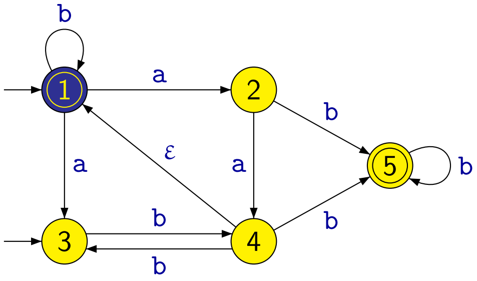
Generalized Nondeterministic Finite Automaton

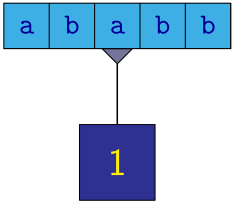




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

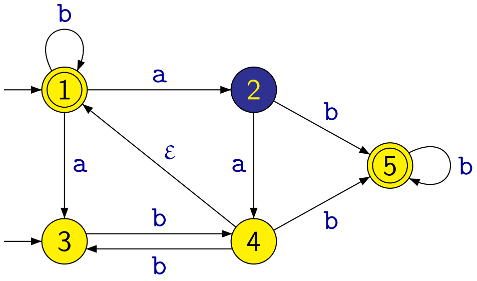
Generalized Nondeterministic Finite Automaton

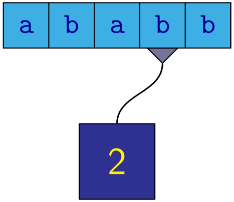




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1$$

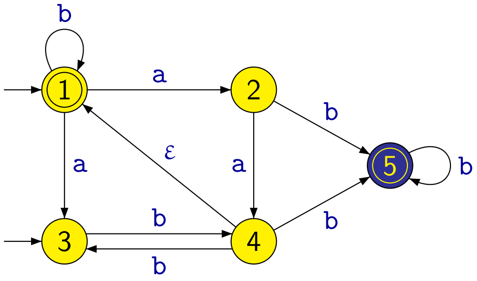
Generalized Nondeterministic Finite Automaton

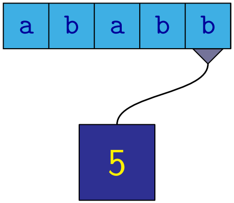




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2$$

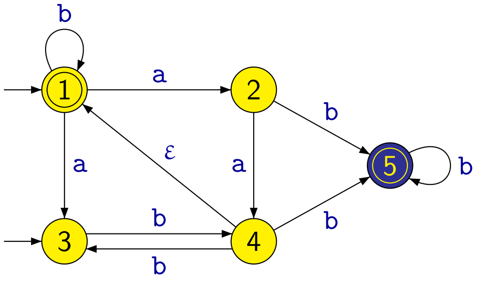
Generalized Nondeterministic Finite Automaton

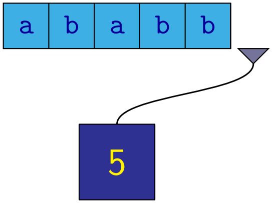




$$1 \xrightarrow{\texttt{a}} 3 \xrightarrow{\texttt{b}} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{\texttt{a}} 2 \xrightarrow{\texttt{b}} 5$$

Generalized Nondeterministic Finite Automaton





$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{b} 5$

Generalized Nondeterministic Finite Automaton

Compared to a nondeterministic finite automaton, a **generalized nondeterministic finite automaton** has the so called **ε-transitions**, i.e., transitions labelled with symbol ε.

When ε-transition is performed, only the state of the control unit is changed but the head on the tape is not moved.

Remark: The computations of a generalized nondeterministic automaton can be of an arbitrary length, even infinite (if the graph of the automaton contains a cycle consisting only of ε-transitions) regardless of the length of the word on the tape.

Generalized Nondeterministic Finite Automaton

Formally, a **generalized nondeterministic finite automaton** (**GNFA**) is defined as a tuple

$$(Q, \Sigma, \delta, I, F)$$

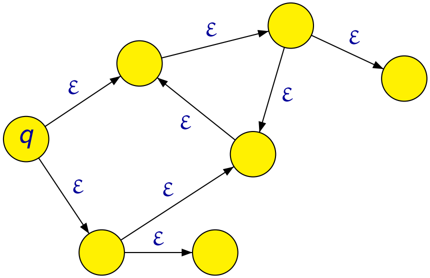
where:

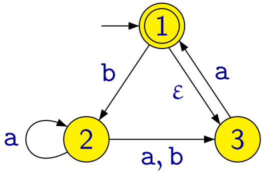
- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ is a **transition function**
- $I \subseteq Q$ is a set of **initial states**
- $F \subseteq Q$ is a set of **accepting states**

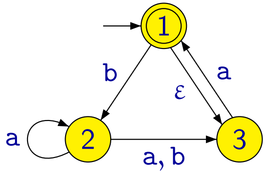
Remark: NFA can be viewed as a special case of GNFA, where $\delta(q, \varepsilon) = \emptyset$ for all $q \in Q$.

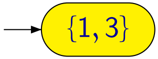
Transformation to a Deterministic Finite Automaton

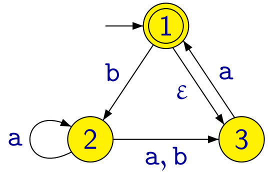
A generalized nondeterministic finite automaton can be transformed into a deterministic one using a similar construction as a nondeterministic finite automaton with the difference that we add to sets of states also all states that are reachable from already added states by some sequence of ε-transitions.

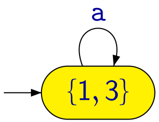


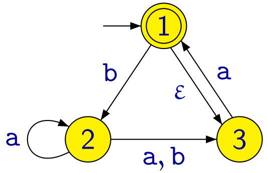

1
2
3
a
b
ε
a
a, b

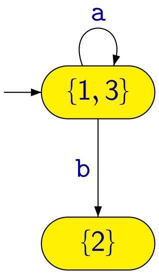

1
2
3
a
b
ε
a
a, b

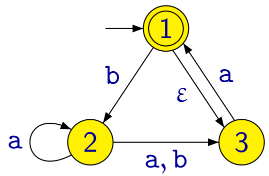

$\{1, 3\}$

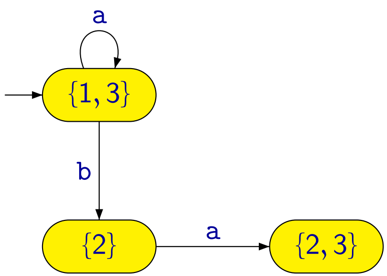

1
b
ε
a
a
2
a, b
3

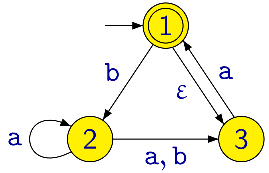

a
{1, 3}

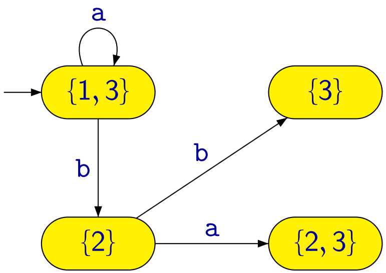

1
2
3
a
b
ε
a
a, b

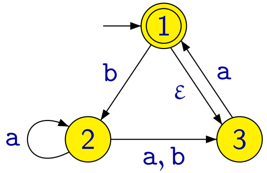

a
{1, 3}
b
{2}

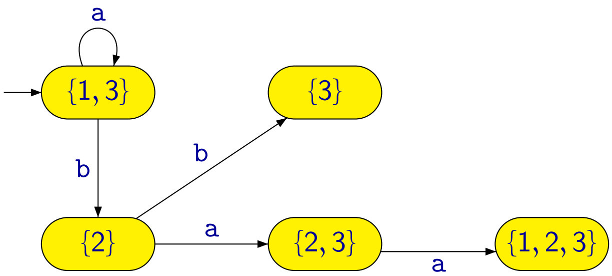

1
b
ε
a
a
2
3
a, b

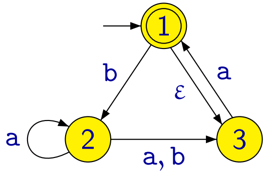

a
{1, 3}
b
{2}
a
{2, 3}

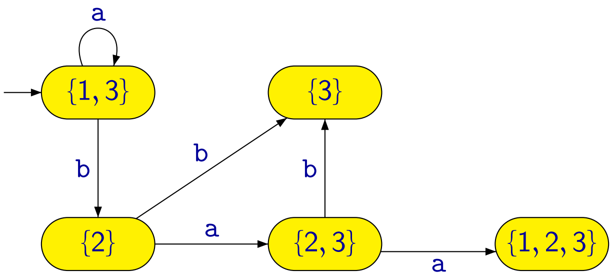

1
b
ε
a
a
2
3
a, b

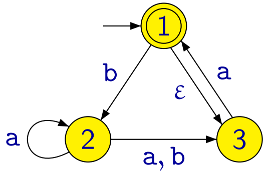

a
{1, 3}
{3}
b
b
{2}
a
{2, 3}

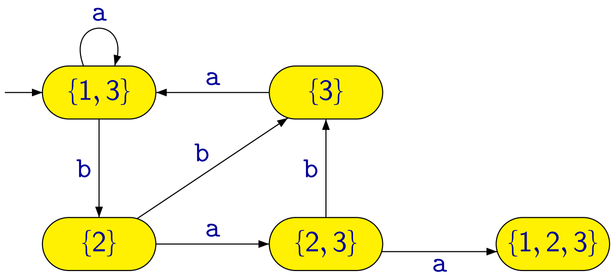

1
2
3
b
a
ε
a
a, b

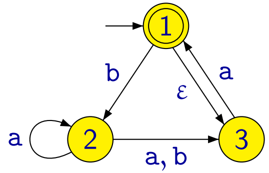

a
{1, 3}
{3}
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

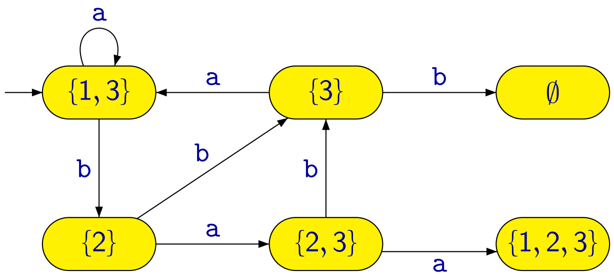

1
b
ε
a
a
2
3
a, b

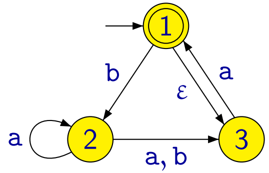

a
{1, 3}
{3}
b
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

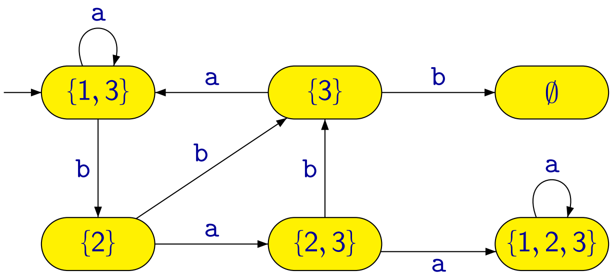

1
b
ε
a
a
2
a, b
3

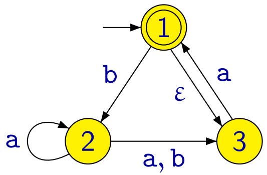

a
{1, 3}
a
{3}
b
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

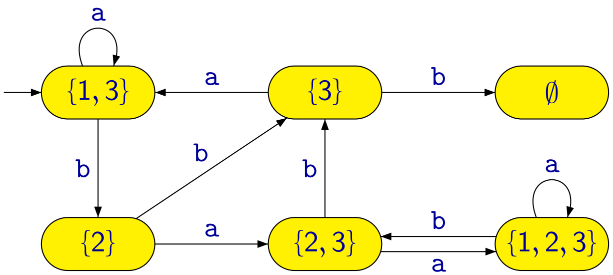

1
b
ε
a
a
2
3
a, b

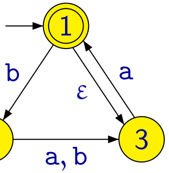

a
{1, 3}
a
{3}
b
∅
b
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

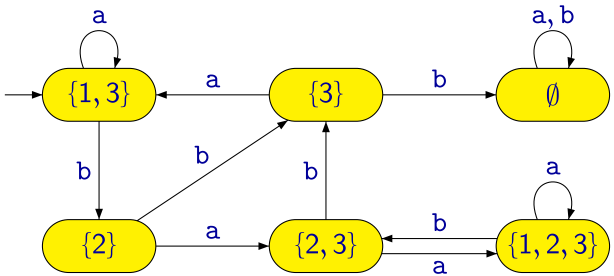

1
b
ε
a
a
2
a, b
3

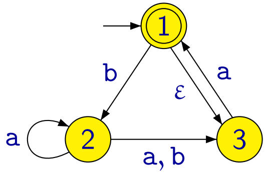

a
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
a
{1, 2, 3}

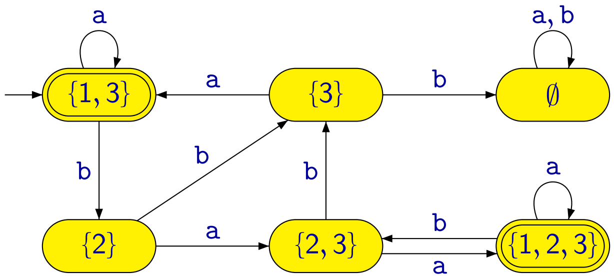

1
b
ε
a
a
2
3
a, b


a
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
b
a
{1, 2, 3}


1
b
ε
a
a
2
3
a, b


a
a, b
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
b
a
{1, 2, 3}


1
2
3
b
ε
a
a
a, b


$\{1,3\}$
$\{3\}$
$\emptyset$
$\{2\}$
$\{2,3\}$
$\{1,2,3\}$
a
a
a, b
b
b
b
b
a
b
a
a

Transformation of GNFA to DFA

Before formally describing the transition of GNFA to DFA, let us introduce some auxiliary definitions.

Let us assume some given GNFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$.

Let us define the function $\hat{\delta} : \mathcal{P}(Q) \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ so that for $K \subseteq Q$ and $a \in \Sigma \cup \{\varepsilon\}$ there is

$$\hat{\delta}(K, a) = \bigcup_{q \in K} \delta(q, a)$$

Transformation of GNFA to DFA

For $K \subseteq Q$, let $Cl_\varepsilon(K)$ be all the states reachable from the states from the set K by some arbitrary sequence of ε-transitions.

This means that the function $Cl_\varepsilon : \mathcal{P}(Q) \to \mathcal{P}(Q)$ is defined so that for $K \subseteq Q$ is $Cl_\varepsilon(K)$ the smallest (with respect to inclusion) set satisfying the following two conditions:

- $K \subseteq Cl_\varepsilon(K)$
- For each $q \in Cl_\varepsilon(K)$ it holds that $\delta(q, \varepsilon) \subseteq Cl_\varepsilon(K)$.

Remark: Let us note that $Cl_\varepsilon(Cl_\varepsilon(K)) = Cl_\varepsilon(K)$ for arbitrary K.

Let us also note that in the case of NFA (where $\delta(q, \varepsilon) = \emptyset$ for each $q \in Q$) is $Cl_\varepsilon(K) = K$.

Transformation of GNFA to DFA

For a given GNFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ we can now construct DFA $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$, where:

- $Q' = \mathcal{P}(Q)$ (so $K \in Q'$ means that $K \subseteq Q$)
- $\delta' : Q' \times \Sigma \to Q'$ is defined so that for $K \in Q'$ and $a \in \Sigma$:

$$\delta'(K, a) = Cl_\varepsilon(\hat{\delta}(Cl_\varepsilon(K), a))$$

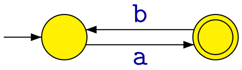
- $q_0' = Cl_\varepsilon(I)$
- $F' = \{K \in Q' \mid Cl_\varepsilon(K) \cap F \neq \emptyset\}$

It is not difficult to verify that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

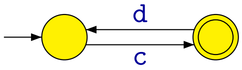
Concatenation of Languages

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

$\mathcal{A}_1$:



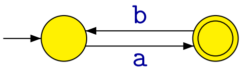
$\mathcal{A}_2$:



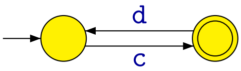
Concatenation of Languages

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

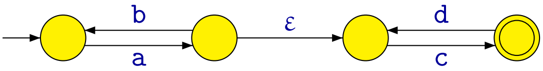
$\mathcal{A}_1$:



$\mathcal{A}_2$:



$\mathcal{A}$:

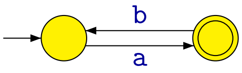


$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cdot \mathcal{L}(\mathcal{A}_2)$

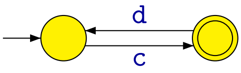
Concatenation of Languages

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

$\mathcal{A}_1$:

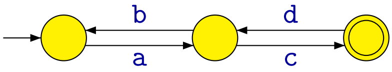


$\mathcal{A}_2$:



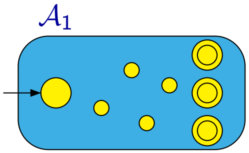
An incorrect construction:

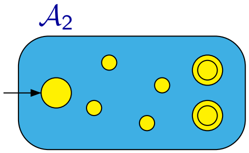
$\mathcal{A}$:



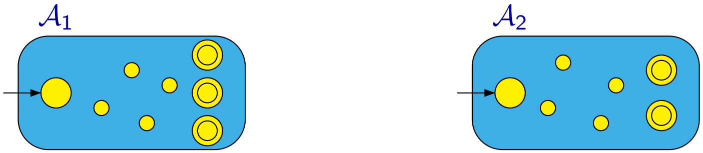
$\mathtt{acdbac} \in \mathcal{L}(\mathcal{A})$ but $\mathtt{acdbac} \notin \mathcal{L}(\mathcal{A}_1) \cdot \mathcal{L}(\mathcal{A}_2)$

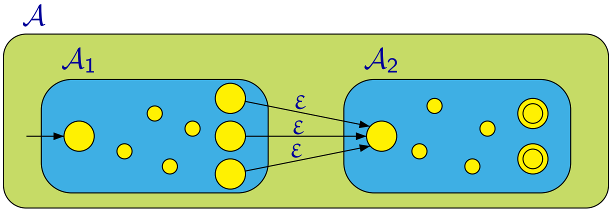
Concatenation of Languages



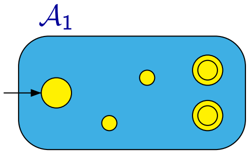


Concatenation of Languages

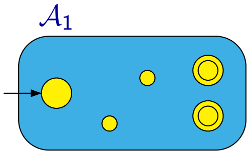


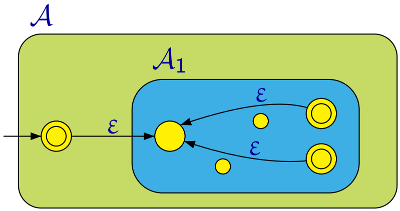


Iteration of a Language



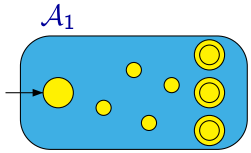
Iteration of a Language

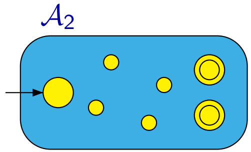




Union of Languages

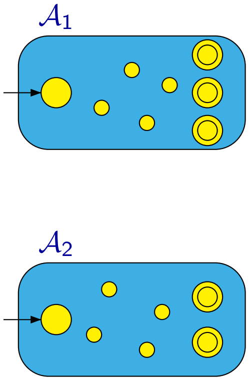
An alternative construction for the union of languages:

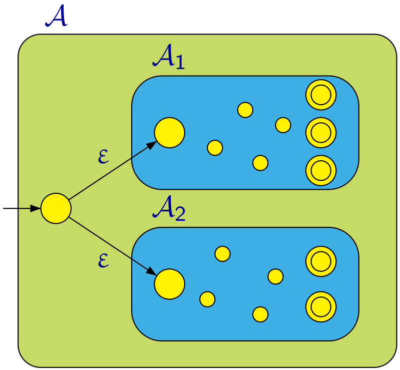




Union of Languages

An alternative construction for the union of languages:





Closure Properties of the Class of Regular Languages

The set of (all) regular languages is closed with respect to:

- union
- intersection
- complement
- concatenation
- iteration
- ...

Transformation of a Regular Expression to a Finite Automaton

Proposition

Every language that can be represented by a regular expression is regular (i.e., it is accepted by some finite automaton).

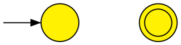
Proof: It is sufficient to show how to construct for a given regular expression α a finite automaton accepting the language $\mathcal{L}(\alpha)$.

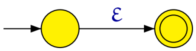
The construction is recursive and proceeds by the structure of the expression α:

- If α is a elementary expression (i.e., $\emptyset$, ε or a):
  - We construct the corresponding automaton directly.
- If α is of the form $(\beta + \gamma)$, $(\beta \cdot \gamma)$ or (β^*):
  - We construct automata accepting languages $\mathcal{L}(\beta)$ and $\mathcal{L}(\gamma)$ recursively.
  - Using these two automata, we construct the automaton accepting the language $\mathcal{L}(\alpha)$.

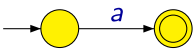
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:


$\emptyset$



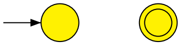
ε

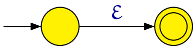


a

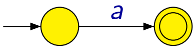
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:


$\emptyset$

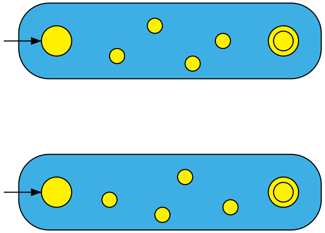


ε



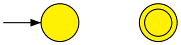
a

The construction for the union:

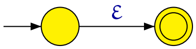


Transformation of a Regular Expression to a Finite Automaton

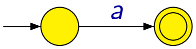
The automata for the elementary expressions:



$\emptyset$

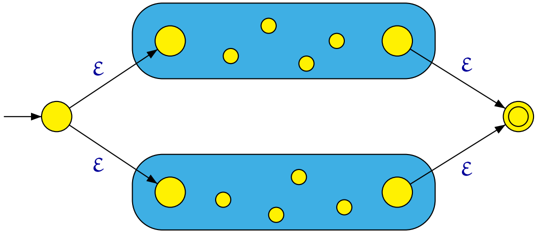


ε



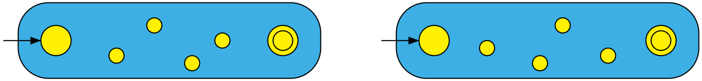
a

The construction for the union:



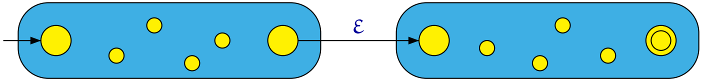
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:



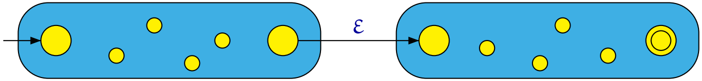
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

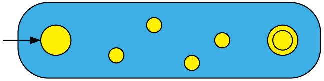


Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

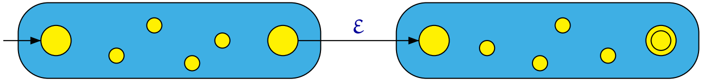


The construction for the iteration:

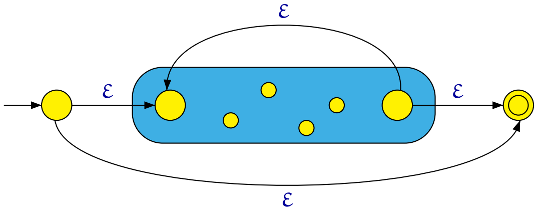


Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:



The construction for the iteration:

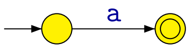


Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a} + \mathtt{b}) \cdot \mathtt{b})^*$:

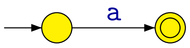
Transformation of a Regular Expression to a Finite Automaton

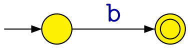
Example: The construction of an automaton for expression $((\mathtt{a} + \mathtt{b}) \cdot \mathtt{b})^*$:



Transformation of a Regular Expression to a Finite Automaton

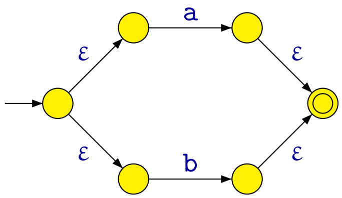
Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:





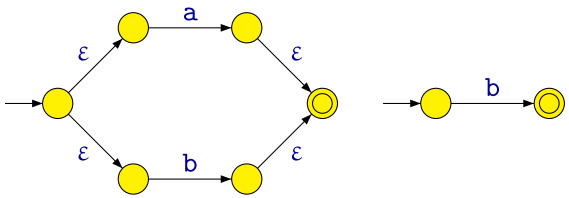
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:



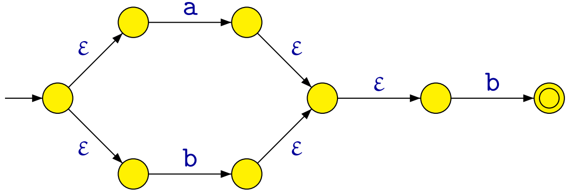
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a} + \mathtt{b}) \cdot \mathtt{b})^*$:



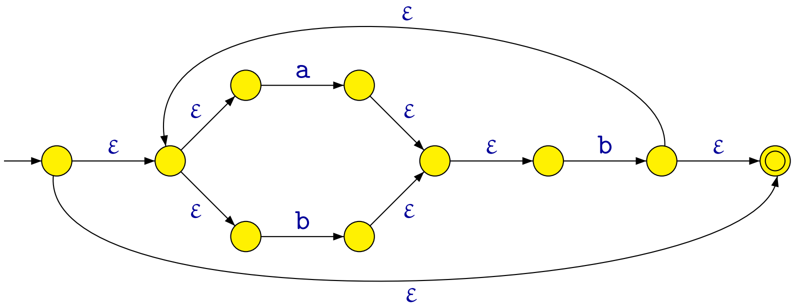
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:



Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:



Transformation of a Regular Expression to a Finite Automaton

If an expression α consists of n symbols (not counting parenthesis) then the resulting automaton has:

- at most $2n$ states,
- at most $4n$ transitions.

Remark: By transforming the generalized nondeterministic automaton into a deterministic one, the number of states can grow exponentially, i.e., the resulting automaton can have up to $2^{2n} = 4^n$ states.

Transformation of an Automaton to a Regular Expression

Proposition

Every regular language can be represented by some regular expression.

Proof: It is sufficient to show how to construct for a given finite automaton $\mathcal{A}$ a regular expression α such that $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A})$.

- We modify $\mathcal{A}$ in such a way that ensures it has exactly one initial and exactly one accepting state.
- Its states will be removed one by one.
- Its transitions will be labelled with regular expressions.
- The resulting automaton will have only two states – the initial and the accepting, and only one transition labelled with the resulting regular expression.

Transformation of an Automaton to a Regular Expression

The main idea: If a state q is removed, for every pair of remaining states q_j, q_k we extend the label on a transition from q_j to q_k by a regular expression representing paths from q_j to q_k going through q.

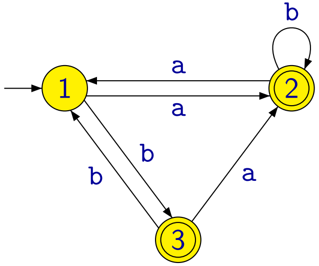
$$\begin{tikzcd}
q_j \arrow[rr, "\alpha"] \arrow[dr, "\beta"'] & & q_k \\
& q \arrow[ur, "\delta"'] \arrow[loop below, "\gamma"] &
\end{tikzcd}$$

After removing of the state q:

$$\begin{tikzcd}
q_j \arrow[r, "\alpha + \beta\gamma^*\delta"] & q_k
\end{tikzcd}$$

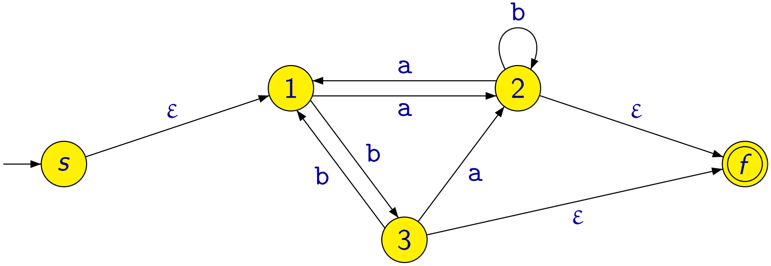
Transformation of an Automaton to a Regular Expression

Example:



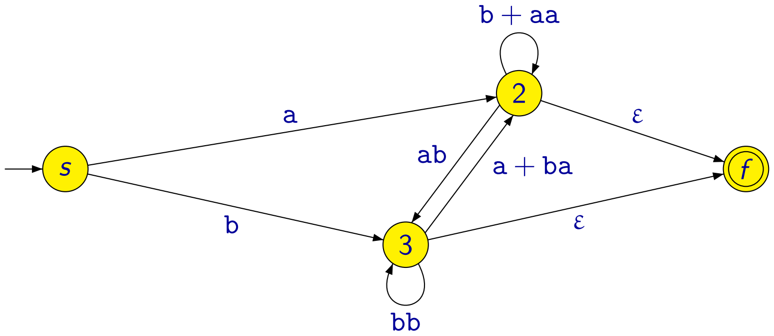
Transformation of an Automaton to a Regular Expression

Example:



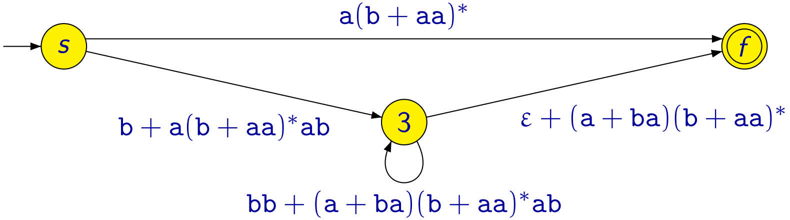
Transformation of an Automaton to a Regular Expression

Example:



Transformation of an Automaton to a Regular Expression

Example:

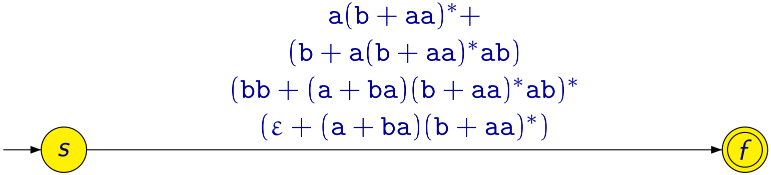


$\mathtt{a(b+aa)^*}$
s
f
3
$\mathtt{b+a(b+aa)^*ab}$
$\varepsilon + \mathtt{(a+ba)(b+aa)^*}$
$\mathtt{bb+(a+ba)(b+aa)^*ab}$

Transformation of an Automaton to a Regular Expression

Example:

$$\begin{tikzcd} \arrow[r] & s \arrow[rrr, "\begin{array}{c}\mathtt{a}(\mathtt{b}+\mathtt{aa})^*+\\(\mathtt{b}+\mathtt{a}(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})\\(\mathtt{bb}+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})^*\\(\varepsilon+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*)\end{array}"] & & & f \end{tikzcd}$$



Equivalence of Finite Automata and Regular Expressions

Theorem

A language is regular iff it can be represented by a regular expression.

Nonregular Languages

Not all languages are regular.

There are languages for which there exist no finite automata accepting them.

Examples of nonregular languages:

- $L_1 = \{a^n b^n \mid n \geq 0\}$
- $L_2 = \{ww \mid w \in \{a,b\}^*\}$
- $L_3 = \{ww^R \mid w \in \{a,b\}^*\}$

Remark: The existence of nonregular languages is already apparent from the fact that there are only countably many (nonisomorphic) automata working over some alphabet Σ but there are uncountably many languages over the alphabet Σ.

Nonregular Languages

How to prove that some language L is not regular?

A language is not regular if there is no automaton (i.e., it is not possible to construct an automaton) accepting the language.

But how to prove that something does not exist?

Nonregular Languages

How to prove that some language L is not regular?

A language is not regular if there is no automaton (i.e., it is not possible to construct an automaton) accepting the language.

But how to prove that something does not exist?

The answer: By contradiction.

E.g., we can assume there is some automaton $\mathcal{A}$ accepting the language L, and show that this assumption leads to a contradiction.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Let $|Q| = n$.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Let $|Q| = n$.

Consider word $z = a^n b^n$.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Let $|Q| = n$.

Consider word $z = a^n b^n$.

Since $z \in L$, there must be an accepting computation of the automaton $\mathcal{A}$

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

where q_0 is an initial state, and $q_{2n} \in F$.

Nonregular Languages

Consider now the first $n+1$ states of the computation

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n+1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

Nonregular Languages

Consider now the first $n+1$ states of the computation

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n+1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

It is an application of so called **pigeonhole principle**.

Pigeonhole principle

If we have $n+1$ pigeons in n holes then there is at least one hole containing at least two pigeons.

Nonregular Languages

Consider now the first $n+1$ states of the computation

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n+1$ elements.

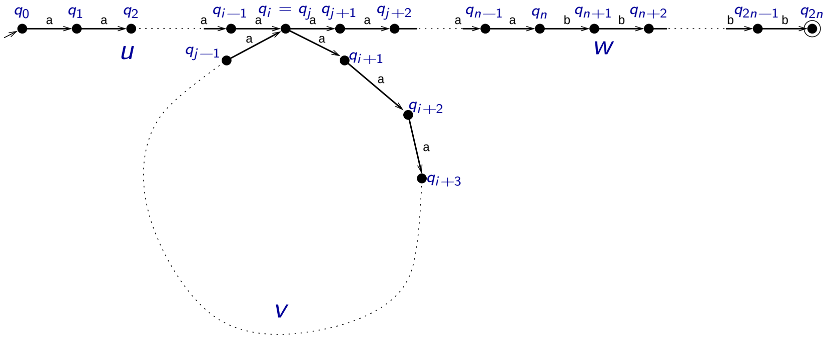
This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

I.e., there are indexes i, j such that $0 \leq i < j \leq n$ and

$$q_i = q_j$$

which means that the automaton $\mathcal{A}$ must go through a cycle when reading the symbols a in the word $z = a^n b^n$.

Nonregular Languages



The word $z = a^n b^n$ can be divided into three parts u, v, w such that $z = uvw$:

$$u = a^i \qquad v = a^{j-i} \qquad w = a^{n-j} b^n$$

Nonregular Languages

For the words $u = a^i$, $v = a^{j-i}$, and $w = a^{n-j}b^n$ we have

$$q_0 \xrightarrow{u} q_i \qquad\qquad q_i \xrightarrow{v} q_j \qquad\qquad q_j \xrightarrow{w} q_{2n}$$

Let r be the length of the word v, i.e., $r = j - i$ (obviously $r > 0$, due to $i < j$).

Since $q_i = q_j$, the automaton accepts word $uw = a^{n-r}b^n$ that does not belong to L:

$$q_0 \xrightarrow{u} q_i \xrightarrow{w} q_{2n}$$

The word $uvvw = a^{n+r}b^n$, that also does not belong to L, is accepted too:

$$q_0 \xrightarrow{u} q_i \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{w} q_{2n}$$

Nonregular Languages

Similarly we can show that every word of the form $uvvvv \cdots vvw$, i.e., of the form $uv^k w$ for some $k \geq 0$, is accepted by the automaton $\mathcal{A}$:

$$q_0 \xrightarrow{u} q_i \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{v} \cdots \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{w} q_{2n}$$

A word of the form $uv^k w$ looks as follows: $a^{n-r+rk} b^n$.

Since $r > 0$, the following equivalence holds only for $k = 1$:

$$n - r + rk = n$$

This means that if $k \neq 1$ then $uv^k w$ does not belong to the language L.
However, the automaton $\mathcal{A}$ accepts each such word, which is a contradiction with the assumption that $\mathcal{L}(\mathcal{A}) = \{a^n b^n \mid n \geq 0\}$.

Context-Free Grammars

Context-Free Grammars

Example: We would like to describe a language of arithmetic expressions, containing expressions such as:

`175`   `(9+15)`   `(((10-4)*((1+34)+2))/(3+(-37)))`

For simplicity we assume that:

- Expressions are fully parenthesized.
- The only arithmetic operations are "`+`", "`-`", "`*`", "`/`" and unary "`-`".
- Values of operands are natural numbers written in decimal — a number is represented as a non-empty sequence of digits.

Alphabet: $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)\}$

Context-Free Grammars

Example (cont.): A description by an inductive definition:

- **Digit** is any of characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- **Number** is a non-empty sequence of digits, i.e.:
  - If α is a digit then α is a number.
  - If α is a digit and β is a number then also $\alpha\beta$ is a number.
- **Expression** is a sequence of symbols constructed according to the following rules:
  - If α is a number then α is an expression.
  - If α is an expression then also $(-\alpha)$ is an expression.
  - If α and β are expressions then also $(\alpha+\beta)$ is an expression.
  - If α and β are expressions then also $(\alpha-\beta)$ is an expression.
  - If α and β are expressions then also $(\alpha*\beta)$ is an expression.
  - If α and β are expressions then also (α/β) is an expression.

Context-Free Grammars

Example (cont.): The same information that was described by the previous inductive definition can be represented by a **context-free grammar**:

New auxiliary symbols, called **nonterminals**, are introduced:

- D — stands for an arbitrary digit
- C — stands for an arbitrary number
- E — stands for an arbitrary expression

$$\begin{array}{llll}
D \longrightarrow 0 & D \longrightarrow 5 & & E \longrightarrow C \\
D \longrightarrow 1 & D \longrightarrow 6 & & E \longrightarrow (\texttt{-}E) \\
D \longrightarrow 2 & D \longrightarrow 7 & C \longrightarrow D & E \longrightarrow (E\texttt{+}E) \\
D \longrightarrow 3 & D \longrightarrow 8 & C \longrightarrow DC & E \longrightarrow (E\texttt{-}E) \\
D \longrightarrow 4 & D \longrightarrow 9 & & E \longrightarrow (E\texttt{*}E) \\
 & & & E \longrightarrow (E\texttt{/}E)
\end{array}$$

Context-Free Grammars

Example (cont.): Written in a more succinct way:

$$\begin{aligned}
D &\longrightarrow \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \mathtt{3} \mid \mathtt{4} \mid \mathtt{5} \mid \mathtt{6} \mid \mathtt{7} \mid \mathtt{8} \mid \mathtt{9} \\
C &\longrightarrow D \mid DC \\
E &\longrightarrow C \mid \mathtt{(-}E\mathtt{)} \mid \mathtt{(}E\mathtt{+}E\mathtt{)} \mid \mathtt{(}E\mathtt{-}E\mathtt{)} \mid \mathtt{(}E\mathtt{*}E\mathtt{)} \mid \mathtt{(}E\mathtt{/}E\mathtt{)}
\end{aligned}$$

Context-Free Grammars

Example: A language where words are (possibly empty) sequences of expressions described in the previous example, where individual expressions are separated by commas (the alphabet must be extended with symbol "$\texttt{,}$"):

$$\begin{aligned}
S &\rightarrow T \mid \varepsilon \\
T &\rightarrow E \mid E\,\texttt{,}\,T \\
D &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
C &\rightarrow D \mid DC \\
E &\rightarrow C \mid \texttt{(-}E\texttt{)} \mid \texttt{(}E\texttt{+}E\texttt{)} \mid \texttt{(}E\texttt{-}E\texttt{)} \mid \texttt{(}E\texttt{*}E\texttt{)} \mid \texttt{(}E\texttt{/}E\texttt{)}
\end{aligned}$$

Context-Free Grammars

Example: Statements of some programming language (a fragment of a grammar):

$$
\begin{aligned}
S &\rightarrow E\texttt{;} \mid T \mid \texttt{if (}E\texttt{)}\ S \mid \texttt{if (}E\texttt{)}\ S\ \texttt{else}\ S \\
&\quad \mid \texttt{while (}E\texttt{)}\ S \mid \texttt{do}\ S\ \texttt{while (}E\texttt{);} \mid \texttt{for (}F\texttt{;}\ F\texttt{;}\ F\texttt{)}\ S \\
&\quad \mid \texttt{return}\ F\texttt{;} \\
T &\rightarrow \texttt{\{}\ U\ \texttt{\}} \\
U &\rightarrow \varepsilon \mid SU \\
F &\rightarrow \varepsilon \mid E \\
E &\rightarrow \quad \ldots
\end{aligned}
$$

Remark:

- S — statement
- T — block of statements
- U — sequence of statements
- E — expression
- F — optional expression that can be omitted

Context-Free Grammars

Formally, a **context-free grammar** is a tuple

$$\mathcal{G} = (\Pi, \Sigma, S, P)$$

where:

- Π is a finite set of **nonterminal symbols** (**nonterminals**)
- Σ is a finite set of **terminal symbols** (**terminals**), where $\Pi \cap \Sigma = \emptyset$
- $S \in \Pi$ is an **initial nonterminal**
- $P \subseteq \Pi \times (\Pi \cup \Sigma)^*$ is a finite set of **rewrite rules**

Context-Free Grammars

Remarks:

- We will use uppercase letters A, B, C, ... to denote nonterminal symbols.
- We will use lowercase letters a, b, c, ... or digits 0, 1, 2, ... to denote terminal symbols.
- We will use lowercase Greek letters α, β, γ, ... do denote strings from $(\Pi \cup \Sigma)^*$.
- We will use the following notation for rules instead of (A, α)

$$A \rightarrow \alpha$$

A – left-hand side of the rule
α – right-hand side of the rule

Context-Free Grammars

Example: Grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ where

- $\Pi = \{A, B, C\}$
- $\Sigma = \{a, b\}$
- $S = A$
- P contains rules

$$
\begin{aligned}
A &\longrightarrow aBBb \\
A &\longrightarrow AaA \\
B &\longrightarrow \varepsilon \\
B &\longrightarrow bCA \\
C &\longrightarrow AB \\
C &\longrightarrow a \\
C &\longrightarrow b
\end{aligned}
$$

Context-Free Grammars

Remark: If we have more rules with the same left-hand side, as for example

$$A \longrightarrow \alpha_1 \qquad A \longrightarrow \alpha_2 \qquad A \longrightarrow \alpha_3$$

we can write them in a more succinct way as

$$A \longrightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$$

For example, the rules of the grammar from the previous slide can be written as

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ B &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\to aBBb \mid AaA \\ B &\to \varepsilon \mid bCA \\ C &\to AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

A

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} \underline{A} &\rightarrow aBBb \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

$\underline{A}$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} \underline{A} &\rightarrow \underline{aBBb} \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

$\underline{A} \Rightarrow \underline{aBBb}$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ B &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow a\underline{B}Bb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \varepsilon \mid \underline{bCA} \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow a\underline{B}Bb \Rightarrow a\underline{bCA}Bb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ B &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} \underline{A} &\longrightarrow aBBb \mid AaA \\ B &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} \underline{A} &\rightarrow \underline{aBBb} \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb \Rightarrow abC\underline{aBBb}Bb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ \underline{B} &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \underline{\varepsilon} \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb \Rightarrow abCaBbBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ B &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ \underline{C} &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ B &\longrightarrow \varepsilon \mid bCA \\ \underline{C} &\longrightarrow AB \mid a \mid \underline{b} \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb \Rightarrow ab\underline{b}aBbBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \underline{\varepsilon} \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b \Rightarrow abbaBbb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\to aBBb \mid AaA \\ B &\to \varepsilon \mid bCA \\ C &\to AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$

For example, the word *abbabb* can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ \underline{B} &\rightarrow \underline{\varepsilon} \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb \Rightarrow abbabb$

Context-Free Grammars

Grammars are used for generating words.

Example: $\mathcal{G} = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ B &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$

For example, the word $abbabb$ can be in grammar $\mathcal{G}$ generated as follows:
$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb$

Context-Free Grammars

On strings from $(\Pi \cup \Sigma)^*$ we define relation $\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$ such that

$$\alpha \Rightarrow \alpha'$$

iff $\alpha = \beta_1 A \beta_2$ and $\alpha' = \beta_1 \gamma \beta_2$ for some $\beta_1, \beta_2, \gamma \in (\Pi \cup \Sigma)^*$ and $A \in \Pi$ where $(A \rightarrow \gamma) \in P$.

Example: If $(B \rightarrow bCA) \in P$ then

$$aCBbA \Rightarrow aC\,bCA\,bA$$

Remark: Informally, $\alpha \Rightarrow \alpha'$ means that it is possible to derive α' from α by one step where an occurrence of some nonterminal A in α is replaced with the right-hand side of some rule $A \rightarrow \gamma$ with A on the left-hand side.

Context-Free Grammars

On strings from $(\Pi \cup \Sigma)^*$ we define relation $\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$ such that

$$\alpha \Rightarrow \alpha'$$

iff $\alpha = \beta_1 A \beta_2$ and $\alpha' = \beta_1 \gamma \beta_2$ for some $\beta_1, \beta_2, \gamma \in (\Pi \cup \Sigma)^*$ and $A \in \Pi$ where $(A \rightarrow \gamma) \in P$.

Example: If $(B \rightarrow bCA) \in P$ then

$$aC\underline{B}bA \Rightarrow aC\underline{bCA}bA$$

Remark: Informally, $\alpha \Rightarrow \alpha'$ means that it is possible to derive α' from α by one step where an occurrence of some nonterminal A in α is replaced with the right-hand side of some rule $A \rightarrow \gamma$ with A on the left-hand side.

Context-Free Grammars

A **derivation** of length n is a sequence $\beta_0, \beta_1, \beta_2, \cdots, \beta_n$, where $\beta_i \in (\Pi \cup \Sigma)^*$, and where $\beta_{i-1} \Rightarrow \beta_i$ for all $1 \leq i \leq n$, which can be written more succinctly as

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$$

The fact that for given $\alpha, \alpha' \in (\Pi \cup \Sigma)^*$ and $n \in \mathbb{N}$ there exists some derivation $\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$, where $\alpha = \beta_0$ and $\alpha' = \beta_n$, is denoted

$$\alpha \Rightarrow^n \alpha'$$

The fact that $\alpha \Rightarrow^n \alpha'$ for some $n \geq 0$, is denoted

$$\alpha \Rightarrow^* \alpha'$$

Remark: Relation $\Rightarrow^*$ is the reflexive and transitive closure of relation $\Rightarrow$ (i.e., the smallest reflexive and transitive relation containing relation $\Rightarrow$).

Context-Free Grammars

Sentential forms are those $\alpha \in (\Pi \cup \Sigma)^*$, for which

$$S \Rightarrow^* \alpha$$

where S is the initial nonterminal.

Context-Free Grammars

A **language** $\mathcal{L}(\mathcal{G})$ generated by a grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is the set of all words over alphabet Σ that can be derived by some derivation from the initial nonterminal S using rules from P, i.e.,

$$\mathcal{L}(\mathcal{G}) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Definition

A language L is **context-free** if there exists some context-free grammar $\mathcal{G}$ such that $L = \mathcal{L}(\mathcal{G})$.

Context-Free Grammars

Example: We want to construct a grammar generating the language

$$L = \{a^n b^n \mid n \geq 0\}$$

Context-Free Grammars

Example: We want to construct a grammar generating the language

$$L = \{a^n b^n \mid n \geq 0\}$$

Grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ where $\Pi = \{S\}$, $\Sigma = \{a, b\}$, and P contains

$$S \longrightarrow \varepsilon \mid aSb$$

Context-Free Grammars

Example: We want to construct a grammar generating the language

$$L = \{a^n b^n \mid n \geq 0\}$$

Grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ where $\Pi = \{S\}$, $\Sigma = \{a, b\}$, and P contains

$$S \longrightarrow \varepsilon \mid aSb$$

$$\begin{array}{l}
S \Rightarrow \varepsilon \\
S \Rightarrow aSb \Rightarrow ab \\
S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb \\
S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb \\
S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaabbbb \\
\quad \ldots
\end{array}$$

Context-Free Grammars

Example: We want to construct a grammar generating the language consisting of all palindroms over the alphabet $\{a, b\}$, i.e.,

$$L = \{w \in \{a, b\}^* \mid w = w^R\}$$

Remark: w^R denotes the **reverse** of a word w, i.e., the word w written backwards.

Context-Free Grammars

Example: We want to construct a grammar generating the language consisting of all palindroms over the alphabet $\{a, b\}$, i.e.,

$$L = \{w \in \{a, b\}^* \mid w = w^R\}$$

Remark: w^R denotes the **reverse** of a word w, i.e., the word w written backwards.

Solution:

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

Context-Free Grammars

Example: We want to construct a grammar generating the language consisting of all palindroms over the alphabet $\{a, b\}$, i.e.,

$$L = \{w \in \{a, b\}^* \mid w = w^R\}$$

Remark: w^R denotes the **reverse** of a word w, i.e., the word w written backwards.

Solution:

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaaba$$

Context-Free Grammars

Example: We want to construct a grammar generating the language L consisting of all correctly parenthesised sequences of symbols '(' and ')'.

For example $(()())(()) \in L$ but $)()) \notin L$.

Context-Free Grammars

Example: We want to construct a grammar generating the language L consisting of all correctly parenthesised sequences of symbols '(' and ')'.

For example $(()())(()) \in L$ but $)()) \notin L$.

Solution:

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

Context-Free Grammars

Example: We want to construct a grammar generating the language L consisting of all correctly parenthesised sequences of symbols '$($' and '$)$'.

For example $(()())(()) \in L$ but $)()) \notin L$.

Solution:

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (SS)(S) \Rightarrow ((S)S)(S) \Rightarrow$
$(()S)(S) \Rightarrow (()(S))(S) \Rightarrow (()())(S) \Rightarrow (()())((S)) \Rightarrow$
$(()())(())$

Context-Free Grammars

Example: We want to construct a grammar generating the language L consisting of all correctly constructed arithmetic experessions where operands are always of the form 'a' and where symbols $+$ and $*$ can be used as operators.

For example $(a + a) * a + (a * a) \in L$.

Context-Free Grammars

Example: We want to construct a grammar generating the language L consisting of all correctly constructed arithmetic experessions where operands are always of the form 'a' and where symbols $+$ and $*$ can be used as operators.

For example $(a + a) * a + (a * a) \in L$.

Solution:

$$E \rightarrow a \mid E + E \mid E * E \mid (E)$$

Context-Free Grammars

Example: We want to construct a grammar generating the language L consisting of all correctly constructed arithmetic experessions where operands are always of the form 'a' and where symbols $+$ and $*$ can be used as operators.

For example $(a + a) * a + (a * a) \in L$.

Solution:

$$E \rightarrow a \mid E + E \mid E * E \mid (E)$$

$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow (E) * E + E \Rightarrow (E + E) * E + E \Rightarrow (a + E) * E + E \Rightarrow (a + a) * E + E \Rightarrow (a + a) * a + E \Rightarrow (a + a) * a + (E) \Rightarrow (a + a) * a + (E * E) \Rightarrow (a + a) * a + (a * E) \Rightarrow (a + a) * a + (a * a)$

Derivation Tree

$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
B &\longrightarrow \varepsilon \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$

Derivation Tree

A

$$
\begin{aligned}
A &\rightarrow aBBb \mid AaA \\
B &\rightarrow \varepsilon \mid bCA \\
C &\rightarrow AB \mid a \mid b
\end{aligned}
$$

A

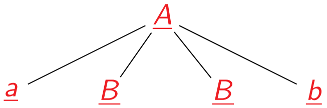
Derivation Tree

$\underline{A}$

$$
\begin{aligned}
\underline{A} &\longrightarrow aBBb \mid AaA \\
B &\longrightarrow \varepsilon \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$

$\underline{A}$

Derivation Tree



$$\underline{A} \longrightarrow \underline{aBBb} \mid AaA$$
$$B \longrightarrow \varepsilon \mid bCA$$
$$C \longrightarrow AB \mid a \mid b$$

$\underline{A} \Rightarrow \underline{aBBb}$

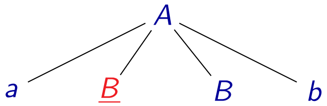
Derivation Tree

$$\begin{tikzcd}[column sep=small] & & & A \arrow[dlll, no head] \arrow[dl, no head] \arrow[dr, no head] \arrow[drrr, no head] & & & \\ a & & B & & B & & b \end{tikzcd}$$

$A \longrightarrow aBBb \mid AaA$
$B \longrightarrow \varepsilon \mid bCA$
$C \longrightarrow AB \mid a \mid b$

$A \Rightarrow aBBb$

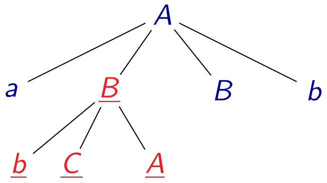
Derivation Tree



$A \longrightarrow aBBb \mid AaA$
$\underline{B} \longrightarrow \varepsilon \mid bCA$
$C \longrightarrow AB \mid a \mid b$

$A \Rightarrow a\underline{B}Bb$

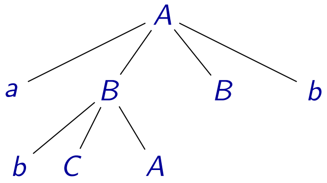
Derivation Tree



$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
\underline{B} &\longrightarrow \varepsilon \mid \underline{bCA} \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$

$$A \Rightarrow a\underline{B}Bb \Rightarrow a\underline{bCA}Bb$$

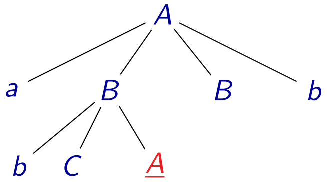
Derivation Tree



$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
B &\longrightarrow \varepsilon \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$

$$A \Rightarrow aBBb \Rightarrow abCABb$$

Derivation Tree

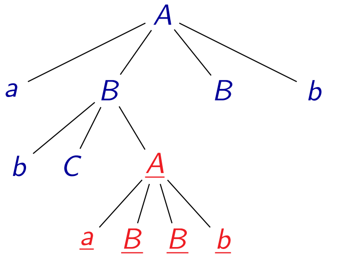


$\underline{A} \longrightarrow aBBb \mid AaA$
$B \longrightarrow \varepsilon \mid bCA$
$C \longrightarrow AB \mid a \mid b$

$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb$

Derivation Tree

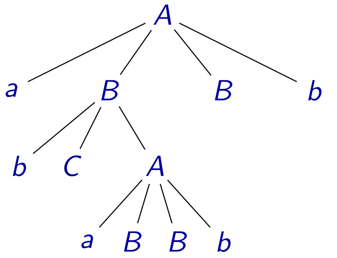
$\underline{A} \longrightarrow \underline{aBBb} \mid AaA$
$B \longrightarrow \varepsilon \mid bCA$
$C \longrightarrow AB \mid a \mid b$



$$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb \Rightarrow abC\underline{aBBb}Bb$$

Derivation Tree

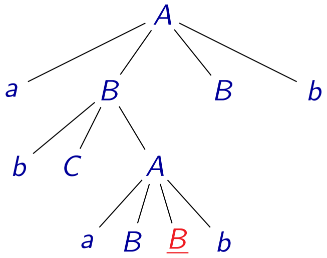
$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
B &\longrightarrow \varepsilon \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$$

Derivation Tree

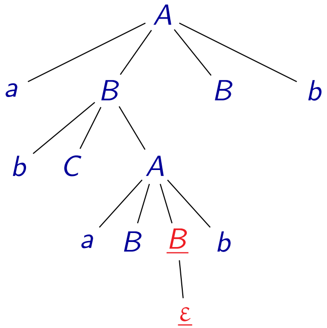
$$\begin{aligned} A &\rightarrow aBBb \mid AaA \\ \underline{B} &\rightarrow \varepsilon \mid bCA \\ C &\rightarrow AB \mid a \mid b \end{aligned}$$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb$$

Derivation Tree

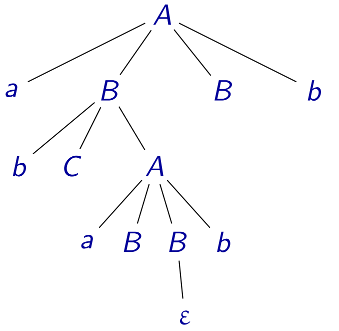
$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
\underline{B} &\longrightarrow \underline{\varepsilon} \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb \Rightarrow abCaBbBb$$

Derivation Tree

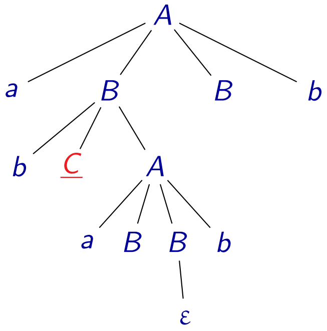
$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
B &\longrightarrow \varepsilon \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$$

Derivation Tree

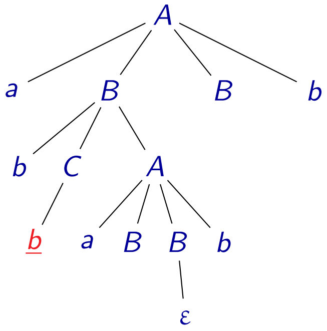
$A \longrightarrow aBBb \mid AaA$
$B \longrightarrow \varepsilon \mid bCA$
$\underline{C} \longrightarrow AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb$

Derivation Tree

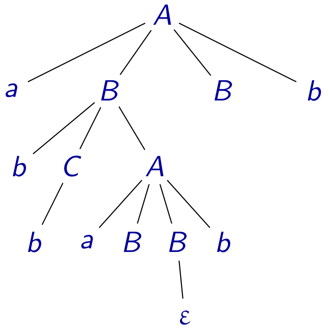
$A \longrightarrow aBBb \mid AaA$
$B \longrightarrow \varepsilon \mid bCA$
$\underline{C} \longrightarrow AB \mid a \mid \underline{b}$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb \Rightarrow ab\underline{b}aBbBb$$

Derivation Tree

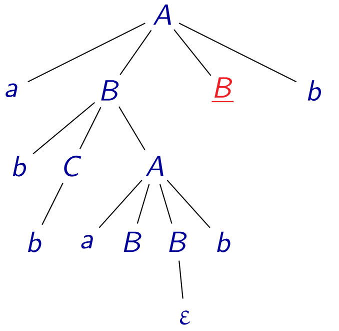
$A \rightarrow aBBb \mid AaA$
$B \rightarrow \varepsilon \mid bCA$
$C \rightarrow AB \mid a \mid b$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$$

Derivation Tree

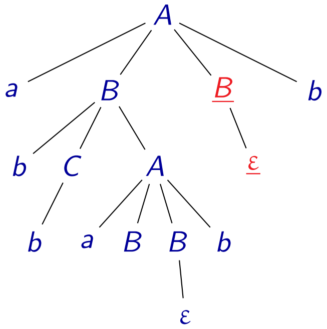
$$\begin{aligned} A &\longrightarrow aBBb \mid AaA \\ \underline{B} &\longrightarrow \varepsilon \mid bCA \\ C &\longrightarrow AB \mid a \mid b \end{aligned}$$



$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b$$

Derivation Tree

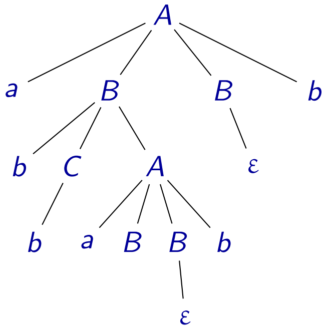
$$A \longrightarrow aBBb \mid AaA$$
$$\underline{B} \longrightarrow \underline{\varepsilon} \mid bCA$$
$$C \longrightarrow AB \mid a \mid b$$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b \Rightarrow abbaBbb$

Derivation Tree

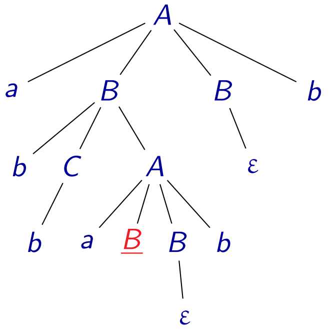
$$A \longrightarrow aBBb \mid AaA$$
$$B \longrightarrow \varepsilon \mid bCA$$
$$C \longrightarrow AB \mid a \mid b$$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb$

Derivation Tree

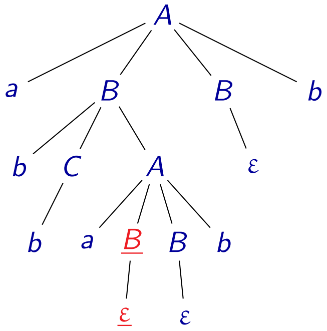
$A \longrightarrow aBBb \mid AaA$
$\underline{B} \longrightarrow \varepsilon \mid bCA$
$C \longrightarrow AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb$

Derivation Tree

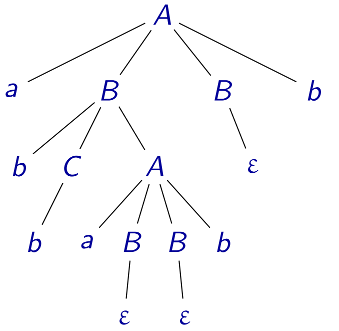
$$
\begin{aligned}
A &\longrightarrow aBBb \mid AaA \\
\underline{B} &\longrightarrow \underline{\varepsilon} \mid bCA \\
C &\longrightarrow AB \mid a \mid b
\end{aligned}
$$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb \Rightarrow abbabb$

Derivation Tree

$A \longrightarrow aBBb \mid AaA$
$B \longrightarrow \varepsilon \mid bCA$
$C \longrightarrow AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb$

Derivation Tree

For each derivation there is some **derivation tree**:

- Nodes of the tree are labelled with terminals and nonterminals.
- The root of the tree is labelled with the initial nonterminal.
- The leafs of the tree are labelled with terminals or with symbols ε.
- The remaining nodes of the tree are labelled with nonterminals.
- If a node is labelled with some nonterminal A then its children are labelled with the symbols from the right-hand side of some rewriting rule $A \longrightarrow \alpha$.

Left and Right Derivation

$$E \longrightarrow a \mid E + E \mid E * E \mid (E)$$

A **left derivation** is a derivation where in every step we always replace the leftmost nonterminal.

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow \underline{E} * E + E \Rightarrow a * \underline{E} + E \Rightarrow a * a + \underline{E} \Rightarrow a * a + a$$

A **right derivation** is a derivation where in every step we always replace the rightmost nonterminal.

$$\underline{E} \Rightarrow E + \underline{E} \Rightarrow \underline{E} + a \Rightarrow E * \underline{E} + a \Rightarrow \underline{E} * a + a \Rightarrow a * a + a$$

A derivation need not be left or right:

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow E * \underline{E} + E \Rightarrow E * a + \underline{E} \Rightarrow \underline{E} * a + a \Rightarrow a * a + a$$

Left and Right Derivation

- There can be several different derivations corresponding to one derivation tree.
- For every derivation tree, there is exactly one left and exactly one right derivation corresponding to the tree.

Equvalence of Grammars

Grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ are **equivalent** if they generate the same language, i.e., if $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$.

Remark: The problem of equivalence of context-free grammars is algorithmically undecidable. It can be shown that it is not possible to construct an algorithm that would decide for any pair of context-free grammars if they are equivalent or not.

Even the problem to decide if a grammar generates the language Σ^* is algorithmically undecidable.

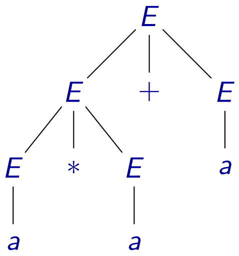
Ambiguous Grammars

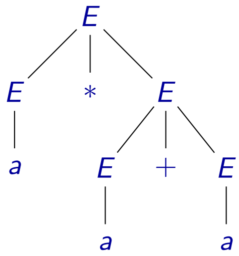
A grammar $\mathcal{G}$ is **ambiguous** if there is a word $w \in \mathcal{L}(\mathcal{G})$ that has two different derivation trees, resp. two different left or two different right derivations.

Example:

$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow a * E + E \Rightarrow a * a + E \Rightarrow a * a + a$

$E \Rightarrow E * E \Rightarrow E * E + E \Rightarrow a * E + E \Rightarrow a * a + E \Rightarrow a * a + a$





Ambiguous Grammars

Sometimes it is possible to replace an ambiguous grammar with a grammar generating the same language but which is not ambiguous.

Example: A grammar

$$E \longrightarrow a \mid E + E \mid E * E \mid (E)$$

can be replaced with the equivalent grammar

$$\begin{aligned} E &\longrightarrow T \mid T + E \\ T &\longrightarrow F \mid F * T \\ F &\longrightarrow a \mid (E) \end{aligned}$$

Remark: If there is no unambiguous grammar equivalent to a given ambiguous grammar, we say it is **inherently ambiguous**.

Context-Free Languages

The class of context-free languages is closed with respect to:

- concatenation
- union
- iteration

The class of context-free languages is not closed with respect to:

- complement
- intersection

Context-Free Languages

We have two grammars $\mathcal{G}_1 = (\Pi_1, \Sigma, S_1, P_1)$ and $\mathcal{G}_2 = (\Pi_2, \Sigma, S_2, P_2)$, and can assume that $\Pi_1 \cap \Pi_2 = \emptyset$ and $S \notin \Pi_1 \cup \Pi_2$.

- Grammar $\mathcal{G}$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}_1) \cdot \mathcal{L}(\mathcal{G}_2)$:

$$\mathcal{G} = (\Pi_1 \cup \Pi_2 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\})$$

- Grammar $\mathcal{G}$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}_1) \cup \mathcal{L}(\mathcal{G}_2)$:

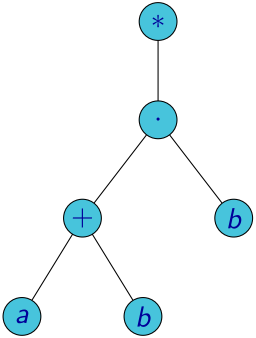
$$\mathcal{G} = (\Pi_1 \cup \Pi_2 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$$

- Grammar $\mathcal{G}$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}_1)^*$:

$$\mathcal{G} = (\Pi_1 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S_1 S\})$$

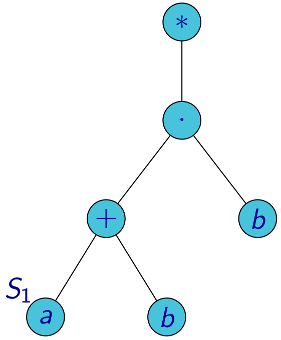
A Context-Free Grammar for a Regular Expression

Example: The construction of a context-free grammar for regular expression $((a + b) \cdot b)^*$:



A Context-Free Grammar for a Regular Expression

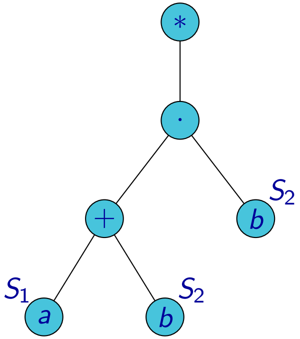
Example: The construction of a context-free grammar for regular expression $((a + b) \cdot b)^*$:



$S_1 \longrightarrow a$

A Context-Free Grammar for a Regular Expression

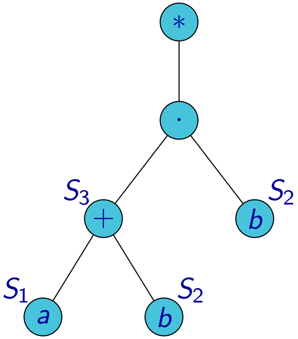
Example: The construction of a context-free grammar for regular expression $((a+b)\cdot b)^*$:



$$S_2 \longrightarrow b$$
$$S_1 \longrightarrow a$$

A Context-Free Grammar for a Regular Expression

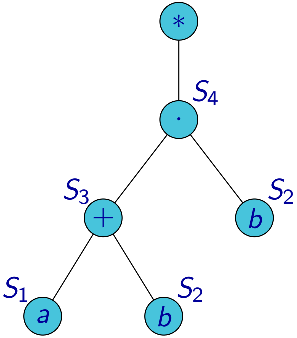
Example: The construction of a context-free grammar for regular expression $((a+b)\cdot b)^*$:



$$S_3 \longrightarrow S_1 \mid S_2$$
$$S_2 \longrightarrow b$$
$$S_1 \longrightarrow a$$

A Context-Free Grammar for a Regular Expression

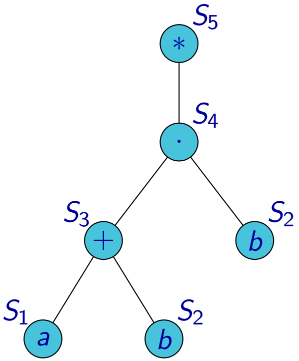
Example: The construction of a context-free grammar for regular expression $((a + b) \cdot b)^*$:



$$
\begin{aligned}
S_4 &\longrightarrow S_3 S_2 \\
S_3 &\longrightarrow S_1 \mid S_2 \\
S_2 &\longrightarrow b \\
S_1 &\longrightarrow a
\end{aligned}
$$

A Context-Free Grammar for a Regular Expression

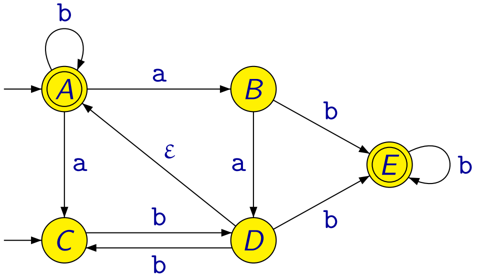
Example: The construction of a context-free grammar for regular expression $((a+b)\cdot b)^*$:



$$
\begin{aligned}
S_5 &\longrightarrow \varepsilon \mid S_4 S_5 \\
S_4 &\longrightarrow S_3 S_2 \\
S_3 &\longrightarrow S_1 \mid S_2 \\
S_2 &\longrightarrow b \\
S_1 &\longrightarrow a
\end{aligned}
$$

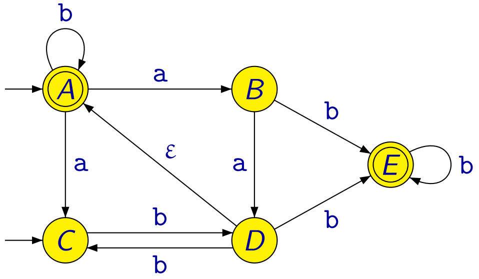
A Context-Free Grammar for a Finite Automaton

Example:



A Context-Free Grammar for a Finite Automaton

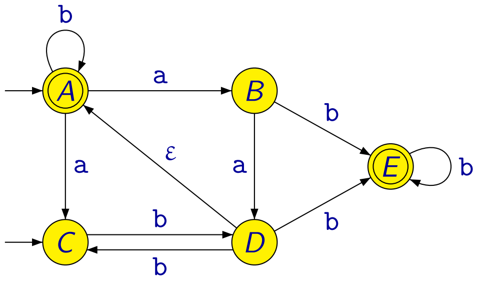
Example:



$S \to A \mid C$

A Context-Free Grammar for a Finite Automaton

Example:

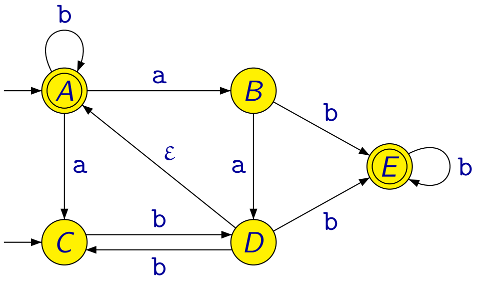


$$S \longrightarrow A \mid C$$

$$\begin{aligned} A &\longrightarrow aB \mid aC \mid bA \\ B &\longrightarrow aD \mid bE \\ C &\longrightarrow bD \\ D &\longrightarrow bC \mid bE \mid A \\ E &\longrightarrow bE \end{aligned}$$

A Context-Free Grammar for a Finite Automaton

Example:



$S \longrightarrow A \mid C$

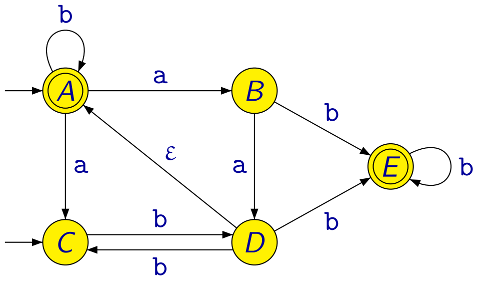
$A \longrightarrow aB \mid aC \mid bA$
$B \longrightarrow aD \mid bE$
$C \longrightarrow bD$
$D \longrightarrow bC \mid bE \mid A$
$E \longrightarrow bE$

$A \longrightarrow \varepsilon$
$E \longrightarrow \varepsilon$

A Context-Free Grammar for a Finite Automaton

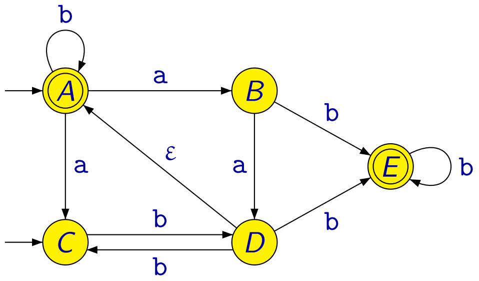
Example:

Alternative construction:



A Context-Free Grammar for a Finite Automaton

Example:

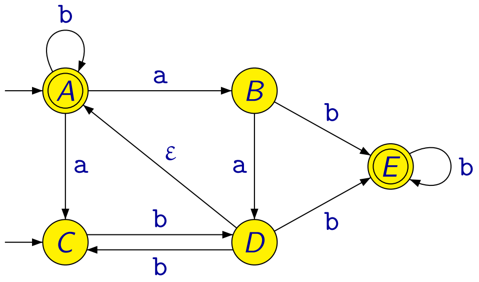


Alternative construction:

$$S \longrightarrow A \mid E$$

A Context-Free Grammar for a Finite Automaton

Example:



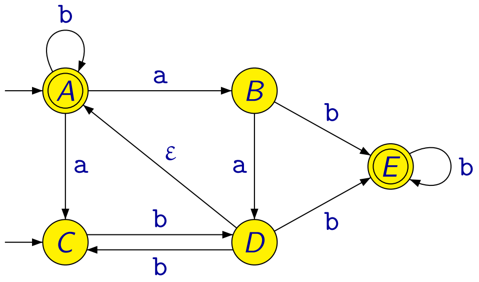
Alternative construction:

$$S \longrightarrow A \mid E$$

$$A \longrightarrow Ab \mid D$$
$$B \longrightarrow Aa$$
$$C \longrightarrow Aa \mid Db$$
$$D \longrightarrow Ba \mid Cb$$
$$E \longrightarrow Bb \mid Db \mid Eb$$

A Context-Free Grammar for a Finite Automaton

Example:



Alternative construction:

$$S \longrightarrow A \mid E$$

$$A \longrightarrow Ab \mid D$$
$$B \longrightarrow Aa$$
$$C \longrightarrow Aa \mid Db$$
$$D \longrightarrow Ba \mid Cb$$
$$E \longrightarrow Bb \mid Db \mid Eb$$

$$A \longrightarrow \varepsilon$$
$$C \longrightarrow \varepsilon$$

Regular grammars

Definition

A grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is **right regular** if all rules in P are of the following forms (where $A, B \in \Pi$, $a \in \Sigma$):

- $A \longrightarrow B$
- $A \longrightarrow aB$
- $A \longrightarrow \varepsilon$

Definition

A grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is **left regular** if all rules in P are of the following forms (kde $A, B \in \Pi$, $a \in \Sigma$):

- $A \longrightarrow B$
- $A \longrightarrow Ba$
- $A \longrightarrow \varepsilon$

Regular grammars

Definition

A grammar $\mathcal{G}$ is **regular** if it right regular or left regular.

Remark: Sometimes a slightly more general definition of right (resp. left) regular grammars is given, allowing all rules of the following forms:

- $A \longrightarrow wB$ (resp. $A \longrightarrow Bw$)
- $A \longrightarrow w$

where $A, B \in \Pi$, $w \in \Sigma^*$.

Such rules can be easily "decomposed" into rules of the form in the previous definition.

Example: Rule $A \longrightarrow abbB$ can be replaced with rules

$$A \longrightarrow aX_1 \qquad X_1 \longrightarrow bX_2 \qquad X_2 \longrightarrow bB$$

where X_1, X_2 are new nonterminals, not used anywhere else in the grammar.

Regular grammars

Proposition

For every regular language L there is a left regular grammar $\mathcal{G}$ such that $\mathcal{L}(\mathcal{G}) = L$ and a right regular grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}') = L$.

Proposition

For every regular grammar $\mathcal{G}$ there is a finite automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G})$.

Pushdown automata

Pushdown automaton

Example: Consider the language over the alphabet $\Sigma = \{(,), [,], <, >\}$ consisting of "correctly parenthesised", i.e., the sequences where every left parenthesis has a corresponding right parenthesis, and where paretheses do not "cross" (as for example in the word `<[>]`).

This language is generated by a context-free grammar

$$A \to \varepsilon \mid (A) \mid [A] \mid <A> \mid AA$$

A typical example of a word that belongs to this language:

`<[](()[<>])>[]`

It is not hard to show that this language is not regular.

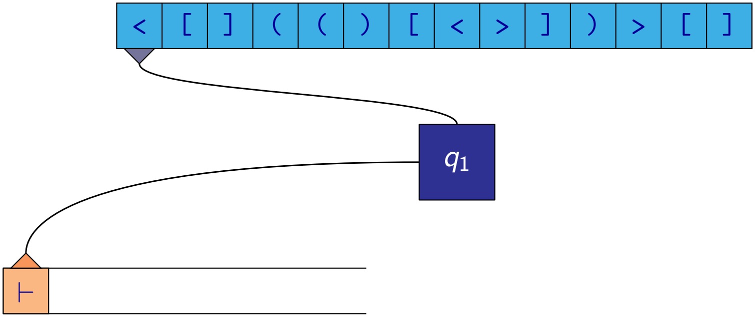
Pushdown automaton

We would like to construct a device, similar to a finite automaton, that would be able to recognize words from this language.

An approppriate possibility seems to be to use a **stack** (of unbounded size) for this recognition.

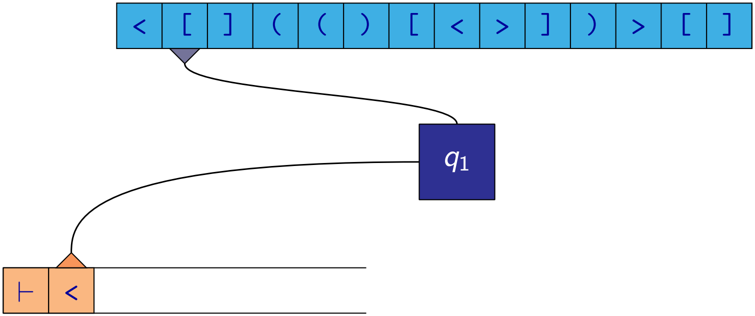
Pushdown automaton

- Word `<[]((()[<>])>[]` belongs to the language.



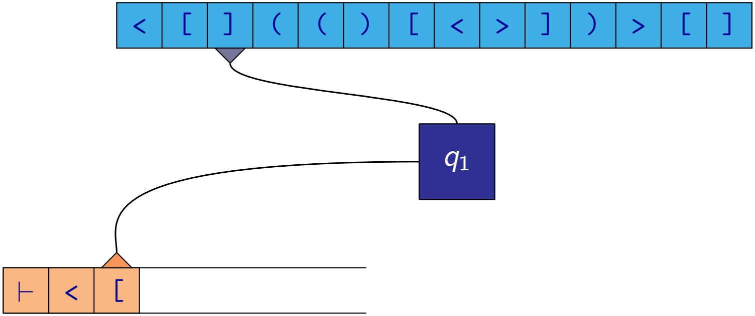
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



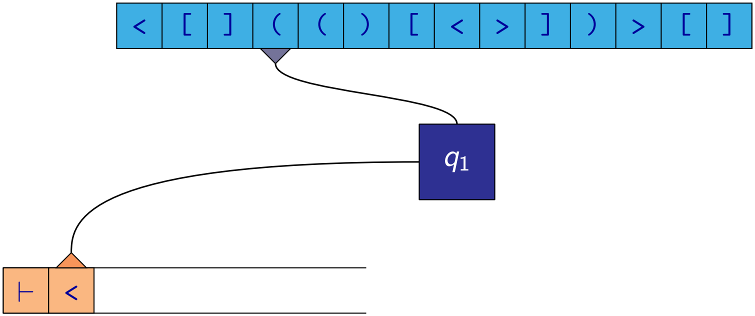
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



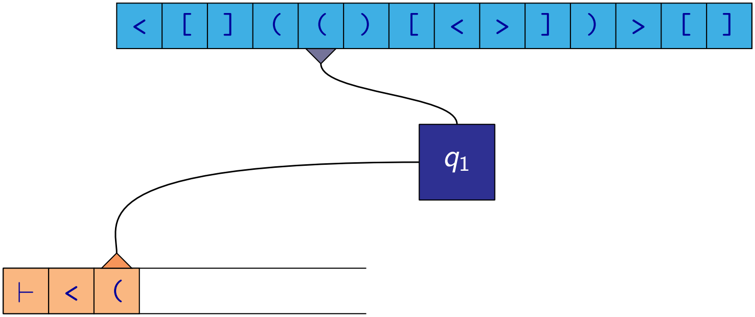
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



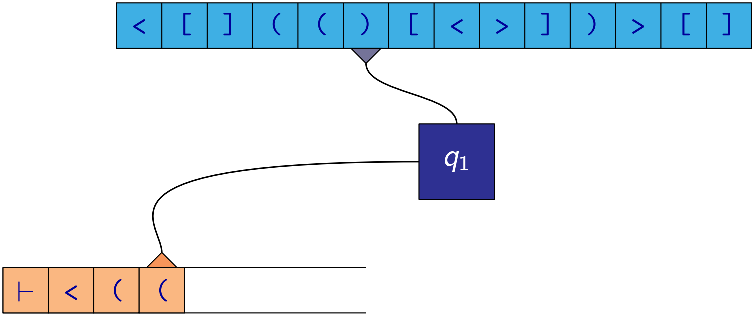
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



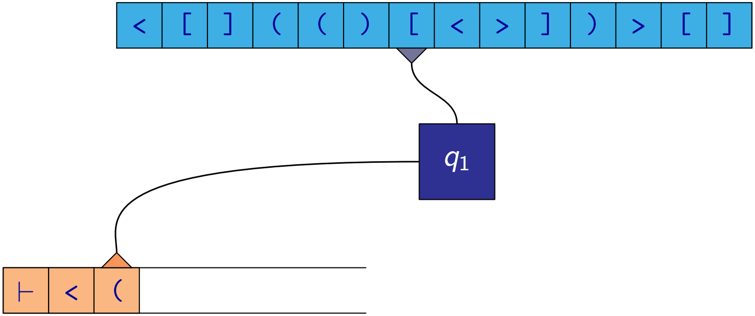
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



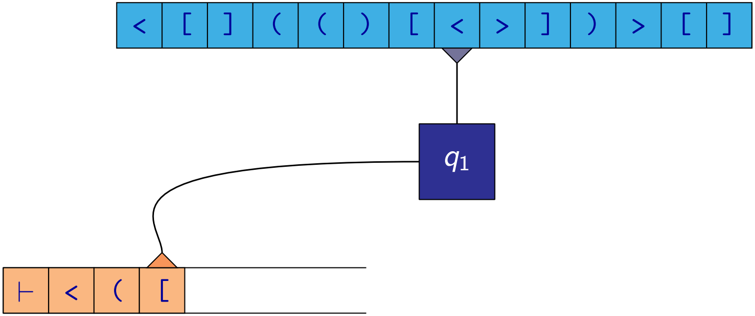
Pushdown automaton

- Word `<[]((）[<>]))>[]` belongs to the language.



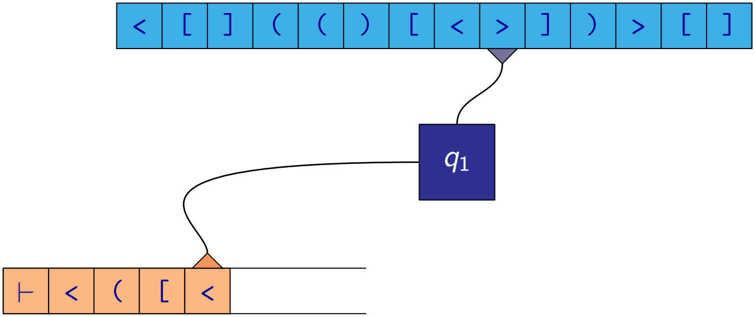
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



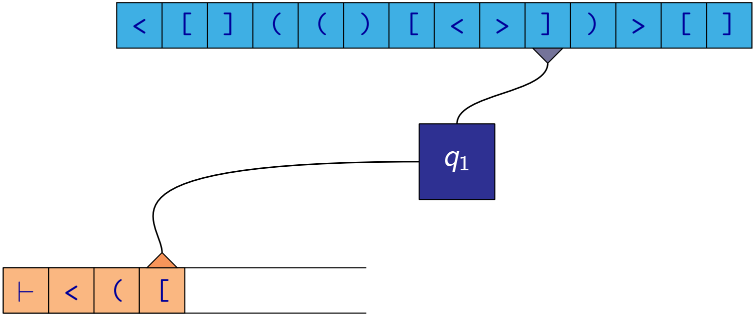
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



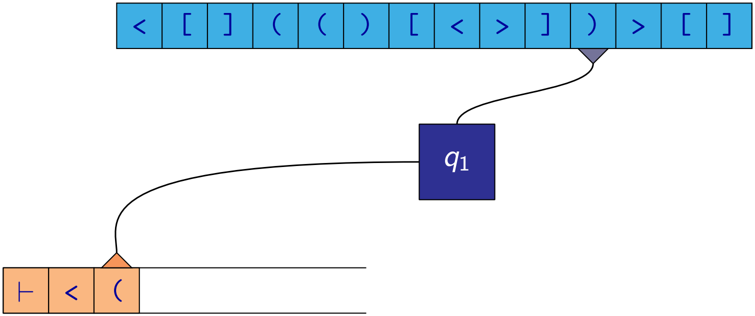
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



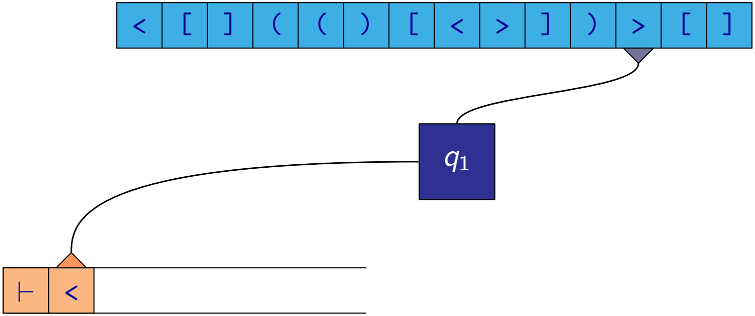
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



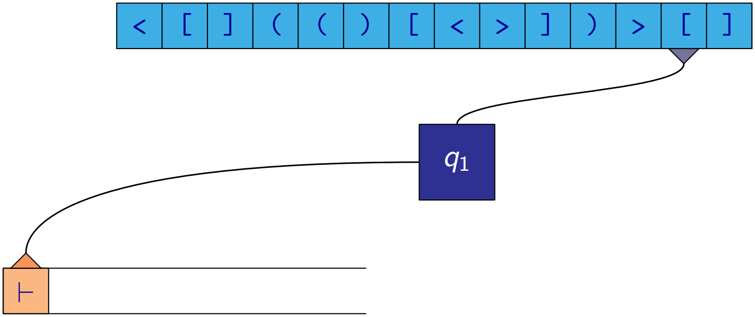
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



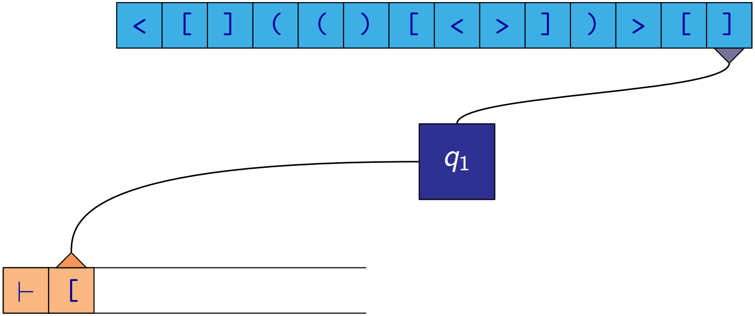
Pushdown automaton

- Word `<[]](()[<>]])>[]` belongs to the language.



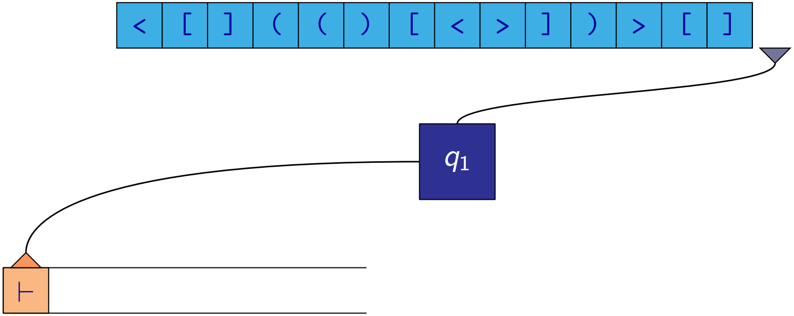
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



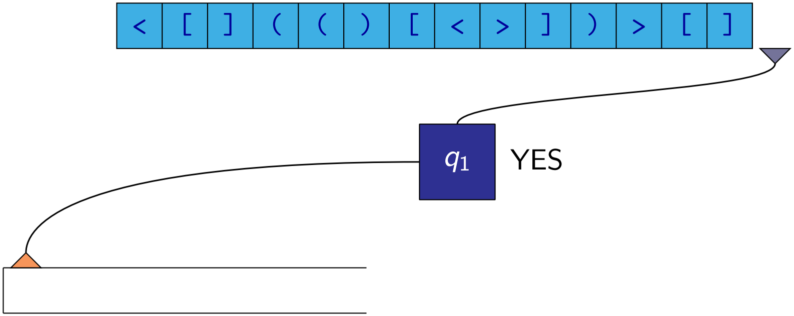
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.



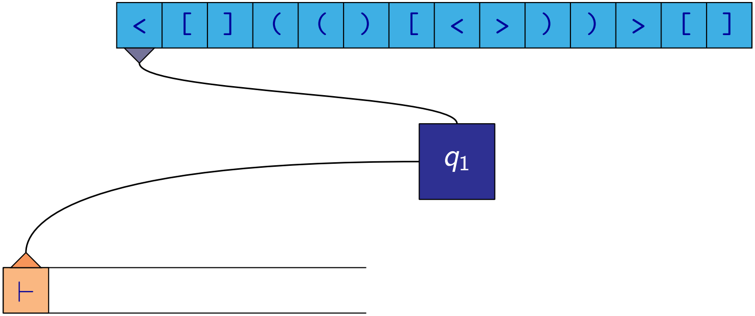
Pushdown automaton

- Word `<[](()[<>])>[]` belongs to the language.
- The automaton has read the whole word and ends with an empty stack, and so the word is accepted by the automaton.



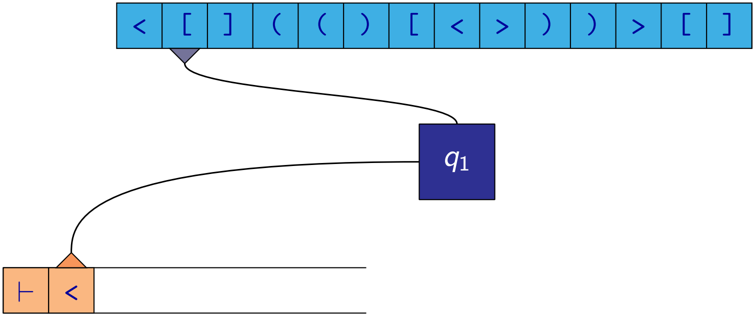
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



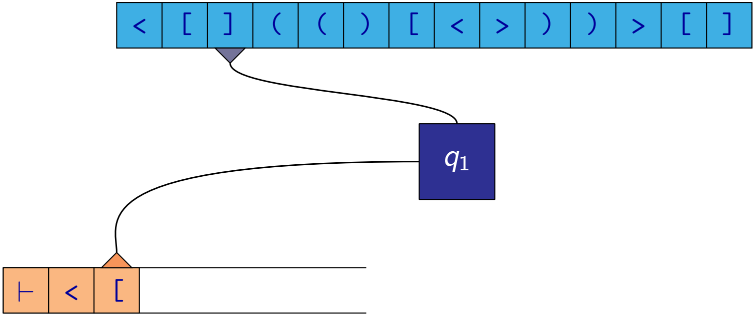
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



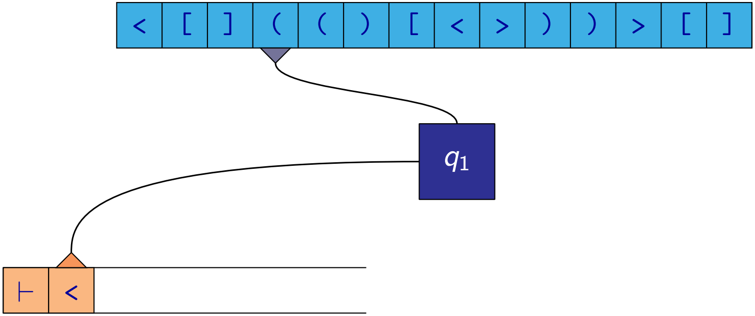
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



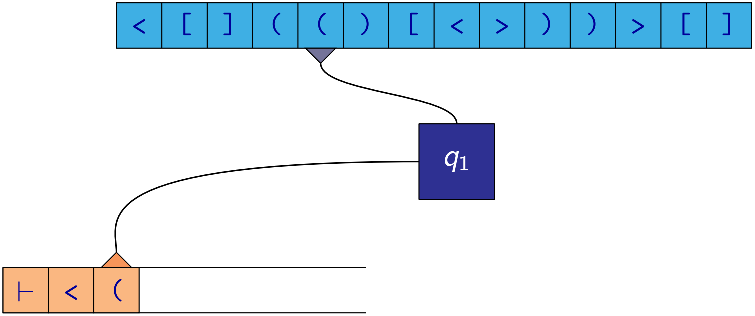
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



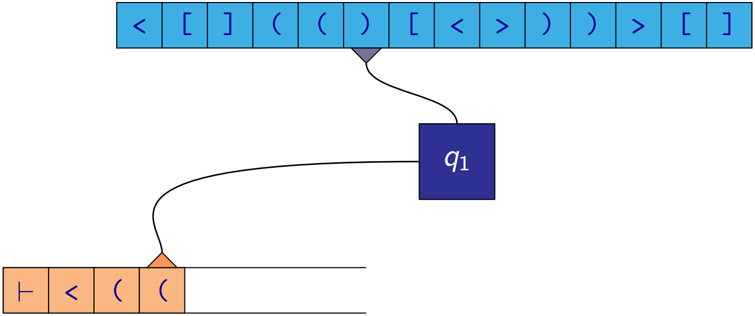
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



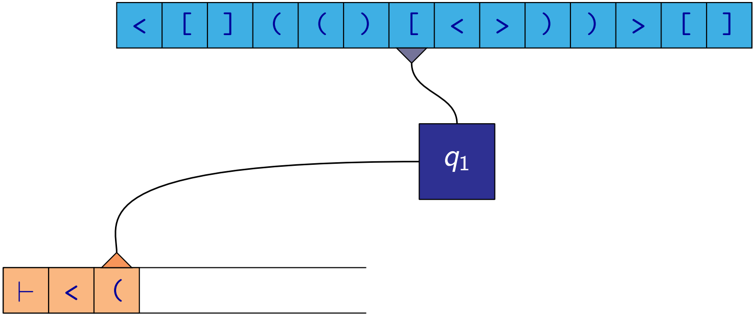
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



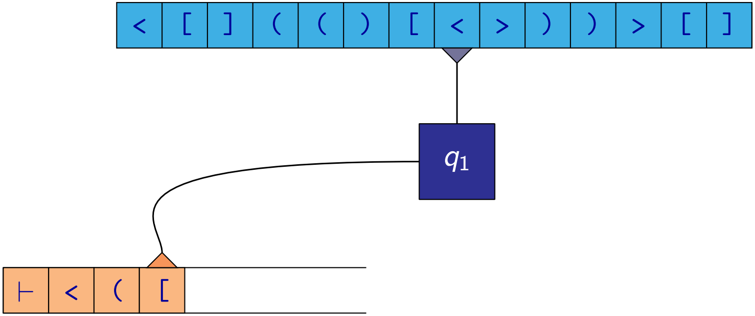
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



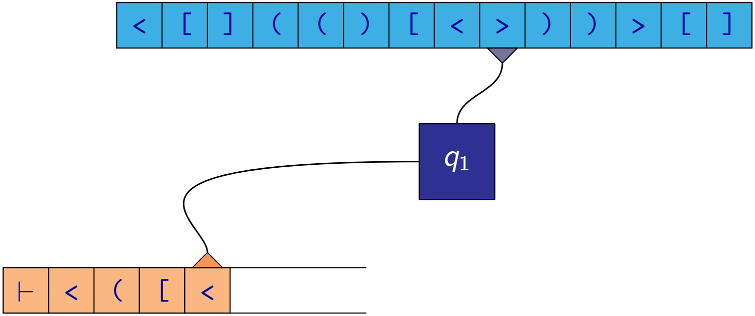
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



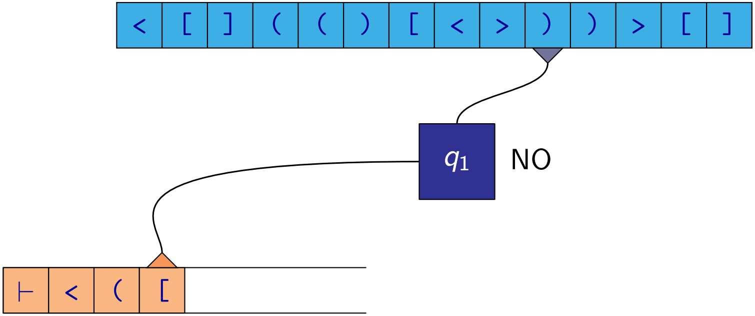
Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.



Pushdown automaton

- Word `<[](()[<>))>[]` does not belong to the language.
- The automaton has found a parenthesis that does not match, so the word is not accepted.



Pushdown automaton

Example:

- We would like to recognize language $L = \{a^n b^n \mid n \geq 1\}$

Again, it is a typical example of a non-regular language.

Pushdown automaton

Example:

- We would like to recognize language $L = \{a^n b^n \mid n \geq 1\}$

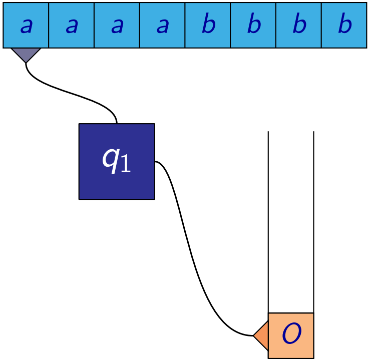
Again, it is a typical example of a non-regular language.

A stack can be used as a counter:

- Symbols of one kind (called for example I) will be pushed to it.
- A number of occurrences of these symbols I on the stack repsents a value of the counter.

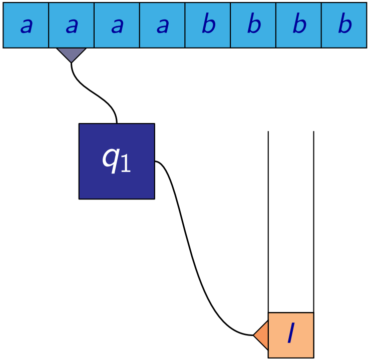
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



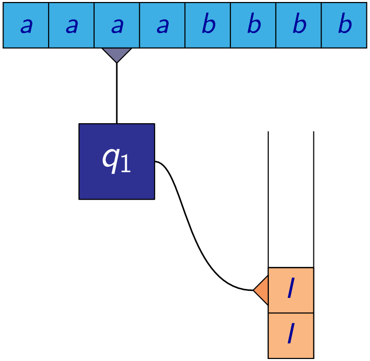
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



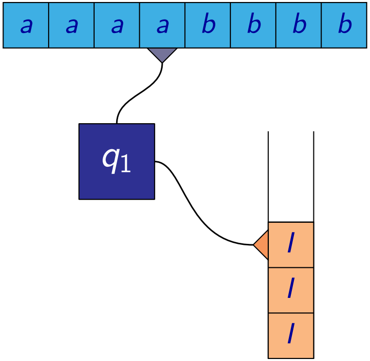
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



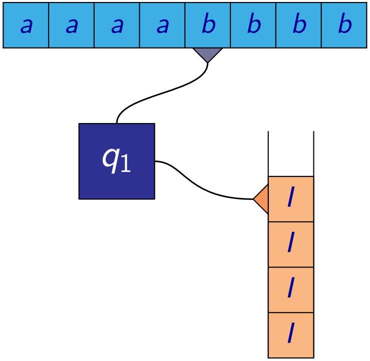
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



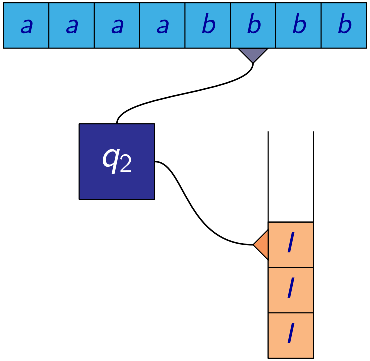
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



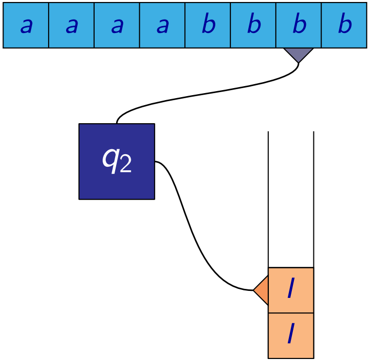
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



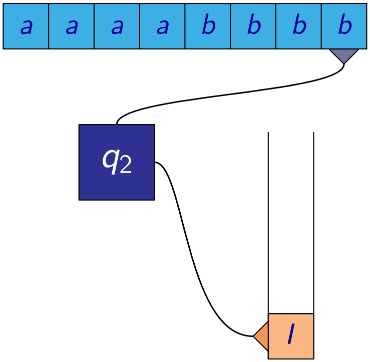
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



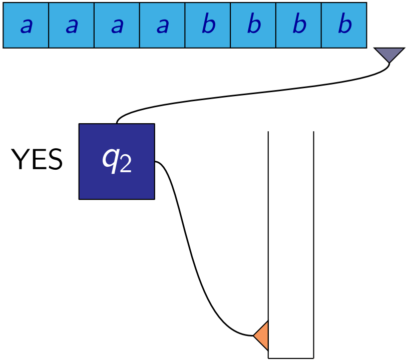
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$



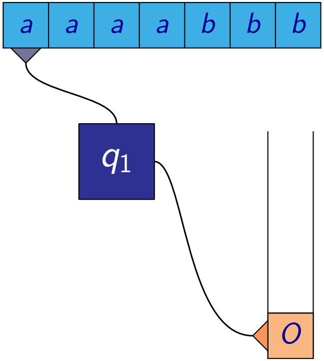
Pushdown automaton

- Word *aaaabbbb* belongs to the language $L = \{a^n b^n \mid n \geq 1\}$
- The automaton has read the whole word and ends with an empty stack, and so the word is accepted by the automaton.



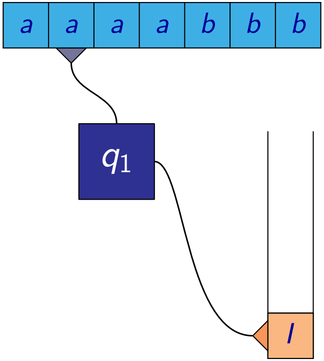
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



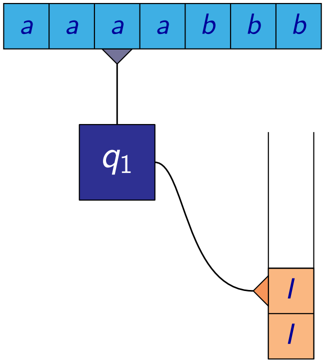
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



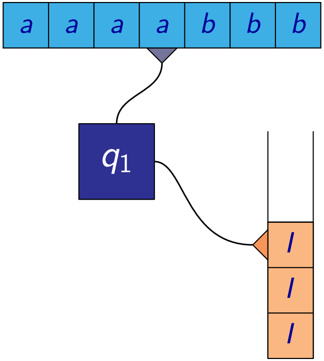
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



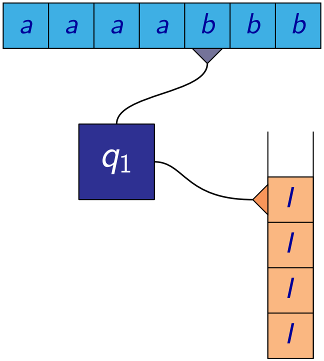
Pushdown automaton

- Word $aaaabbb$ does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



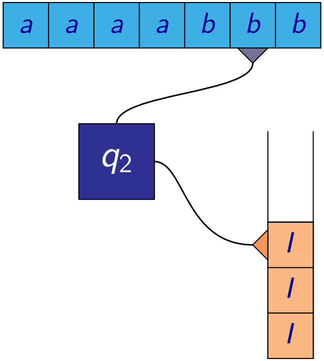
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



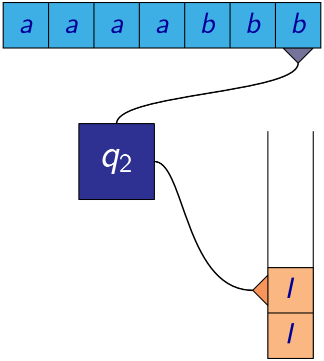
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



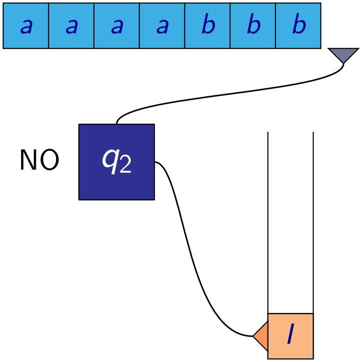
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



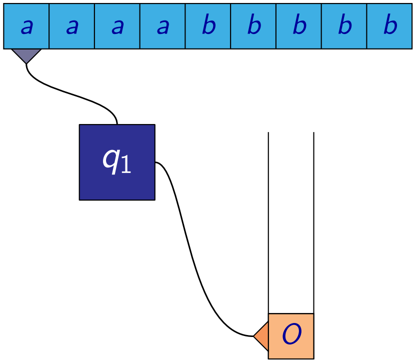
Pushdown automaton

- Word *aaaabbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$
- The automaton has read all word but the stack is not empty and so the word is not accepted by the automaton.



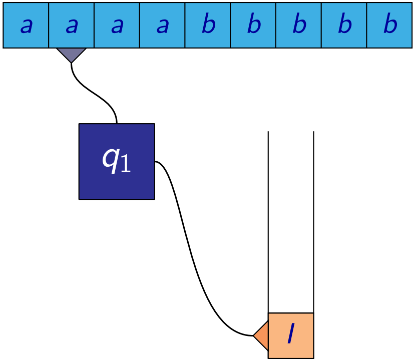
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



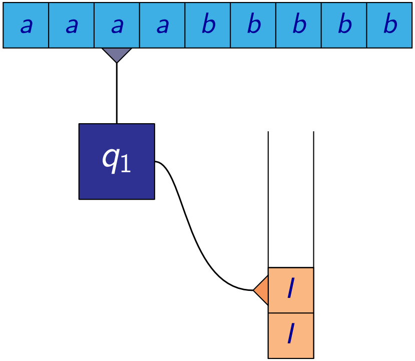
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



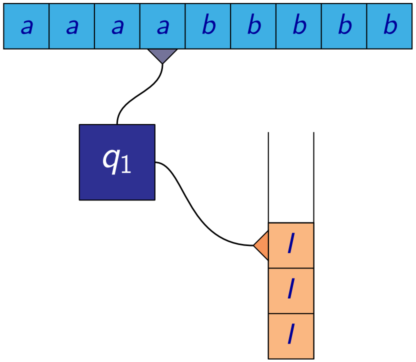
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



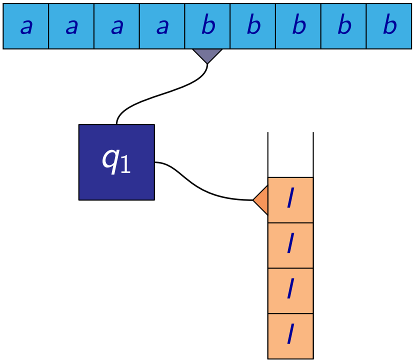
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



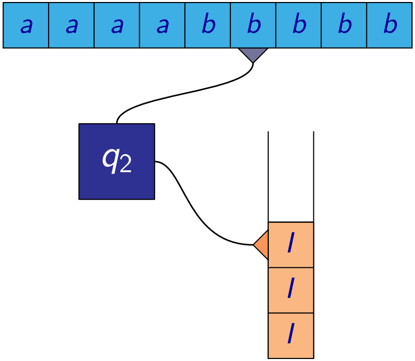
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



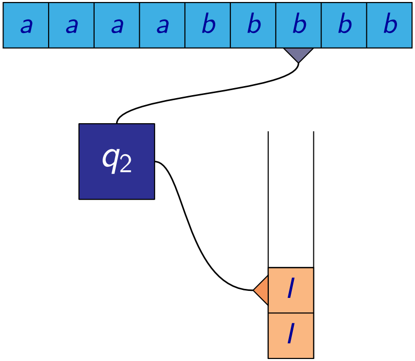
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



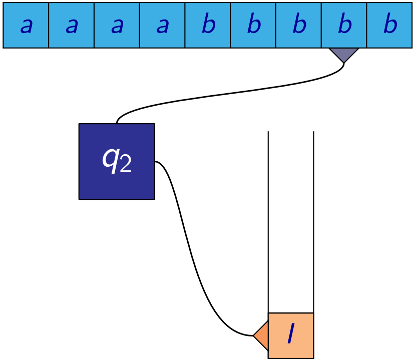
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



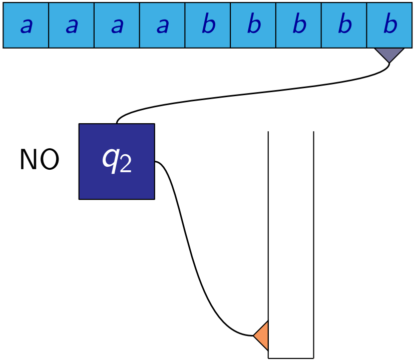
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



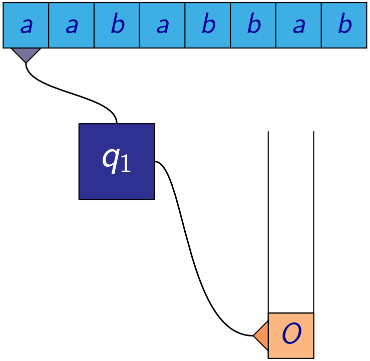
Pushdown automaton

- Word *aaaabbbbb* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$
- The automaton reads b, it should remove a symbol from the stack but there is no symbol there. So the word is not accepted.



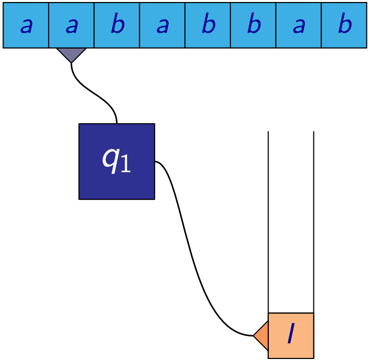
Pushdown automaton

- Word *aababbab* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



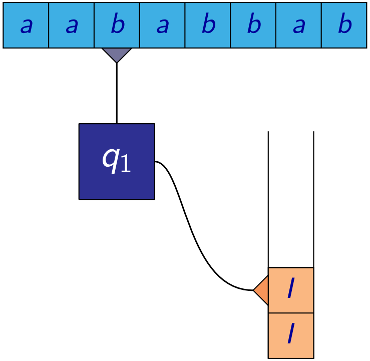
Pushdown automaton

- Word *aababbab* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



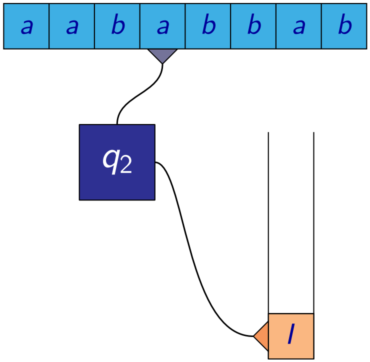
Pushdown automaton

- Word *aababbab* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



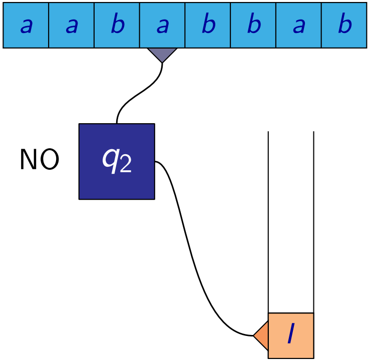
Pushdown automaton

- Word *aababbab* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$



Pushdown automaton

- Word *aababbab* does not belong to language $L = \{a^n b^n \mid n \geq 1\}$
- The automaton has read a but it is already in the state where it removes symbols from the stack, and so the word is not accepted.



Pushdown automaton

- A pushdown automaton can be nondeterministic and it can have ε-transitions.

Pushdown automaton

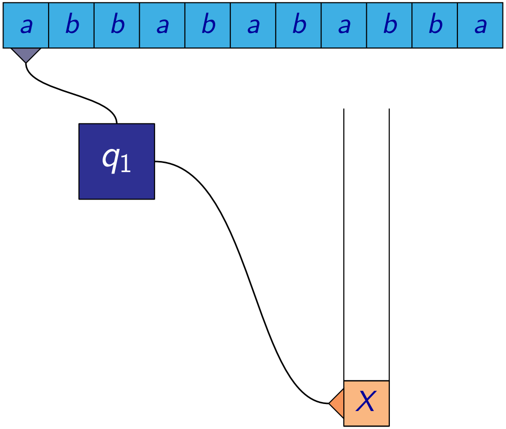
- A pushdown automaton can be nondeterministic and it can have ε-transitions.

Example:

- Let us consider the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$.
- The first half of a word can be stored on the stack.
- When reading the second part, the automaton removes the symbols from the stack if they are same as symbols in the input.
- If the stack is empty after reading all word, the second is the same (the reverse of) the first.
- The automaton can nondeterministically guess the position of the "boundery" between the first and the second half of the word. Those computations where the automaton guesses wrong are nonaccepting.

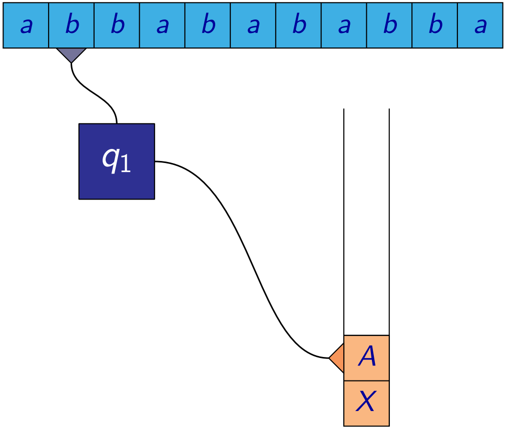
Pushdown automaton

- Word *abbababababba* belongs to the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$



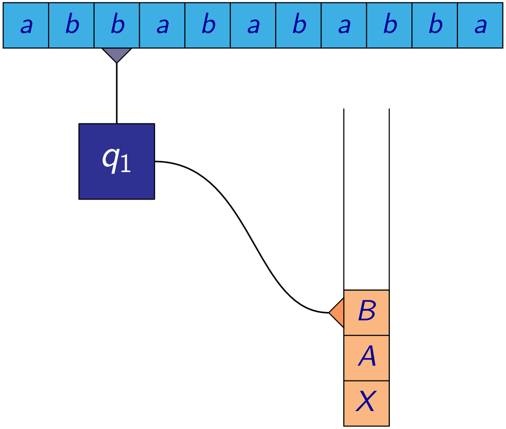
Pushdown automaton

- Word *abbababab ba* belongs to the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$



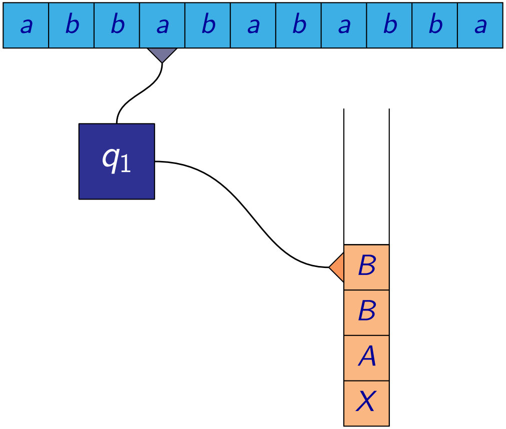
Pushdown automaton

- Word *abbababababba* belongs to the language
  $L = \{w \in \{a,b\}^* \mid w = w^R\}$



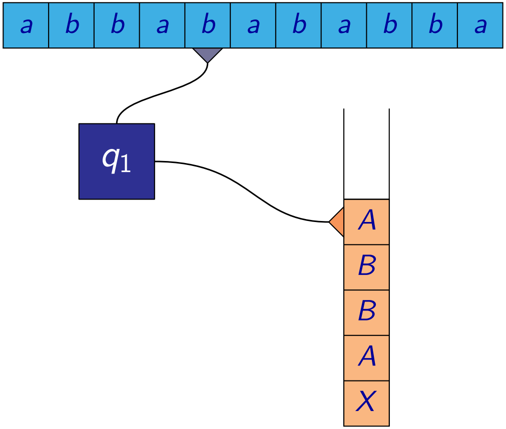
Pushdown automaton

- Word *abbababbabba* belongs to the language $L = \{w \in \{a,b\}^* \mid w = w^R\}$



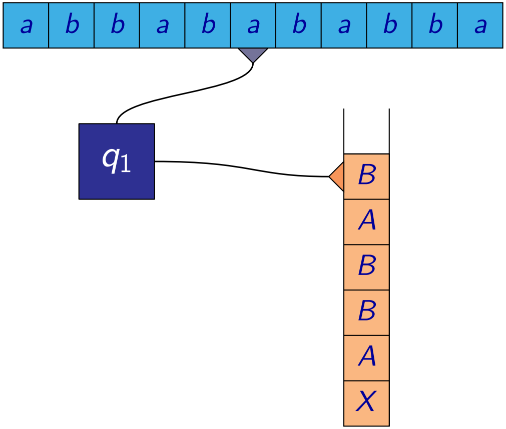
Pushdown automaton

- Word *abbabababba* belongs to the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$



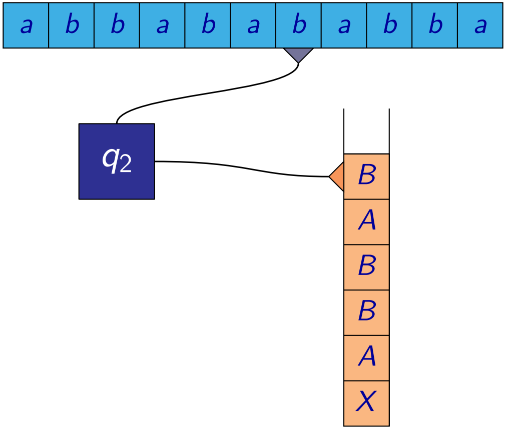
Pushdown automaton

- Word *abbababba* belongs to the language $L = \{w \in \{a,b\}^* \mid w = w^R\}$



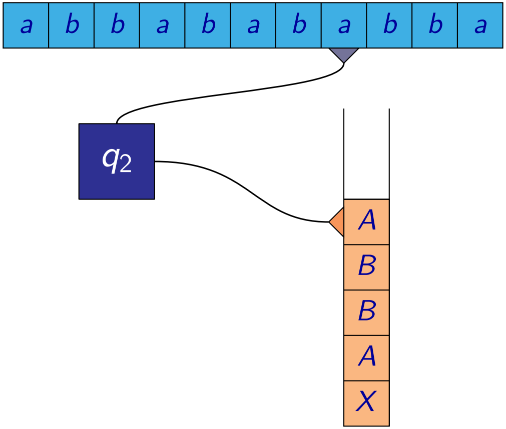
Pushdown automaton

- Word *abbababbabba* belongs to the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$



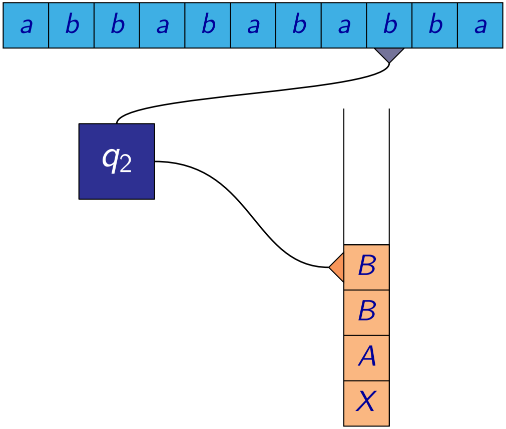
Pushdown automaton

- Word *abbababababba* belongs to the language $L = \{w \in \{a,b\}^* \mid w = w^R\}$



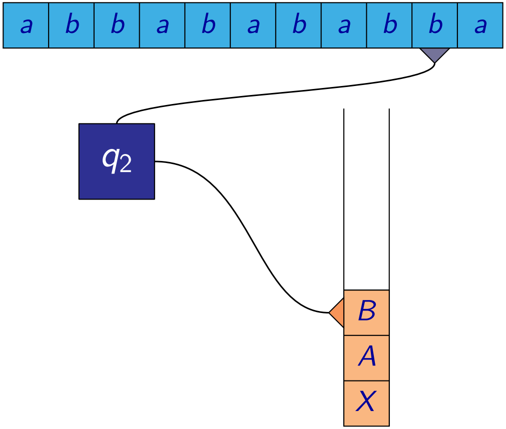
Pushdown automaton

- Word *abbabababba* belongs to the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$



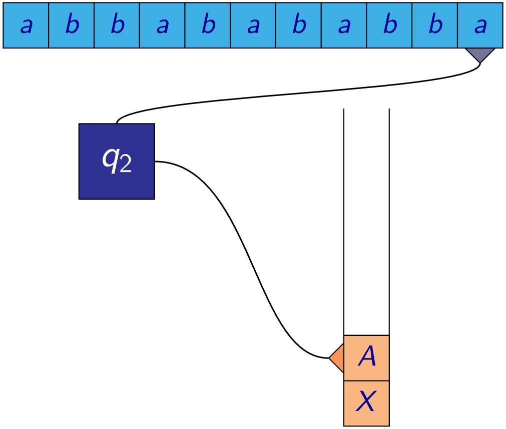
Pushdown automaton

- Word *abbababababba* belongs to the language $L = \{w \in \{a,b\}^* \mid w = w^R\}$



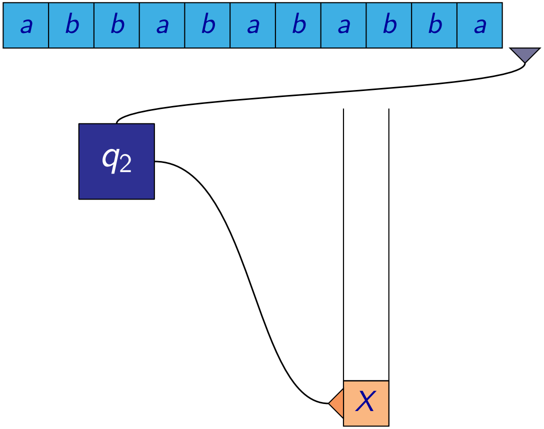
Pushdown automaton

- Word *abbababababba* belongs to the language $L = \{w \in \{a, b\}^* \mid w = w^R\}$



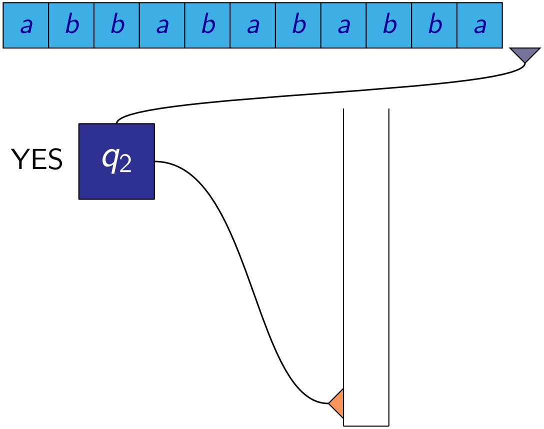
Pushdown automaton

- Word *abbabababba* belongs to the language $L = \{w \in \{a,b\}^* \mid w = w^R\}$



Pushdown automaton

- Word *abbababababba* belongs to the language $L = \{w \in \{a,b\}^* \mid w = w^R\}$



Pushdown automaton

Definition

A **pushdown automaton** (**PDA**) is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, X_0)$ where

- Q is a finite non-empty set of states
- Σ is a finite non-empty set called an input alphabet
- Γ is a finite non-empty set called a stack alphabet
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is a (nondeterministic) transition function
- $q_0 \in Q$ is the initial state
- $X_0 \in \Gamma$ is the initial stack symbol

Pushdown automaton

Example: $L = \{ a^n b^n \mid n \geq 1 \}$

$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, O)$ where

- $Q = \{q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{O, I\}$
- $$\begin{array}{ll} \delta(q_1, a, O) = \{(q_1, I)\} & \delta(q_1, b, O) = \emptyset \\ \delta(q_1, a, I) = \{(q_1, II)\} & \delta(q_1, b, I) = \{(q_2, \varepsilon)\} \\ \delta(q_2, a, I) = \emptyset & \delta(q_2, b, I) = \{(q_2, \varepsilon)\} \\ \delta(q_2, a, O) = \emptyset & \delta(q_2, b, O) = \emptyset \end{array}$$

Remark: We often omit those values of transition function δ that are $\emptyset$.

Pushdown automaton

To represent transition functions, we will use a notation where a transition function is viewed as a set of **rules**:

- For every $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $X \in \Gamma$, and $\alpha \in \Gamma^*$, where
$$(q', \alpha) \in \delta(q, a, X)$$
there is a corresponding rule
$$qX \xrightarrow{a} q'\alpha\,.$$

Example: If

$$\delta(q_5, b, C) = \{(q_3, ACC),\ (q_5, BB),\ (q_{13}, \varepsilon)\}$$

it can be represented as three rules:

$$q_5\,C \xrightarrow{b} q_3ACC \qquad q_5\,C \xrightarrow{b} q_5BB \qquad q_5\,C \xrightarrow{b} q_{13}$$

Pushdown automaton

Example: The automaton, recognizing the language $L = \{ a^n b^n \mid n \geq 1 \}$, that was described before:

$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, O)$ where

- $Q = \{q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{O, I\}$
- $q_1 O \xrightarrow{a} q_1 I$
  $q_1 I \xrightarrow{a} q_1 II$
  $q_1 I \xrightarrow{b} q_2$
  $q_2 I \xrightarrow{b} q_2$

Pushdown automaton

Example: $L = \{ w \in \{a,b\}^* \mid w = w^R \}$

$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where

- $Q = \{q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{X, A, B\}$
- $$\begin{array}{ll}
\delta(q_1, a, X) = \{(q_1, AX), (q_2, X)\} & \delta(q_1, b, X) = \{(q_1, BX), (q_2, X)\} \\
\delta(q_1, a, A) = \{(q_1, AA), (q_2, A)\} & \delta(q_1, b, A) = \{(q_1, BA), (q_2, A)\} \\
\delta(q_1, a, B) = \{(q_1, AB), (q_2, B)\} & \delta(q_1, b, B) = \{(q_1, BB), (q_2, B)\} \\
\delta(q_1, \varepsilon, X) = \{(q_2, X)\} & \delta(q_2, \varepsilon, X) = \{(q_2, \varepsilon)\} \\
\delta(q_1, \varepsilon, A) = \{(q_2, A)\} & \delta(q_2, \varepsilon, A) = \emptyset \\
\delta(q_1, \varepsilon, B) = \{(q_2, B)\} & \delta(q_2, \varepsilon, B) = \emptyset \\
\delta(q_2, a, A) = \{(q_2, \varepsilon)\} & \delta(q_2, b, A) = \emptyset \\
\delta(q_2, a, B) = \emptyset & \delta(q_2, b, B) = \{(q_2, \varepsilon)\} \\
\delta(q_2, a, X) = \emptyset & \delta(q_2, b, X) = \emptyset
\end{array}$$

Pushdown automaton

Example: $L = \{ w \in \{a, b\}^* \mid w = w^R \}$

$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where

- $Q = \{q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{X, A, B\}$
-

$q_1 X \xrightarrow{a} q_1 AX$	$q_1 X \xrightarrow{b} q_1 BX$	$q_2 X \xrightarrow{\varepsilon} q_2$
$q_1 A \xrightarrow{a} q_1 AA$	$q_1 A \xrightarrow{b} q_1 BA$	$q_2 A \xrightarrow{a} q_2$
$q_1 B \xrightarrow{a} q_1 AB$	$q_1 B \xrightarrow{b} q_1 BB$	$q_2 B \xrightarrow{b} q_2$
$q_1 X \xrightarrow{a} q_2 X$	$q_1 X \xrightarrow{b} q_2 X$	$q_1 X \xrightarrow{\varepsilon} q_2 X$
$q_1 A \xrightarrow{a} q_2 A$	$q_1 A \xrightarrow{b} q_2 A$	$q_1 A \xrightarrow{\varepsilon} q_2 A$
$q_1 B \xrightarrow{a} q_2 B$	$q_1 B \xrightarrow{b} q_2 B$	$q_1 B \xrightarrow{\varepsilon} q_2 B$

Computation of a Pushdown Automaton

Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, X_0)$ be a pushdown automaton.

Configurations of $\mathcal{M}$:

- A **configuration** of a PDA is a triple

  $$(q, w, \alpha)$$

  where $q \in Q$, $w \in \Sigma^*$, and $\alpha \in \Gamma^*$.

- An **initial configuration** is a configuration (q_0, w, X_0), where $w \in \Sigma^*$.

Computation of a Pushdown Automaton

Steps performed by $\mathcal{M}$:

- Binary relation $\longrightarrow$ on configurations of $\mathcal{M}$ represents the possible steps of computation performed by PDA $\mathcal{M}$.

  That $\mathcal{M}$ can go from configuration (q, w, α) to configuration (q', w', α') is written as

$$(q, w, \alpha) \longrightarrow (q', w', \alpha') .$$

- The relation $\longrightarrow$ is defined as follows:

$$(q, aw, X\beta) \longrightarrow (q', w, \alpha\beta) \quad \text{iff} \quad (q', \alpha) \in \delta(q, a, X)$$

  where $q, q' \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $w \in \Sigma^*$, $X \in \Gamma$, and $\alpha, \beta \in \Gamma^*$.

Computation of a Pushdown Automaton

Computations of $\mathcal{M}$:

- We define binary relation $\longrightarrow^*$ on configurations of $\mathcal{M}$ as the reflexive and transitive closure of $\longrightarrow$, i.e.,

$$(q, w, \alpha) \longrightarrow^* (q', w', \alpha')$$

if there is a sequence of configurations

$$(q_0, w_0, \alpha_0), (q_1, w_1, \alpha_1), \ldots, (q_n, w_n, \alpha_n)$$

such that

- $(q, w, \alpha) = (q_0, w_0, \alpha_0)$,
- $(q', w', \alpha') = (q_n, w_n, \alpha_n)$, and
- $(q_i, w_i, \alpha_i) \longrightarrow (q_{i+1}, w_{i+1}, \alpha_{i+1})$ for each $i = 0, 1, \ldots, n-1$, i.e.,

$$(q_0, w_0, \alpha_0) \longrightarrow (q_1, w_1, \alpha_1) \longrightarrow \quad \cdots \quad \longrightarrow (q_n, w_n, \alpha_n)$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$q_1X \xrightarrow{a} q_1AX \qquad q_1X \xrightarrow{b} q_1BX$$
$$q_1A \xrightarrow{a} q_1AA \qquad q_1A \xrightarrow{b} q_1BA$$
$$q_1B \xrightarrow{a} q_1AB \qquad q_1B \xrightarrow{b} q_1BB$$
$$q_1X \xrightarrow{a} q_2X \qquad q_1X \xrightarrow{b} q_2X$$
$$q_1A \xrightarrow{a} q_2A \qquad q_1A \xrightarrow{b} q_2A$$
$$q_1B \xrightarrow{a} q_2B \qquad q_1B \xrightarrow{b} q_2B$$
$$q_1X \xrightarrow{\varepsilon} q_2X$$
$$q_1A \xrightarrow{\varepsilon} q_2A$$
$$q_1B \xrightarrow{\varepsilon} q_2B$$
$$q_2X \xrightarrow{\varepsilon} q_2$$
$$q_2A \xrightarrow{a} q_2$$
$$q_2B \xrightarrow{b} q_2$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$(q_1,\ abbababababba,\ X)$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$(q_1,\ abbababababba,\ X)$
$\longrightarrow\ (q_1,\ bbababababba,\ AX)$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$(q_1,\ abbababababba,\ X)$
$\longrightarrow\ (q_1,\ bbababababba,\ AX)$
$\longrightarrow\ (q_1,\ bababababba,\ BAX)$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$(q_1,\ abbababababba,\ X)$
$\longrightarrow\ (q_1,\ bbababababba,\ AX)$
$\longrightarrow\ (q_1,\ bababababba,\ BAX)$
$\longrightarrow\ (q_1,\ abababbba,\ BBAX)$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{array}{l}
(q_1,\ abbabababba,\ X) \\
\longrightarrow\ (q_1,\ bbabababba,\ AX) \\
\longrightarrow\ (q_1,\ babababba,\ BAX) \\
\longrightarrow\ (q_1,\ abababba,\ BBAX) \\
\longrightarrow\ (q_1,\ bababba,\ ABBAX)
\end{array}
$$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{aligned}
& (q_1,\ abbababababba,\ X) \\
\longrightarrow\ & (q_1,\ bbababababba,\ AX) \\
\longrightarrow\ & (q_1,\ babababba,\ BAX) \\
\longrightarrow\ & (q_1,\ abababba,\ BBAX) \\
\longrightarrow\ & (q_1,\ bababba,\ ABBAX) \\
\longrightarrow\ & (q_1,\ ababba,\ BABBAX)
\end{aligned}
$$

$$
\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}
$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
& (q_1,\ abbabababba,\ X) \\
\longrightarrow\ & (q_1,\ bbabababba,\ AX) \\
\longrightarrow\ & (q_1,\ babababba,\ BAX) \\
\longrightarrow\ & (q_1,\ abababba,\ BBAX) \\
\longrightarrow\ & (q_1,\ bababba,\ ABBAX) \\
\longrightarrow\ & (q_1,\ ababba,\ BABBAX) \\
\longrightarrow\ & (q_2,\ babba,\ BABBAX)
\end{aligned}$$

$$\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
&(q_1,\ abbababababba,\ X) \\
&\longrightarrow\ (q_1,\ bbababababba,\ AX) \\
&\longrightarrow\ (q_1,\ bababababba,\ BAX) \\
&\longrightarrow\ (q_1,\ abababba,\ BBAX) \\
&\longrightarrow\ (q_1,\ bababba,\ ABBAX) \\
&\longrightarrow\ (q_1,\ ababba,\ BABBAX) \\
&\longrightarrow\ (q_2,\ babba,\ BABBAX) \\
&\longrightarrow\ (q_2,\ abba,\ ABBAX)
\end{aligned}$$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$(q_1,\ abbabababba,\ X)$
$\longrightarrow\ (q_1,\ bbabababba,\ AX)$
$\longrightarrow\ (q_1,\ babababba,\ BAX)$
$\longrightarrow\ (q_1,\ ababababba,\ BBAX)$
$\longrightarrow\ (q_1,\ bababba,\ ABBAX)$
$\longrightarrow\ (q_1,\ ababba,\ BABBAX)$
$\longrightarrow\ (q_2,\ babba,\ BABBAX)$
$\longrightarrow\ (q_2,\ abba,\ ABBAX)$
$\longrightarrow\ (q_2,\ bba,\ BBAX)$

$q_1X \xrightarrow{a} q_1AX$ $\qquad$ $q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$ $\qquad$ $q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$ $\qquad$ $q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$ $\qquad$ $q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$ $\qquad$ $q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$ $\qquad$ $q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$
$q_1A \xrightarrow{\varepsilon} q_2A$
$q_1B \xrightarrow{\varepsilon} q_2B$
$q_2X \xrightarrow{\varepsilon} q_2$
$q_2A \xrightarrow{a} q_2$
$q_2B \xrightarrow{b} q_2$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{array}{l}
(q_1,\ abbabababba,\ X) \\
\longrightarrow\ (q_1,\ bbabababba,\ AX) \\
\longrightarrow\ (q_1,\ babababba,\ BAX) \\
\longrightarrow\ (q_1,\ abababba,\ BBAX) \\
\longrightarrow\ (q_1,\ bababba,\ ABBAX) \\
\longrightarrow\ (q_1,\ ababba,\ BABBAX) \\
\longrightarrow\ (q_2,\ babba,\ BABBAX) \\
\longrightarrow\ (q_2,\ abba,\ ABBAX) \\
\longrightarrow\ (q_2,\ bba,\ BBAX) \\
\longrightarrow\ (q_2,\ ba,\ BAX)
\end{array}
$$

$$
\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}
$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{aligned}
& (q_1,\ abbabababba,\ X) \\
\longrightarrow\ & (q_1,\ bbabababba,\ AX) \\
\longrightarrow\ & (q_1,\ babababba,\ BAX) \\
\longrightarrow\ & (q_1,\ ababababba,\ BBAX) \\
\longrightarrow\ & (q_1,\ bababba,\ ABBAX) \\
\longrightarrow\ & (q_1,\ ababba,\ BABBAX) \\
\longrightarrow\ & (q_2,\ babba,\ BABBAX) \\
\longrightarrow\ & (q_2,\ abba,\ ABBAX) \\
\longrightarrow\ & (q_2,\ bba,\ BBAX) \\
\longrightarrow\ & (q_2,\ ba,\ BAX) \\
\longrightarrow\ & (q_2,\ a,\ AX)
\end{aligned}
$$

$$
\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}
$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$(q_1,\ abbabababba,\ X)$
$\longrightarrow\ (q_1,\ bbabababba,\ AX)$
$\longrightarrow\ (q_1,\ babababba,\ BAX)$
$\longrightarrow\ (q_1,\ ababababba,\ BBAX)$
$\longrightarrow\ (q_1,\ bababba,\ ABBAX)$
$\longrightarrow\ (q_1,\ ababba,\ BABBAX)$
$\longrightarrow\ (q_2,\ babba,\ BABBAX)$
$\longrightarrow\ (q_2,\ abba,\ ABBAX)$
$\longrightarrow\ (q_2,\ bba,\ BBAX)$
$\longrightarrow\ (q_2,\ ba,\ BAX)$
$\longrightarrow\ (q_2,\ a,\ AX)$
$\longrightarrow\ (q_2,\ \varepsilon,\ X)$

$q_1 X \xrightarrow{a} q_1 AX$	$q_1 X \xrightarrow{b} q_1 BX$
$q_1 A \xrightarrow{a} q_1 AA$	$q_1 A \xrightarrow{b} q_1 BA$
$q_1 B \xrightarrow{a} q_1 AB$	$q_1 B \xrightarrow{b} q_1 BB$
$q_1 X \xrightarrow{a} q_2 X$	$q_1 X \xrightarrow{b} q_2 X$
$q_1 A \xrightarrow{a} q_2 A$	$q_1 A \xrightarrow{b} q_2 A$
$q_1 B \xrightarrow{a} q_2 B$	$q_1 B \xrightarrow{b} q_2 B$
$q_1 X \xrightarrow{\varepsilon} q_2 X$	
$q_1 A \xrightarrow{\varepsilon} q_2 A$	
$q_1 B \xrightarrow{\varepsilon} q_2 B$	
$q_2 X \xrightarrow{\varepsilon} q_2$	
$q_2 A \xrightarrow{a} q_2$	
$q_2 B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{aligned}
& (q_1,\ abbabababba,\ X) \\
\longrightarrow\ & (q_1,\ bbabababba,\ AX) \\
\longrightarrow\ & (q_1,\ babababba,\ BAX) \\
\longrightarrow\ & (q_1,\ abababba,\ BBAX) \\
\longrightarrow\ & (q_1,\ bababba,\ ABBAX) \\
\longrightarrow\ & (q_1,\ ababba,\ BABBAX) \\
\longrightarrow\ & (q_2,\ babba,\ BABBAX) \\
\longrightarrow\ & (q_2,\ abba,\ ABBAX) \\
\longrightarrow\ & (q_2,\ bba,\ BBAX) \\
\longrightarrow\ & (q_2,\ ba,\ BAX) \\
\longrightarrow\ & (q_2,\ a,\ AX) \\
\longrightarrow\ & (q_2,\ \varepsilon,\ X) \\
\longrightarrow\ & (q_2,\ \varepsilon,\ \varepsilon)
\end{aligned}
$$

$$
\begin{array}{ll}
q_1 X \xrightarrow{a} q_1 AX & q_1 X \xrightarrow{b} q_1 BX \\
q_1 A \xrightarrow{a} q_1 AA & q_1 A \xrightarrow{b} q_1 BA \\
q_1 B \xrightarrow{a} q_1 AB & q_1 B \xrightarrow{b} q_1 BB \\
q_1 X \xrightarrow{a} q_2 X & q_1 X \xrightarrow{b} q_2 X \\
q_1 A \xrightarrow{a} q_2 A & q_1 A \xrightarrow{b} q_2 A \\
q_1 B \xrightarrow{a} q_2 B & q_1 B \xrightarrow{b} q_2 B \\
q_1 X \xrightarrow{\varepsilon} q_2 X & \\
q_1 A \xrightarrow{\varepsilon} q_2 A & \\
q_1 B \xrightarrow{\varepsilon} q_2 B & \\
q_2 X \xrightarrow{\varepsilon} q_2 & \\
q_2 A \xrightarrow{a} q_2 & \\
q_2 B \xrightarrow{b} q_2 &
\end{array}
$$

Computation of a Pushdown Automaton

In the previous definition, the set of configurations was defined as

$$Conf = Q \times \Sigma^* \times \Gamma^*$$

and relation $\longrightarrow$ was a subset of the set $Conf \times Conf$.

Computation of a Pushdown Automaton

Alternatively, we could define configurations in such a way that they do not contain an input word:

$$Conf \;=\; Q \times \Gamma^*$$

The relation $\longrightarrow$ is then defined as a subset of the set $Conf \times (\Sigma \cup \{\varepsilon\}) \times Conf$, where the notation

$$q\alpha \xrightarrow{a} q'\alpha'$$

that after reading symbol a (or reading nothing when $a = \varepsilon$), the given pushdown automaton can go from configuration (q, α) to configuration (q', α'), i.e.,

$$qX\beta \xrightarrow{a} q'\gamma\beta \quad \text{iff} \quad (q', \gamma) \in \delta(q, a, X)$$

where $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $X \in \Gamma$, and $\beta, \gamma \in \Gamma^*$.

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$q_1X \xrightarrow{a} q_1AX \qquad q_1X \xrightarrow{b} q_1BX$$
$$q_1A \xrightarrow{a} q_1AA \qquad q_1A \xrightarrow{b} q_1BA$$
$$q_1B \xrightarrow{a} q_1AB \qquad q_1B \xrightarrow{b} q_1BB$$
$$q_1X \xrightarrow{a} q_2X \qquad q_1X \xrightarrow{b} q_2X$$
$$q_1A \xrightarrow{a} q_2A \qquad q_1A \xrightarrow{b} q_2A$$
$$q_1B \xrightarrow{a} q_2B \qquad q_1B \xrightarrow{b} q_2B$$
$$q_1X \xrightarrow{\varepsilon} q_2X$$
$$q_1A \xrightarrow{\varepsilon} q_2A$$
$$q_1B \xrightarrow{\varepsilon} q_2B$$
$$q_2X \xrightarrow{\varepsilon} q_2$$
$$q_2A \xrightarrow{a} q_2$$
$$q_2B \xrightarrow{b} q_2$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

q_1X

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$q_1 X \xrightarrow{a} q_1 AX$$

$q_1 X \xrightarrow{a} q_1 AX$	$q_1 X \xrightarrow{b} q_1 BX$
$q_1 A \xrightarrow{a} q_1 AA$	$q_1 A \xrightarrow{b} q_1 BA$
$q_1 B \xrightarrow{a} q_1 AB$	$q_1 B \xrightarrow{b} q_1 BB$
$q_1 X \xrightarrow{a} q_2 X$	$q_1 X \xrightarrow{b} q_2 X$
$q_1 A \xrightarrow{a} q_2 A$	$q_1 A \xrightarrow{b} q_2 A$
$q_1 B \xrightarrow{a} q_2 B$	$q_1 B \xrightarrow{b} q_2 B$
$q_1 X \xrightarrow{\varepsilon} q_2 X$	
$q_1 A \xrightarrow{\varepsilon} q_2 A$	
$q_1 B \xrightarrow{\varepsilon} q_2 B$	
$q_2 X \xrightarrow{\varepsilon} q_2$	
$q_2 A \xrightarrow{a} q_2$	
$q_2 B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$q_1X \xrightarrow{a} q_1AX \xrightarrow{b} q_1BAX$$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX
\end{aligned}
$$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned} q_1X &\xrightarrow{a} q_1AX \\ &\xrightarrow{b} q_1BAX \\ &\xrightarrow{b} q_1BBAX \\ &\xrightarrow{a} q_1ABBAX \end{aligned}$$

$q_1X \xrightarrow{a} q_1AX$	$q_1X \xrightarrow{b} q_1BX$
$q_1A \xrightarrow{a} q_1AA$	$q_1A \xrightarrow{b} q_1BA$
$q_1B \xrightarrow{a} q_1AB$	$q_1B \xrightarrow{b} q_1BB$
$q_1X \xrightarrow{a} q_2X$	$q_1X \xrightarrow{b} q_2X$
$q_1A \xrightarrow{a} q_2A$	$q_1A \xrightarrow{b} q_2A$
$q_1B \xrightarrow{a} q_2B$	$q_1B \xrightarrow{b} q_2B$
$q_1X \xrightarrow{\varepsilon} q_2X$	
$q_1A \xrightarrow{\varepsilon} q_2A$	
$q_1B \xrightarrow{\varepsilon} q_2B$	
$q_2X \xrightarrow{\varepsilon} q_2$	
$q_2A \xrightarrow{a} q_2$	
$q_2B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
q_1 X &\xrightarrow{a} q_1 AX \\
&\xrightarrow{b} q_1 BAX \\
&\xrightarrow{b} q_1 BBAX \\
&\xrightarrow{a} q_1 ABBAX \\
&\xrightarrow{b} q_1 BABBAX
\end{aligned}$$

$$\begin{array}{ll}
q_1 X \xrightarrow{a} q_1 AX & q_1 X \xrightarrow{b} q_1 BX \\
q_1 A \xrightarrow{a} q_1 AA & q_1 A \xrightarrow{b} q_1 BA \\
q_1 B \xrightarrow{a} q_1 AB & q_1 B \xrightarrow{b} q_1 BB \\
q_1 X \xrightarrow{a} q_2 X & q_1 X \xrightarrow{b} q_2 X \\
q_1 A \xrightarrow{a} q_2 A & q_1 A \xrightarrow{b} q_2 A \\
q_1 B \xrightarrow{a} q_2 B & q_1 B \xrightarrow{b} q_2 B \\
q_1 X \xrightarrow{\varepsilon} q_2 X & \\
q_1 A \xrightarrow{\varepsilon} q_2 A & \\
q_1 B \xrightarrow{\varepsilon} q_2 B & \\
q_2 X \xrightarrow{\varepsilon} q_2 & \\
q_2 A \xrightarrow{a} q_2 & \\
q_2 B \xrightarrow{b} q_2 &
\end{array}$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX \\
&\xrightarrow{a} q_1ABBAX \\
&\xrightarrow{b} q_1BABBAX \\
&\xrightarrow{a} q_2BABBAX
\end{aligned}
$$

$$
\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}
$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$
\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX \\
&\xrightarrow{a} q_1ABBAX \\
&\xrightarrow{b} q_1BABBAX \\
&\xrightarrow{a} q_2BABBAX \\
&\xrightarrow{b} q_2ABBAX
\end{aligned}
$$

$$
\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}
$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX \\
&\xrightarrow{a} q_1ABBAX \\
&\xrightarrow{b} q_1BABBAX \\
&\xrightarrow{a} q_2BABBAX \\
&\xrightarrow{b} q_2ABBAX \\
&\xrightarrow{a} q_2BBAX
\end{aligned}$$

$$\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
q_1 X &\xrightarrow{a} q_1 AX \\
&\xrightarrow{b} q_1 BAX \\
&\xrightarrow{b} q_1 BBAX \\
&\xrightarrow{a} q_1 ABBAX \\
&\xrightarrow{b} q_1 BABBAX \\
&\xrightarrow{a} q_2 BABBAX \\
&\xrightarrow{b} q_2 ABBAX \\
&\xrightarrow{a} q_2 BBAX \\
&\xrightarrow{b} q_2 BAX
\end{aligned}$$

$q_1 X \xrightarrow{a} q_1 AX$	$q_1 X \xrightarrow{b} q_1 BX$
$q_1 A \xrightarrow{a} q_1 AA$	$q_1 A \xrightarrow{b} q_1 BA$
$q_1 B \xrightarrow{a} q_1 AB$	$q_1 B \xrightarrow{b} q_1 BB$
$q_1 X \xrightarrow{a} q_2 X$	$q_1 X \xrightarrow{b} q_2 X$
$q_1 A \xrightarrow{a} q_2 A$	$q_1 A \xrightarrow{b} q_2 A$
$q_1 B \xrightarrow{a} q_2 B$	$q_1 B \xrightarrow{b} q_2 B$
$q_1 X \xrightarrow{\varepsilon} q_2 X$	
$q_1 A \xrightarrow{\varepsilon} q_2 A$	
$q_1 B \xrightarrow{\varepsilon} q_2 B$	
$q_2 X \xrightarrow{\varepsilon} q_2$	
$q_2 A \xrightarrow{a} q_2$	
$q_2 B \xrightarrow{b} q_2$	

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX \\
&\xrightarrow{a} q_1ABBAX \\
&\xrightarrow{b} q_1BABBAX \\
&\xrightarrow{a} q_2BABBAX \\
&\xrightarrow{b} q_2ABBAX \\
&\xrightarrow{a} q_2BBAX \\
&\xrightarrow{b} q_2BAX \\
&\xrightarrow{b} q_2AX
\end{aligned}$$

$$\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX \\
&\xrightarrow{a} q_1ABBAX \\
&\xrightarrow{b} q_1BABBAX \\
&\xrightarrow{a} q_2BABBAX \\
&\xrightarrow{b} q_2ABBAX \\
&\xrightarrow{a} q_2BBAX \\
&\xrightarrow{b} q_2BAX \\
&\xrightarrow{b} q_2AX \\
&\xrightarrow{a} q_2X
\end{aligned}$$

$$\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
q_1A &\xrightarrow{a} q_1AA \\
q_1B &\xrightarrow{a} q_1AB \\
q_1X &\xrightarrow{a} q_2X \\
q_1A &\xrightarrow{a} q_2A \\
q_1B &\xrightarrow{a} q_2B \\
q_1X &\xrightarrow{\varepsilon} q_2X \\
q_1A &\xrightarrow{\varepsilon} q_2A \\
q_1B &\xrightarrow{\varepsilon} q_2B \\
q_2X &\xrightarrow{\varepsilon} q_2 \\
q_2A &\xrightarrow{a} q_2 \\
q_2B &\xrightarrow{b} q_2
\end{aligned}$$

$$\begin{aligned}
q_1X &\xrightarrow{b} q_1BX \\
q_1A &\xrightarrow{b} q_1BA \\
q_1B &\xrightarrow{b} q_1BB \\
q_1X &\xrightarrow{b} q_2X \\
q_1A &\xrightarrow{b} q_2A \\
q_1B &\xrightarrow{b} q_2B
\end{aligned}$$

Computation of a Pushdown Automaton

Example: $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, X)$ where $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{X, A, B\}$

$$\begin{aligned}
q_1X &\xrightarrow{a} q_1AX \\
&\xrightarrow{b} q_1BAX \\
&\xrightarrow{b} q_1BBAX \\
&\xrightarrow{a} q_1ABBAX \\
&\xrightarrow{b} q_1BABBAX \\
&\xrightarrow{a} q_2BABBAX \\
&\xrightarrow{b} q_2ABBAX \\
&\xrightarrow{a} q_2BBAX \\
&\xrightarrow{b} q_2BAX \\
&\xrightarrow{b} q_2AX \\
&\xrightarrow{a} q_2X \\
&\xrightarrow{\varepsilon} q_2
\end{aligned}$$

$$\begin{array}{ll}
q_1X \xrightarrow{a} q_1AX & q_1X \xrightarrow{b} q_1BX \\
q_1A \xrightarrow{a} q_1AA & q_1A \xrightarrow{b} q_1BA \\
q_1B \xrightarrow{a} q_1AB & q_1B \xrightarrow{b} q_1BB \\
q_1X \xrightarrow{a} q_2X & q_1X \xrightarrow{b} q_2X \\
q_1A \xrightarrow{a} q_2A & q_1A \xrightarrow{b} q_2A \\
q_1B \xrightarrow{a} q_2B & q_1B \xrightarrow{b} q_2B \\
q_1X \xrightarrow{\varepsilon} q_2X & \\
q_1A \xrightarrow{\varepsilon} q_2A & \\
q_1B \xrightarrow{\varepsilon} q_2B & \\
q_2X \xrightarrow{\varepsilon} q_2 & \\
q_2A \xrightarrow{a} q_2 & \\
q_2B \xrightarrow{b} q_2 &
\end{array}$$

Pushdown automaton

Two different definitions acceptace of words are used:

- A pushdown automaton $\mathcal{M}$ accepting by an **empty stack** accepts a word w iff there is some computation of $\mathcal{M}$ on w such that $\mathcal{M}$ reads all symbols of w and after reading them, the stack is empty.

- A pushdown automaton $\mathcal{M}$ accepting by an **accepting state** accepts a word w iff there is some computation of $\mathcal{M}$ on w such that $\mathcal{M}$ reads all symbols of w and after reading them, the control unit of $\mathcal{M}$ is in some state from a given set of accepting states F.

Pushdown automaton

- A word $w \in \Sigma^*$ is **accepted** by PDA $\mathcal{M}$ **by empty stack** iff

$$(q_0, w, X_0) \longrightarrow^* (q, \varepsilon, \varepsilon)$$

for some $q \in Q$.

Definition

The **langugage** $\mathcal{L}(\mathcal{M})$ **accepted** by PDA $\mathcal{M}$ **by empty stack** is defined as

$$\mathcal{L}(\mathcal{M}) = \{ w \in \Sigma^* \mid (\exists q \in Q)((q_0, w, X_0) \longrightarrow^* (q, \varepsilon, \varepsilon)) \}.$$

Pushdown automaton

Let us extend the definition of PDA $\mathcal{M}$ with a set of **accepting states** F (where $F \subseteq Q$).

- A word $w \in \Sigma^*$ is **accepted** by PDA $\mathcal{M}$ **by accepting state** iff

$$(q_0, w, X_0) \longrightarrow^* (q, \varepsilon, \alpha)$$

for some $q \in F$ and $\alpha \in \Gamma^*$.

Definition

The **langugage** $\mathcal{L}(\mathcal{M})$ **accepted** by PDA $\mathcal{M}$ **by accepting state** is defined as

$$\mathcal{L}(\mathcal{M}) = \{\, w \in \Sigma^* \mid (\exists q \in F)(\exists \alpha \in \Gamma^*)((q_0, w, X_0) \longrightarrow^* (q, \varepsilon, \alpha)) \,\}.$$

Pushdown automata

In the case of **nondeterministic** pushdown automata, there is no difference in the class of accepted languages between recognizing by empty stack and recognizing by accepting state.

We can easily perform the following constructions:

- To construct for a given (nondeterministic) pushdown automaton, that recognizes a language L by empty stack, an equivalent (nondeterministic) pushdown automaton recognizing this language L by accepting states.
- To construct for a given (nondeterministic) pushdown automaton, that recognizes a language L by accepting states, an equivalent (nondeterministic) pushdown automaton recognizing the language L by empty stack.

Deterministic Pushdown Automata

A pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, X_0)$ is **deterministic** when:

- For each $q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$ and $X \in \Gamma$ it holds that:
$$|\delta(q, a, X)| \leq 1$$
- For each $q \in Q$ and $X \in \Gamma$ holds at most one of the following possibilities:
  - There exists a rule $qX \xrightarrow{\varepsilon} q'\alpha$ for some $q' \in Q$ and $\alpha \in \Gamma^*$.
  - There exists a rule $qX \xrightarrow{a} q'\alpha$ for some $a \in \Sigma$, $q' \in Q$ and $\alpha \in \Gamma^*$.

Deterministic Pushdown Automata

Note that **deterministic** pushdown automata accepting by empty stack are able to recognize only **prefix-free** languages, i.e., languages L where:

- if $w \in L$, then there is no word $w' \in L$ such that w is a proper prefix of w'.

Remark: Instead of language $L \subseteq \Sigma^*$, that possibly is or is not prefix-free, we can take the prefix-free language

$$L' = L \cdot \{\dashv\}$$

over the alphabet $\Sigma \cup \{\dashv\}$, where $\dashv \notin \Sigma$ is a special "marker" representing the end of a word.
I.e., instead of testing whether $w \in L$, where $w \in \Sigma^*$, we can test whether $(w \dashv) \in L'$.

Deterministic Pushdown Automata

- For each deterministic pushdown automaton recognizing by empty stack we can easily construct an equivalent deterministic pushdown automaton recognizing by accepting states.

- For each deterministic pushdown automaton recognizing language L (where $L \subseteq \Sigma^*$) by accepting states we can easily construct a deterministic pushdown automaton recognizing by empty stack the language $L \cdot \{\dashv\}$, where $\dashv \notin \Sigma$.

Equivalence of CFG and PDA

Theorem

For every context-free grammar $\mathcal{G}$ we can construct a pushdown automaton $\mathcal{M}$ (with one control state) such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{G})$.

Proof: For CFG $\mathcal{G} = (\Pi, \Sigma, S, P)$ we construct PDA $\mathcal{M} = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, S)$, where

- $\Gamma = \Pi \cup \Sigma$
- For each rule $(X \rightarrow \alpha) \in P$ from the context-free grammar $\mathcal{G}$ (where $X \in \Pi$ a $\alpha \in (\Pi \cup \Sigma)^*$), we add a corresponing rule

  $$q_0 X \xrightarrow{\varepsilon} q_0 \alpha$$

  to the trasition function δ of the pushdown automaton $\mathcal{M}$.
- For each symbol $a \in \Sigma$, we add a rule

  $$q_0 a \xrightarrow{a} q_0$$

  to the trasition function δ of the pushdown automaton $\mathcal{M}$.

Equivalence of CFG and PDA

Example: Consider a context-free grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$, where

- $\Pi = \{S, E, T, F\}$
- $\Sigma = \{\, \mathtt{a}, \mathtt{+}, \mathtt{*}, \mathtt{(}, \mathtt{)}, \dashv \}$
- The set P contains the following rules:

$$\begin{aligned} S &\longrightarrow E\dashv \\ E &\longrightarrow T \mid E\mathtt{+}T \\ T &\longrightarrow F \mid T\mathtt{*}F \\ F &\longrightarrow \mathtt{a} \mid \mathtt{(}E\mathtt{)} \end{aligned}$$

Equivalence of CFG and PDA

For the given grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ with rules

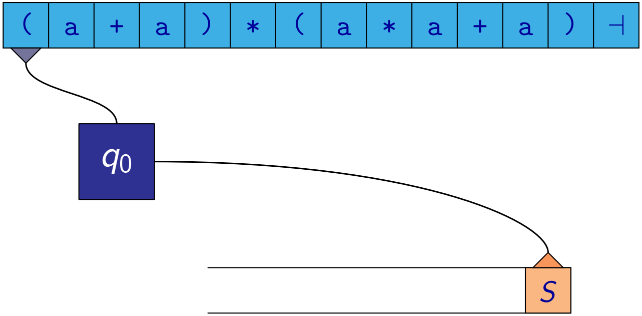
$$\begin{aligned} S &\longrightarrow E \dashv \\ E &\longrightarrow T \mid E\texttt{+}T \\ T &\longrightarrow F \mid T\texttt{*}F \\ F &\longrightarrow \texttt{a} \mid \texttt{(}E\texttt{)} \end{aligned}$$

we construct a pushdown automaton $\mathcal{M} = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, S)$, where

- $\Sigma = \{\texttt{a}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}, \dashv\}$
- $\Gamma = \{S, E, T, F, \texttt{a}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}, \dashv\}$
- The trasition function δ contains the following rules:

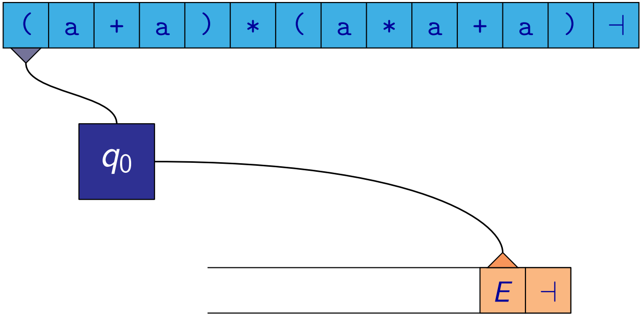
$$\begin{array}{llll}
q_0 S \xrightarrow{\varepsilon} q_0 E \dashv & q_0 F \xrightarrow{\varepsilon} q_0 \texttt{a} & q_0 \texttt{a} \xrightarrow{\texttt{a}} q_0 & q_0 \texttt{(} \xrightarrow{\texttt{(}} q_0 \\
q_0 E \xrightarrow{\varepsilon} q_0 T & q_0 F \xrightarrow{\varepsilon} q_0 \texttt{(}E\texttt{)} & q_0 \texttt{+} \xrightarrow{\texttt{+}} q_0 & q_0 \texttt{)} \xrightarrow{\texttt{)}} q_0 \\
q_0 E \xrightarrow{\varepsilon} q_0 E\texttt{+}T & & q_0 \texttt{*} \xrightarrow{\texttt{*}} q_0 & q_0 \dashv \xrightarrow{\dashv} q_0 \\
q_0 T \xrightarrow{\varepsilon} q_0 F & & & \\
q_0 T \xrightarrow{\varepsilon} q_0 T\texttt{*}F & & &
\end{array}$$

Equivalence of CFG and PDA



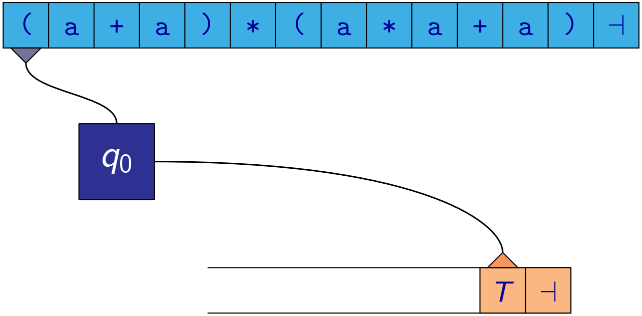
$\underline{S}$

Equivalence of CFG and PDA



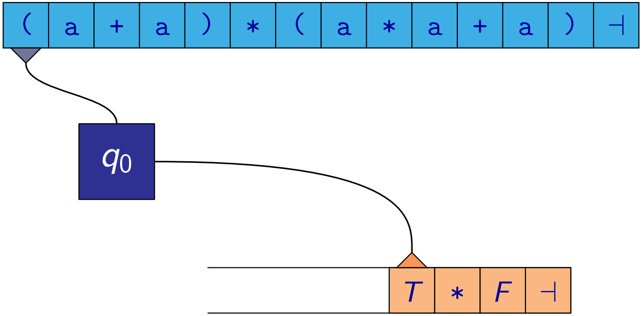
$\underline{S} \Rightarrow \underline{E}\dashv$

Equivalence of CFG and PDA



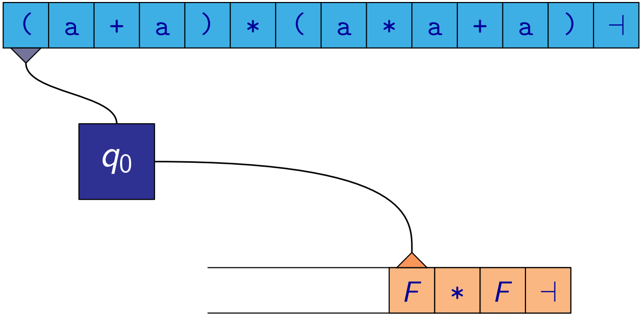
$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv$

Equivalence of CFG and PDA



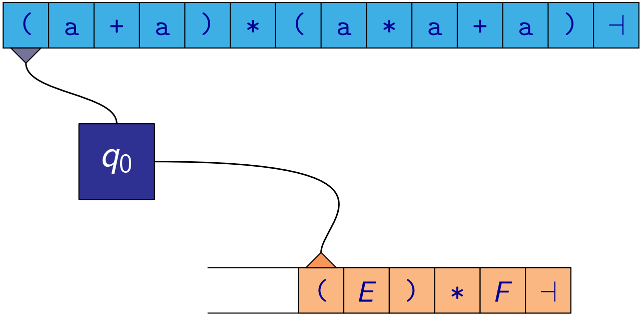
$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow \underline{T}*F\dashv$

Equivalence of CFG and PDA



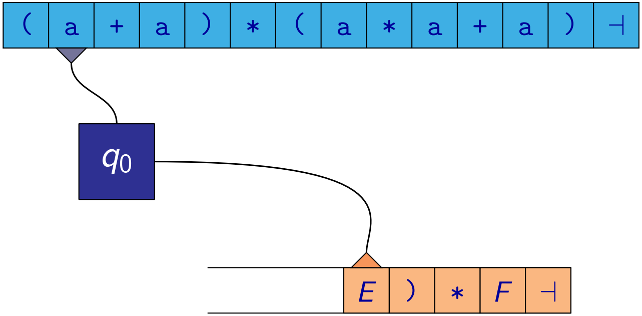
$$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow \underline{T}*F\dashv \Rightarrow \underline{F}*F\dashv$$

Equivalence of CFG and PDA



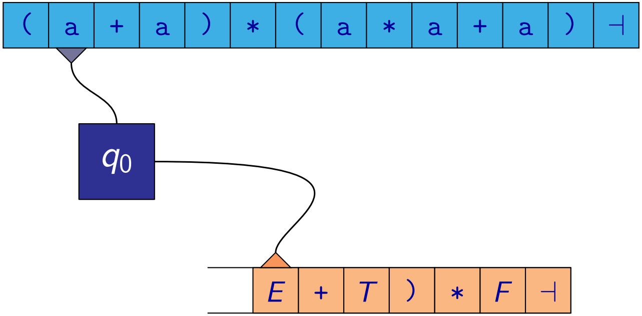
$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow \underline{T}*F\dashv \Rightarrow \underline{F}*F\dashv \Rightarrow (\underline{E})*F\dashv$

Equivalence of CFG and PDA



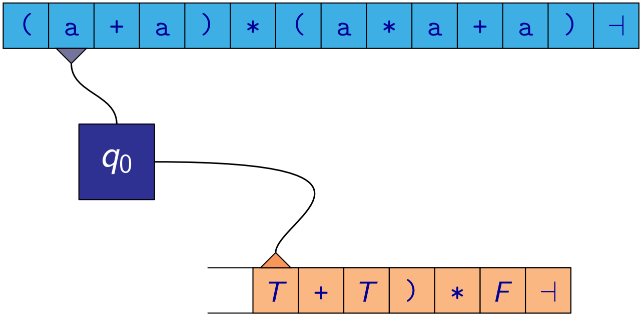
$$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow \underline{T}*F\dashv \Rightarrow \underline{F}*F\dashv \Rightarrow (\underline{E})*F\dashv$$

Equivalence of CFG and PDA



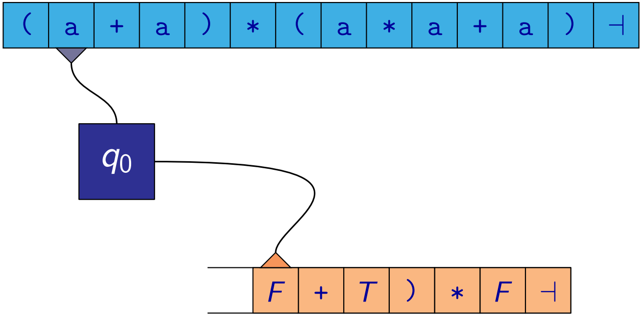
$$\cdots \Rightarrow \underline{T}\dashv \Rightarrow \underline{T}*F\dashv \Rightarrow \underline{F}*F\dashv \Rightarrow (\underline{E})*F\dashv \Rightarrow (\underline{E}+T)*F\dashv$$

Equivalence of CFG and PDA



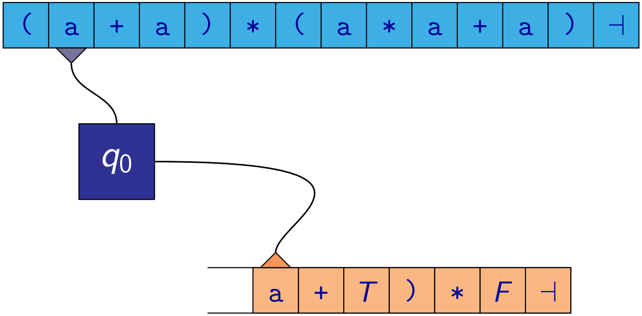
$$\cdots \Rightarrow \underline{F}*F\dashv \Rightarrow (\underline{E})*F\dashv \Rightarrow (\underline{E}+T)*F\dashv \Rightarrow (\underline{T}+T)*F\dashv$$

Equivalence of CFG and PDA



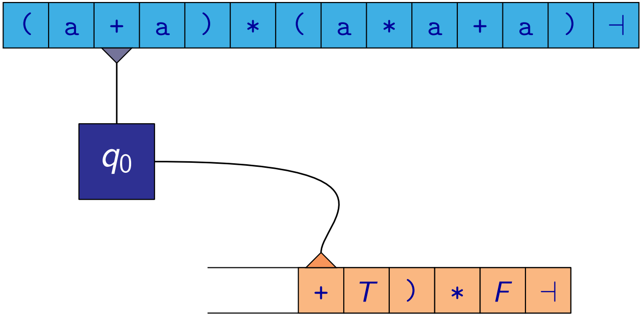
$$\cdots \Rightarrow (\underline{E})*F\dashv \Rightarrow (\underline{E}+T)*F\dashv \Rightarrow (\underline{T}+T)*F\dashv \Rightarrow (\underline{F}+T)*F\dashv$$

Equivalence of CFG and PDA



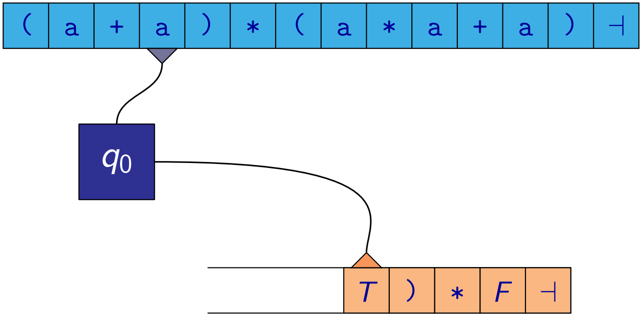
$$\cdots \Rightarrow (\underline{E}+T)*F\dashv \Rightarrow (\underline{T}+T)*F\dashv \Rightarrow (\underline{F}+T)*F\dashv \Rightarrow (\mathtt{a}+\underline{T})*F\dashv$$

Equivalence of CFG and PDA



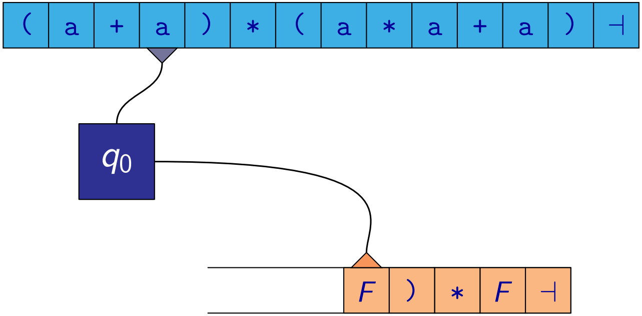
$$\cdots \Rightarrow (\underline{E}+T)*F\dashv \Rightarrow (\underline{T}+T)*F\dashv \Rightarrow (\underline{F}+T)*F\dashv \Rightarrow (\mathtt{a}+\underline{T})*F\dashv$$

Equivalence of CFG and PDA



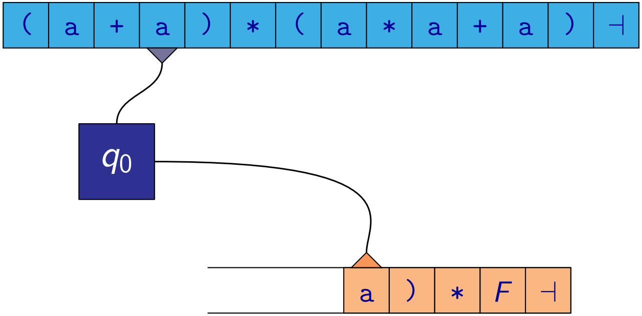
$$\cdots \Rightarrow (\underline{E}{+}T){*}F\dashv \Rightarrow (\underline{T}{+}T){*}F\dashv \Rightarrow (\underline{F}{+}T){*}F\dashv \Rightarrow (\mathrm{a}{+}\underline{T}){*}F\dashv$$

Equivalence of CFG and PDA



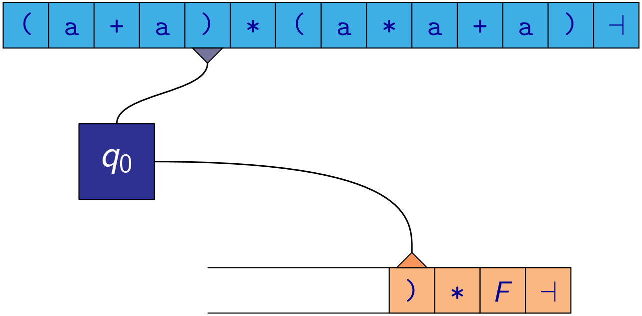
$\cdots \Rightarrow (\underline{T}+T)*F\dashv \Rightarrow (\underline{F}+T)*F\dashv \Rightarrow (\mathrm{a}+\underline{T})*F\dashv \Rightarrow (\mathrm{a}+\underline{F})*F\dashv$

Equivalence of CFG and PDA



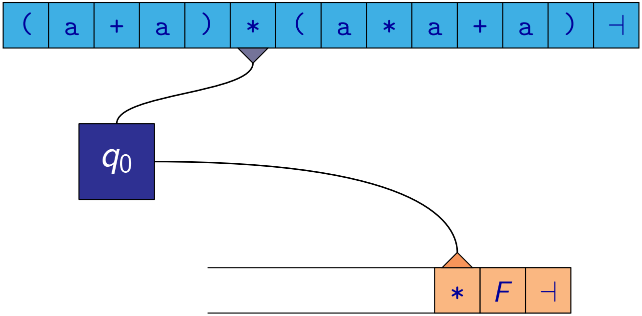
$$\cdots \Rightarrow (\underline{F}+T)*F\dashv \;\Rightarrow\; (a+\underline{T})*F\dashv \;\Rightarrow\; (a+\underline{F})*F\dashv \;\Rightarrow\; (a+a)*\underline{F}\dashv$$

Equivalence of CFG and PDA



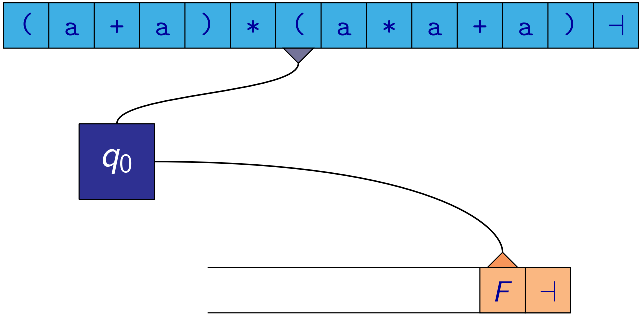
$$\cdots \Rightarrow (\underline{F}+T)*F\dashv \Rightarrow (a+\underline{T})*F\dashv \Rightarrow (a+\underline{F})*F\dashv \Rightarrow (a+a)*\underline{F}\dashv$$

Equivalence of CFG and PDA



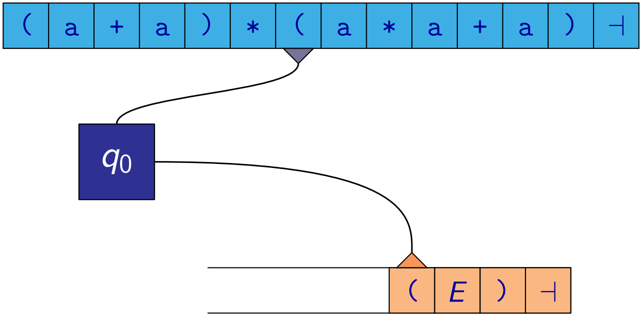
$$\cdots \Rightarrow (\underline{F}+T)*F\dashv \Rightarrow (\mathtt{a}+\underline{T})*F\dashv \Rightarrow (\mathtt{a}+\underline{F})*F\dashv \Rightarrow (\mathtt{a}+\mathtt{a})*\underline{F}\dashv$$

Equivalence of CFG and PDA



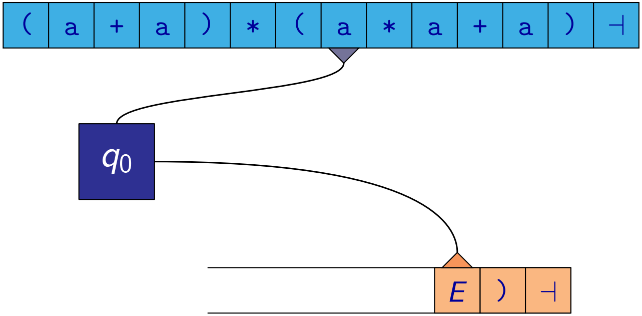
$$\cdots \Rightarrow (\underline{F}+T)*F\dashv \Rightarrow (\mathrm{a}+\underline{T})*F\dashv \Rightarrow (\mathrm{a}+\underline{F})*F\dashv \Rightarrow (\mathrm{a}+\mathrm{a})*\underline{F}\dashv$$

Equivalence of CFG and PDA



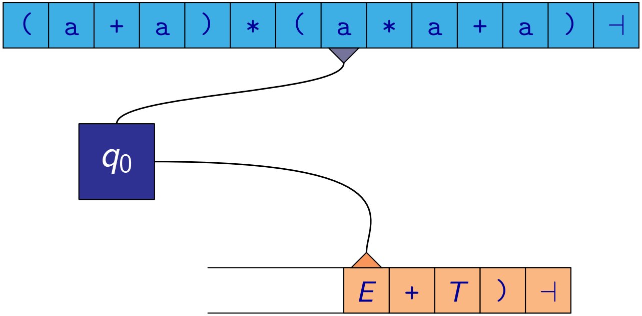
$\cdots \Rightarrow$ (a+$\underline{T}$)*F ⊣ $\Rightarrow$ (a+$\underline{F}$)*F ⊣ $\Rightarrow$ (a+a)*$\underline{F}$ ⊣ $\Rightarrow$ (a+a)*($\underline{E}$) ⊣

Equivalence of CFG and PDA



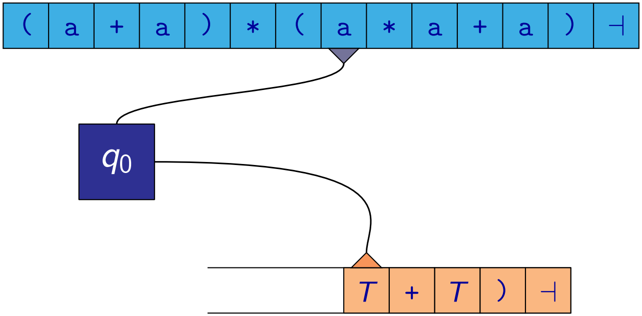
$$\cdots \Rightarrow (a+\underline{T})*F\dashv \Rightarrow (a+\underline{F})*F\dashv \Rightarrow (a+a)*\underline{F}\dashv \Rightarrow (a+a)*(\underline{E})\dashv$$

Equivalence of CFG and PDA



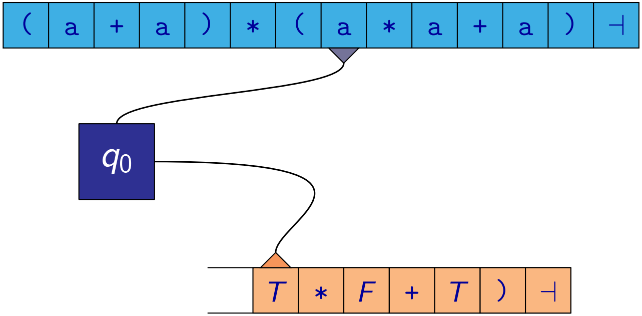
$\cdots \Rightarrow$ (a+a)*$\underline{F}$⊣ $\Rightarrow$ (a+a)*($\underline{E}$)⊣ $\Rightarrow$ (a+a)*($\underline{E}$+T)⊣

Equivalence of CFG and PDA



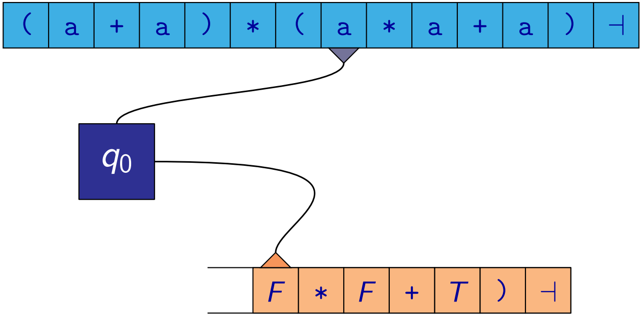
$$\cdots \Rightarrow \texttt{(a+a)*(}\underline{E}\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(}\underline{E}\texttt{+}T\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(}\underline{T}\texttt{+}T\texttt{)}\dashv$$

Equivalence of CFG and PDA



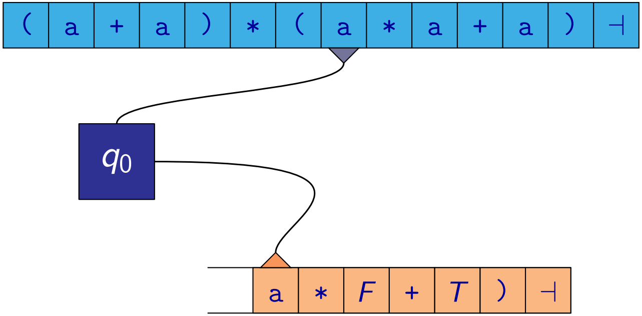
$$\cdots \Rightarrow \texttt{(a+a)*(}\underline{E}\texttt{+}T\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(}\underline{T}\texttt{+}T\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(}\underline{T}\texttt{*}F\texttt{+}T\texttt{)}\dashv$$

Equivalence of CFG and PDA



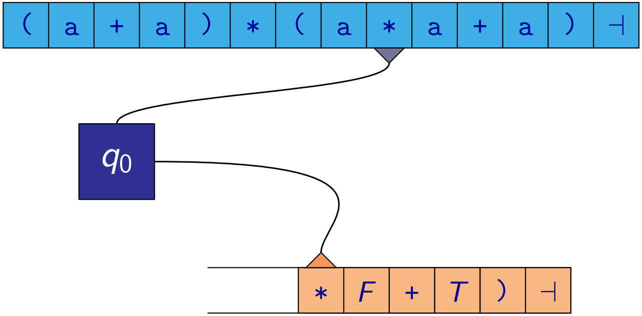
$$\cdots \Rightarrow \texttt{(a+a)*(}\underline{T}\texttt{*}F\texttt{+}T\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(}\underline{F}\texttt{*}F\texttt{+}T\texttt{)}\dashv$$

Equivalence of CFG and PDA



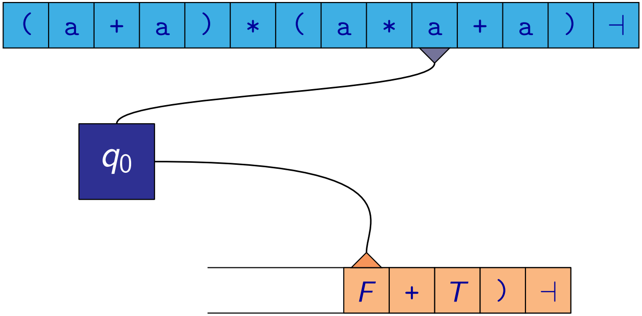
$$\cdots \Rightarrow \texttt{(a+a)*(}\underline{F}\texttt{*}F\texttt{+}T\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(a*}\underline{F}\texttt{+}T\texttt{)}\dashv$$

Equivalence of CFG and PDA



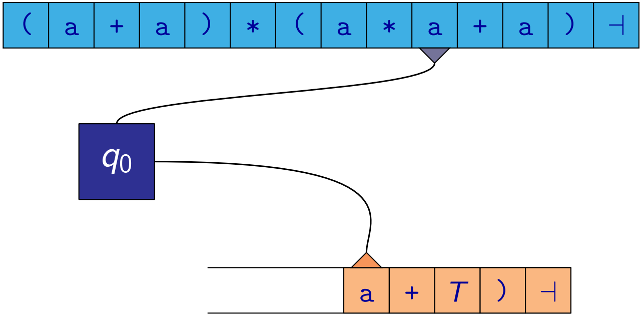
$\cdots \Rightarrow$ (a+a)*($\underline{F}$*F+T)⊣ $\Rightarrow$ (a+a)*(a*$\underline{F}$+T)⊣

Equivalence of CFG and PDA



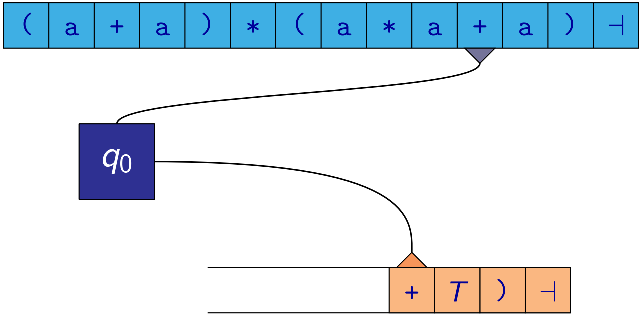
$$\cdots \Rightarrow \texttt{(a+a)*(}\underline{F}\texttt{*}F\texttt{+}T\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(a*}\underline{F}\texttt{+}T\texttt{)}\dashv$$

Equivalence of CFG and PDA



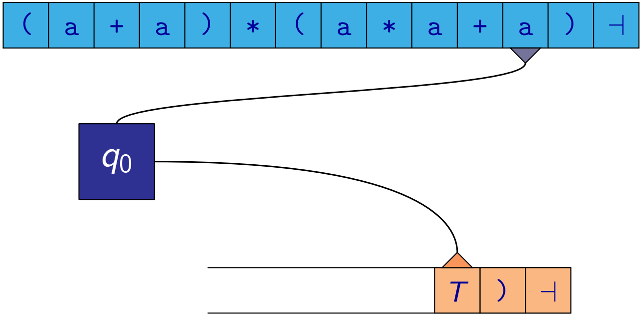
$\cdots \Rightarrow$ (a+a)*(a*$\underline{F}$+T)$\dashv$ $\Rightarrow$ (a+a)*(a*a+$\underline{T}$)$\dashv$

Equivalence of CFG and PDA



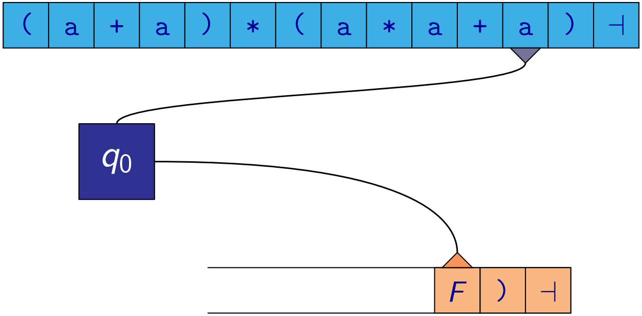
$\cdots \Rightarrow$ `(a+a)*(a*`$\underline{F}$`+`T`)`$\dashv$ $\Rightarrow$ `(a+a)*(a*a+`$\underline{T}$`)`$\dashv$

Equivalence of CFG and PDA



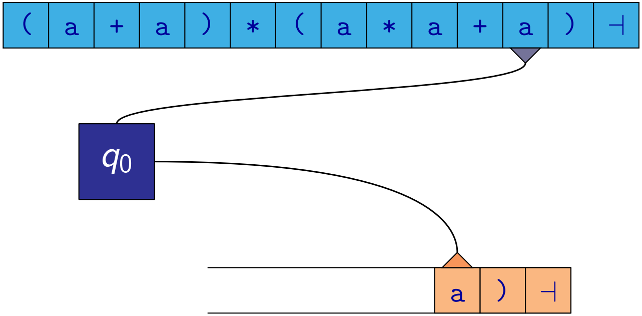
$\cdots \;\Rightarrow\; \texttt{(a+a)*(a*}\underline{F}\texttt{+}T\texttt{)}\dashv \;\Rightarrow\; \texttt{(a+a)*(a*a+}\underline{T}\texttt{)}\dashv$

Equivalence of CFG and PDA



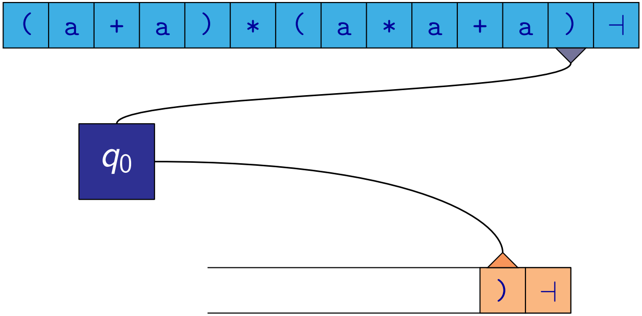
$$\cdots \Rightarrow \texttt{(a+a)*(a*a+}\underline{T}\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(a*a+}\underline{F}\texttt{)}\dashv$$

Equivalence of CFG and PDA



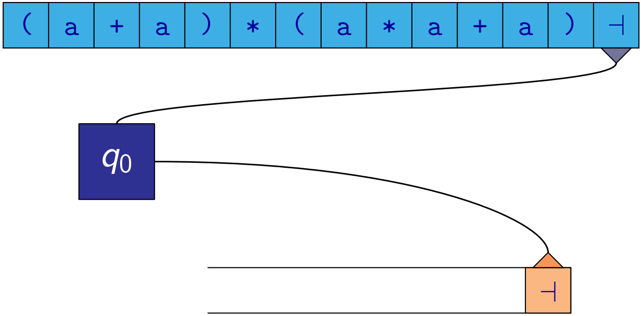
$$\cdots \Rightarrow \texttt{(a+a)*(a*a+}\underline{F}\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(a*a+a)}\dashv$$

Equivalence of CFG and PDA



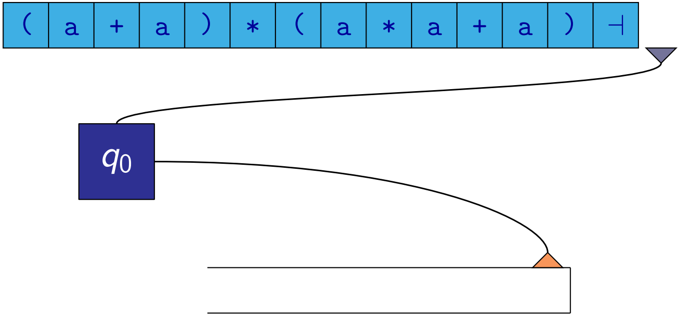
$\cdots \Rightarrow$ (a+a)*(a*a+$\underline{F}$)$\dashv$ $\Rightarrow$ (a+a)*(a*a+a)$\dashv$

Equivalence of CFG and PDA



$$\cdots \Rightarrow \texttt{(a+a)*(a*a+}\underline{F}\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(a*a+a)}\dashv$$

Equivalence of CFG and PDA



$$\cdots \Rightarrow \texttt{(a+a)*(a*a+}\underline{F}\texttt{)}\dashv \Rightarrow \texttt{(a+a)*(a*a+a)}\dashv$$

Equivalence of CFG and PDA

We can see from the previous example that the pushdown automaton $\mathcal{M}$ basically performs a **left derivation** in grammar $\mathcal{G}$.

It can be easily shown that:

- For every left derivation in grammar $\mathcal{G}$ there is some corresponding computation of automaton $\mathcal{M}$.
- For every computation of automaton $\mathcal{M}$ there is some corresponding left derivation in grammar $\mathcal{G}$.

Remark: The described approach corresponds to the syntactic analysis that proceeds **top down**.

Equivalence of CFG and PDA

Alternatively, it is also possible to proceed from **bottom up**.

This could be implemented by a nondeterministic pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, X_0)$ constructed for a given grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ as follows:

- $\Gamma = \Pi \cup \Sigma \cup \{\vdash\}$, where $\vdash \notin (\Pi \cup \Sigma)$
- $X_0 = \vdash$
- Q contains states corresponding to all suffixes of right-hand sides from P a also a special state $\langle S \rangle$ (where $S \in \Pi$ is the initial nonterminal of grammar $\mathcal{G}$) and a special state q_{acc}.

  A state corresponding to suffix α (where $\alpha \in (\Pi \cup \Sigma)^*$) will be denoted $\langle \alpha \rangle$.

  A special case is a state corresponding to suffix ε. This state will be denoted $\langle \rangle$.
- $q_0 = \langle \rangle$

Equivalence of CFG and PDA

- For every input symbol $a \in \Sigma$ and every stack symbol $W \in \Gamma$ the following rule is added to δ:

$$\langle\rangle W \xrightarrow{a} \langle\rangle aW$$

- For every rule $X \rightarrow Y_1 Y_2 \cdots Y_n$ from grammar $\mathcal{G}$ (where $X \in \Pi$, $n \geq 0$, and $Y_i \in (\Pi \cup \Sigma)$ for $1 \leq i \leq n$) the following set of rules is added to δ:

$$\begin{gathered}
\langle\rangle Y_n \xrightarrow{\varepsilon} \langle Y_n\rangle \\
\langle Y_n\rangle Y_{n-1} \xrightarrow{\varepsilon} \langle Y_{n-1} Y_n\rangle \\
\langle Y_{n-1} Y_n\rangle Y_{n-2} \xrightarrow{\varepsilon} \langle Y_{n-2} Y_{n-1} Y_n\rangle \\
\vdots \\
\langle Y_2 Y_3 \ldots Y_n\rangle Y_1 \xrightarrow{\varepsilon} \langle Y_1 Y_2 Y_3 \cdots Y_n\rangle
\end{gathered}$$

and for every $W \in \Gamma$ we add the rules

$$\langle Y_1 Y_2 \cdots Y_n\rangle W \xrightarrow{\varepsilon} \langle\rangle XW$$

Equivalence of CFG and PDA

- For example if grammar $\mathcal{G}$ contains rule
$$B \rightarrow CaADb$$
the transition function δ of automaton $\mathcal{M}$ will contain rules
$$\langle\rangle b \xrightarrow{\varepsilon} \langle b\rangle$$
$$\langle b\rangle D \xrightarrow{\varepsilon} \langle Db\rangle$$
$$\langle Db\rangle A \xrightarrow{\varepsilon} \langle ADb\rangle$$
$$\langle ADb\rangle a \xrightarrow{\varepsilon} \langle aADb\rangle$$
$$\langle aADb\rangle C \xrightarrow{\varepsilon} \langle CaADb\rangle$$
and also for every $W \in \Gamma$ the will be a rule
$$\langle CaADb\rangle W \xrightarrow{\varepsilon} \langle\rangle BW$$

Equivalence of CFG and PDA

- In particular, for ε-rules of grammar $\mathcal{G}$, the corresponding rules will be as follows: for ε-rule
$$X \ \longrightarrow \ \varepsilon$$
of grammar $\mathcal{G}$, where $X \in \Pi$, there will be corresponding rules
$$\langle\rangle W \xrightarrow{\varepsilon} \langle\rangle XW$$
where $W \in \Gamma$.

- We finish the construction by adding the following two special rules to δ (where $S \in \Pi$ is the initial nonterminal of grammar $\mathcal{G}$):
$$\langle\rangle S \xrightarrow{\varepsilon} \langle S\rangle \qquad\qquad \langle S\rangle\vdash \xrightarrow{\varepsilon} q_{acc}$$

Equivalence of CFG and PDA

Example: Consider the same grammar $\mathcal{G}$ as in the previous example:

$$\begin{aligned} S &\to E \dashv \\ E &\to T \mid E\texttt{+}T \\ T &\to F \mid T\texttt{*}F \\ F &\to \texttt{a} \mid \texttt{(}E\texttt{)} \end{aligned}$$

For this grammar we construct a corresponding pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, X_0)$, where

- $\Sigma = \{ \texttt{a}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}, \dashv \}$
- $\Gamma = \{ S, E, T, F, \texttt{a}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}, \dashv, \vdash \}$
- $Q = \{ \langle\rangle, \langle\dashv\rangle, \langle E \dashv\rangle, \langle T\rangle, \langle\texttt{+}T\rangle, \langle E\texttt{+}T\rangle, \langle F\rangle, \langle\texttt{*}F\rangle, \langle T\texttt{*}F\rangle,$
  $\langle\texttt{a}\rangle, \langle\texttt{)}\rangle, \langle E\texttt{)}\rangle, \langle\texttt{(}E\texttt{)}\rangle, \langle S\rangle, q_{acc} \}$
- $q_0 = \langle\rangle$
- $X_0 = \vdash$

Equivalence of CFG and PDA

For each $X \in \Gamma$ the following rules are added to δ:

$$\langle\rangle X \xrightarrow{\mathtt{a}} \langle\rangle \mathtt{a} X$$
$$\langle\rangle X \xrightarrow{+} \langle\rangle + X$$
$$\langle\rangle X \xrightarrow{*} \langle\rangle * X$$
$$\langle\rangle X \xrightarrow{(} \langle\rangle (X$$
$$\langle\rangle X \xrightarrow{)} \langle\rangle) X$$
$$\langle\rangle X \xrightarrow{\dashv} \langle\rangle \dashv X$$

$$\langle\rangle S \xrightarrow{\varepsilon} \langle S\rangle$$
$$\langle S\rangle \vdash \xrightarrow{\varepsilon} q_{acc}$$

$$\langle\rangle \dashv \xrightarrow{\varepsilon} \langle\dashv\rangle$$
$$\langle\dashv\rangle E \xrightarrow{\varepsilon} \langle E\dashv\rangle \qquad \langle E\dashv\rangle X \xrightarrow{\varepsilon} \langle\rangle S X$$

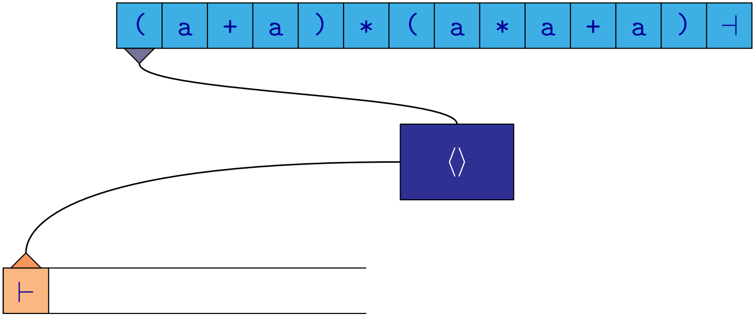
$$\langle\rangle T \xrightarrow{\varepsilon} \langle T\rangle \qquad \langle T\rangle X \xrightarrow{\varepsilon} \langle\rangle E X$$
$$\langle T\rangle + \xrightarrow{\varepsilon} \langle +T\rangle$$
$$\langle +T\rangle E \xrightarrow{\varepsilon} \langle E+T\rangle \qquad \langle E+T\rangle X \xrightarrow{\varepsilon} \langle\rangle E X$$

$$\langle\rangle F \xrightarrow{\varepsilon} \langle F\rangle \qquad \langle F\rangle X \xrightarrow{\varepsilon} \langle\rangle T X$$
$$\langle F\rangle * \xrightarrow{\varepsilon} \langle *F\rangle$$
$$\langle *F\rangle T \xrightarrow{\varepsilon} \langle T*F\rangle \qquad \langle T*F\rangle X \xrightarrow{\varepsilon} \langle\rangle T X$$

$$\langle\rangle \mathtt{a} \xrightarrow{\varepsilon} \langle \mathtt{a}\rangle \qquad \langle \mathtt{a}\rangle X \xrightarrow{\varepsilon} \langle\rangle F X$$

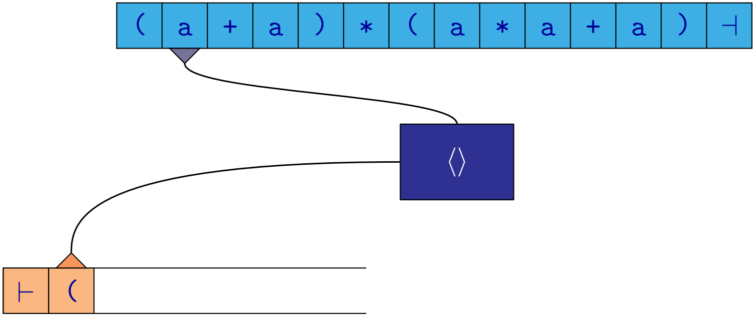
$$\langle\rangle) \xrightarrow{\varepsilon} \langle)\rangle$$
$$\langle)\rangle E \xrightarrow{\varepsilon} \langle E)\rangle$$
$$\langle E)\rangle (\xrightarrow{\varepsilon} \langle (E)\rangle \qquad \langle (E)\rangle X \xrightarrow{\varepsilon} \langle\rangle F X$$

Equivalence of CFG and PDA



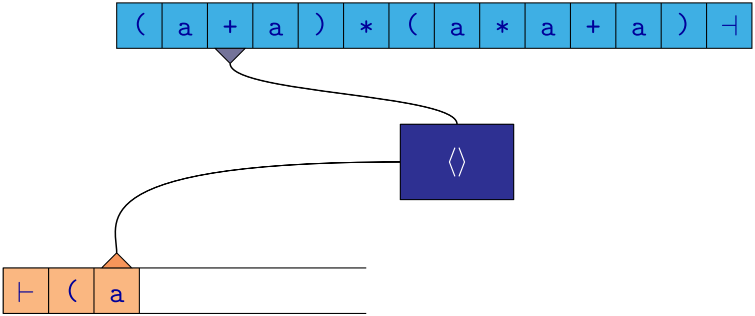
$(a+a)*(a*a+a) \dashv$

Equivalence of CFG and PDA



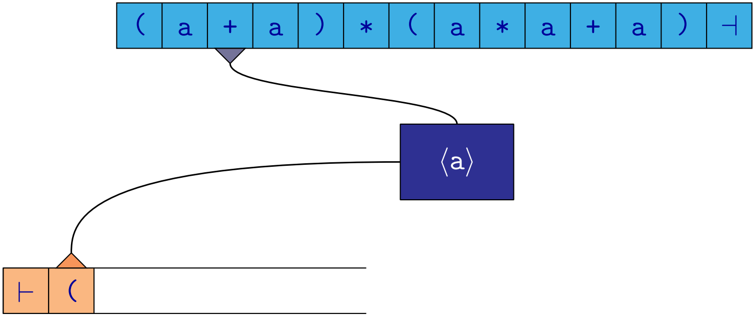
$\texttt{(a+a)*(a*a+a)}\dashv$

Equivalence of CFG and PDA



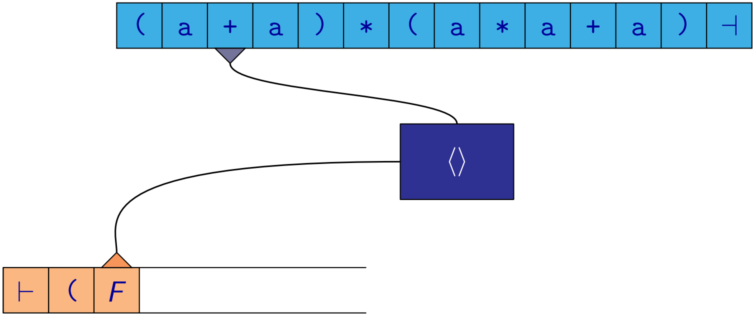
(a+a)*(a*a+a)⊣

Equivalence of CFG and PDA



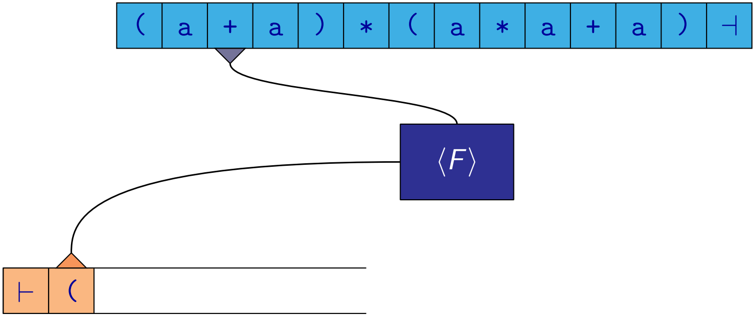
(a+a)*(a*a+a) ⊣

Equivalence of CFG and PDA



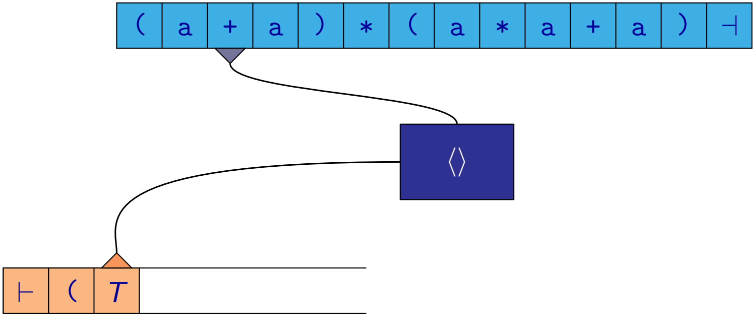
$(\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \texttt{(a+a)*(a*a+a)}\dashv$

Equivalence of CFG and PDA



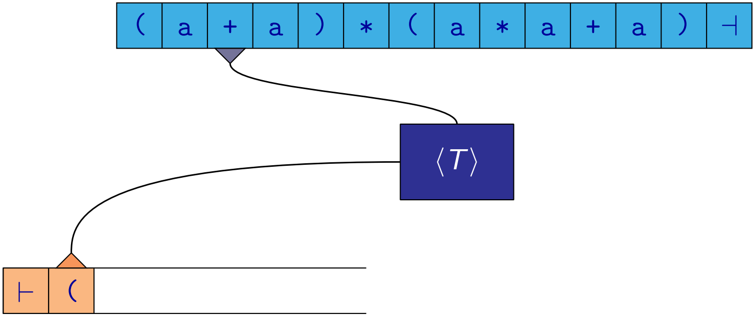
$(\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \texttt{(a+a)*(a*a+a)}\dashv$

Equivalence of CFG and PDA



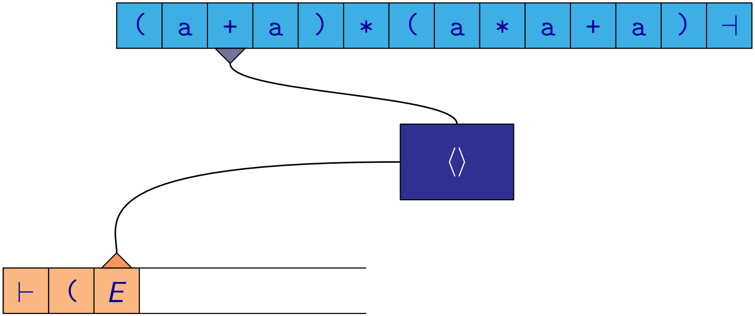
$(\underline{T}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \texttt{(a+a)*(a*a+a)}\dashv$

Equivalence of CFG and PDA



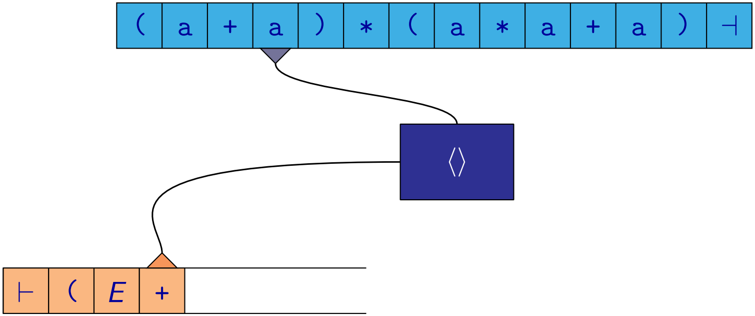
$(\underline{T}+a)*(a*a+a)\dashv \ \Rightarrow\ (\underline{F}+a)*(a*a+a)\dashv \ \Rightarrow\ (a+a)*(a*a+a)\dashv$

Equivalence of CFG and PDA



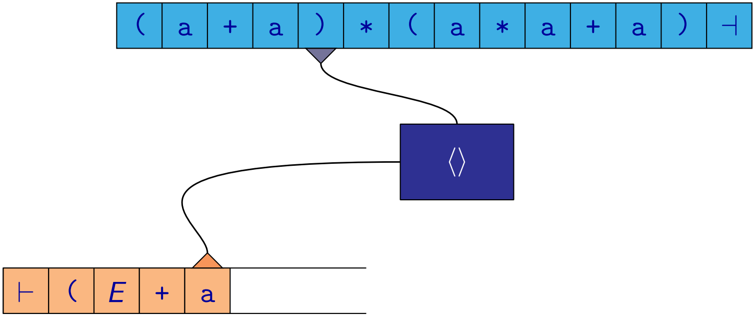
$(\underline{E}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{T}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



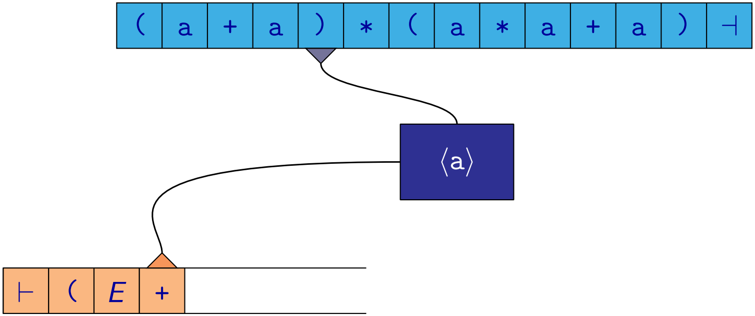
$(\underline{E}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{T}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



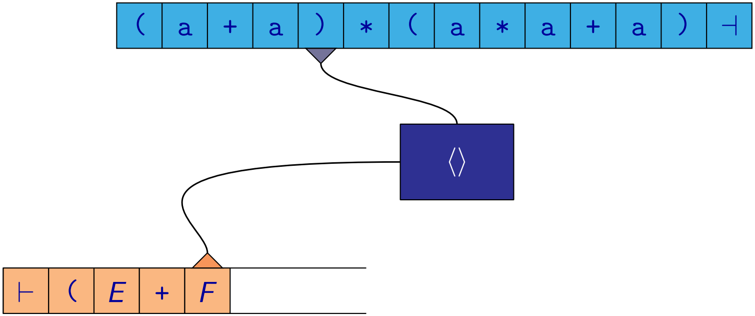
$$(\underline{E}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{T}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



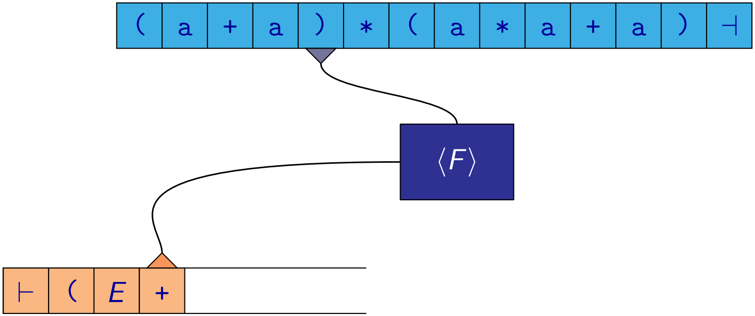
$(\underline{E}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{T}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; (\underline{F}\texttt{+a)*(a*a+a)}\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



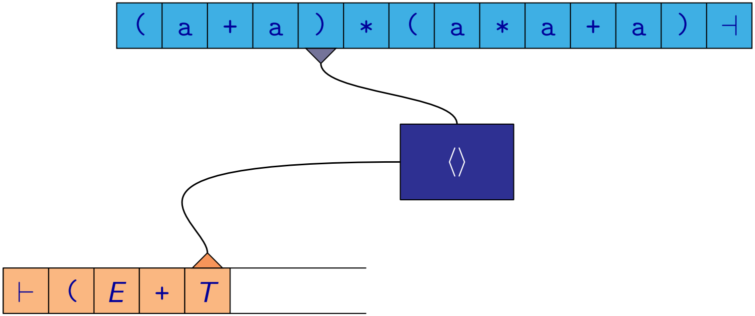
$$(E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; (\underline{E}+a)*(a*a+a)\dashv \;\Rightarrow\; (\underline{T}+a)*(a*a+a)\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



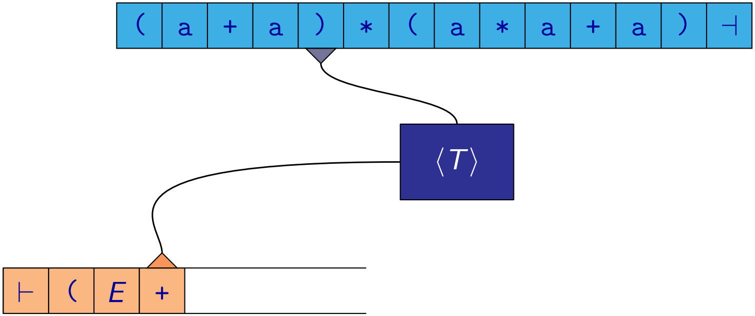
$$(E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; (\underline{E}+a)*(a*a+a)\dashv \;\Rightarrow\; (\underline{T}+a)*(a*a+a)\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



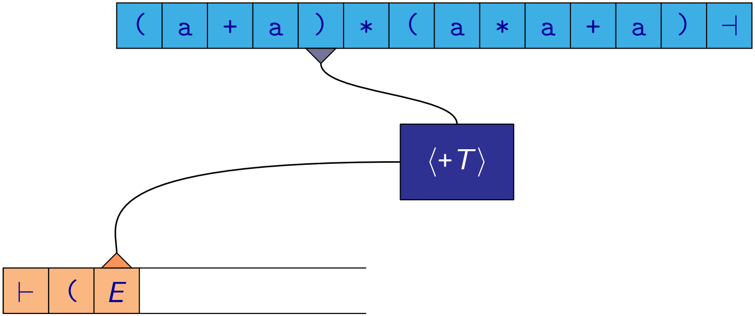
$$(E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; (\underline{E}+a)*(a*a+a)\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



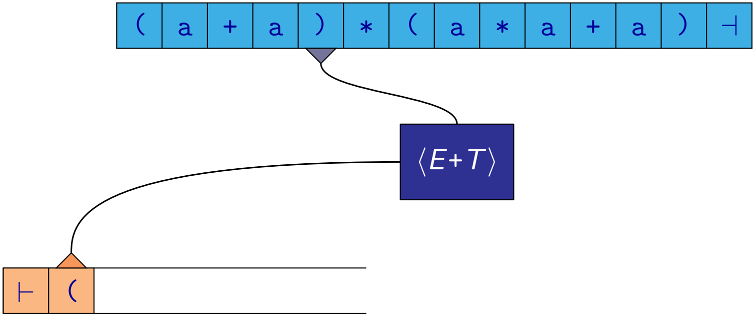
$$(E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; (\underline{E}+a)*(a*a+a)\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



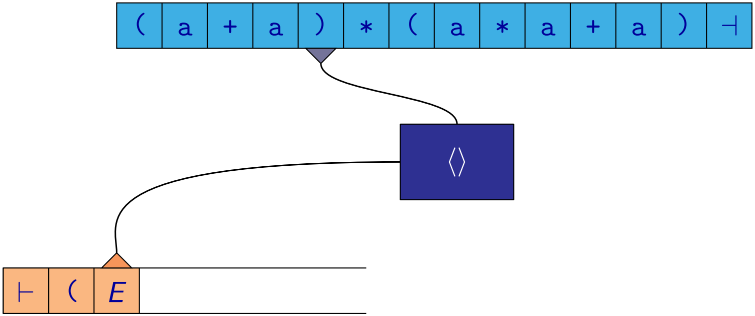
$(E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; (\underline{E}+a)*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



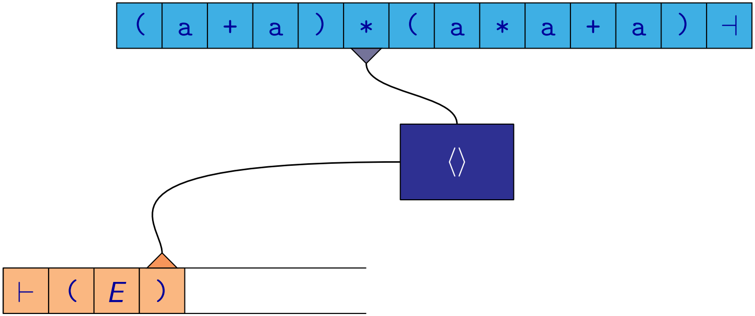
$$(E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; (\underline{E}+a)*(a*a+a)\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



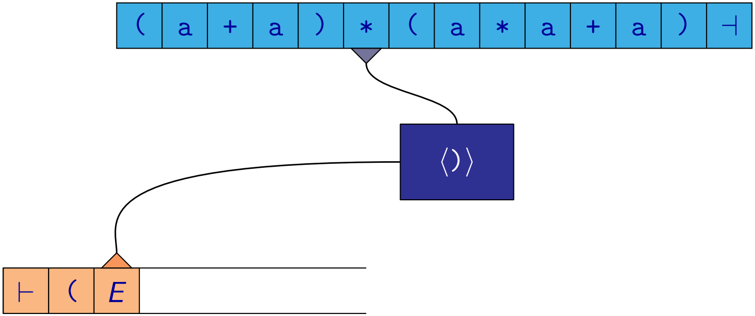
$(\underline{E})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



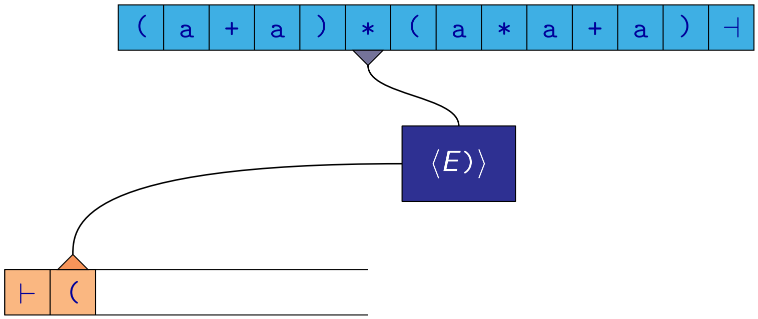
$(\underline{E})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



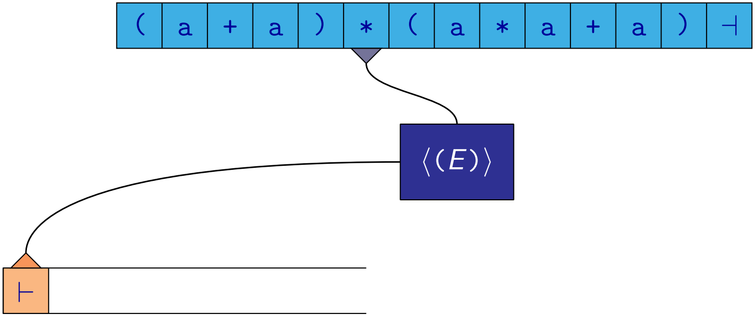
$(\underline{E})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



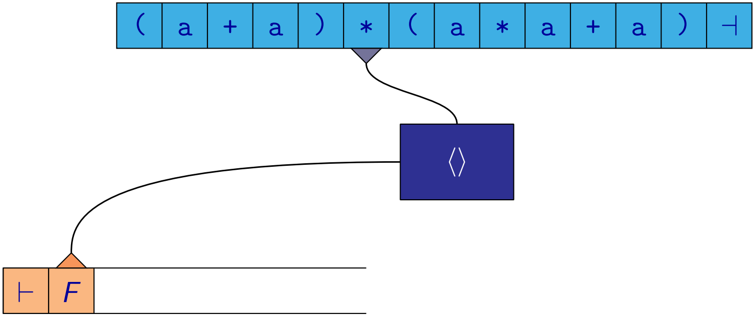
$(\underline{E})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



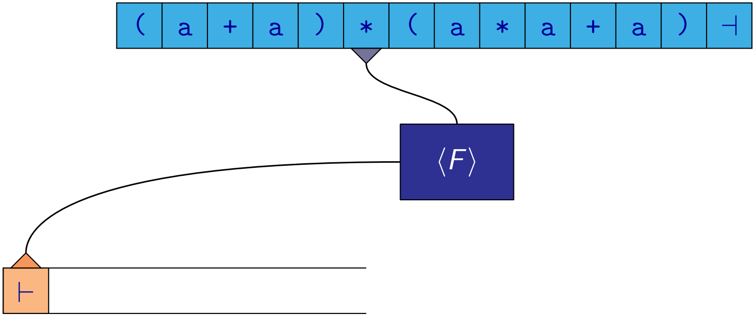
$$(\underline{E})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{T})*(a*a+a)\dashv \;\Rightarrow\; (E+\underline{F})*(a*a+a)\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



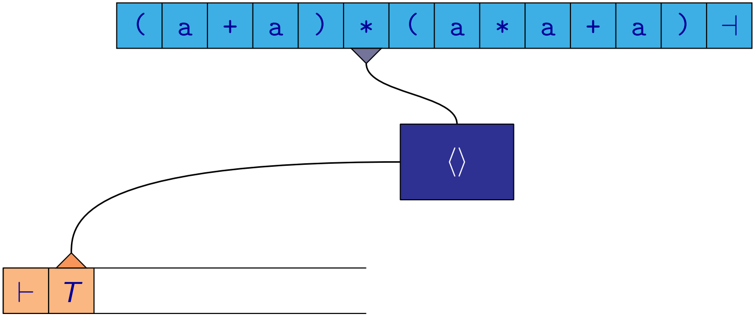
$\underline{F}$∗(a∗a+a)⊣ $\Rightarrow$ ($\underline{E}$)∗(a∗a+a)⊣ $\Rightarrow$ (E+$\underline{T}$)∗(a∗a+a)⊣ $\Rightarrow$ $\cdots$

Equivalence of CFG and PDA



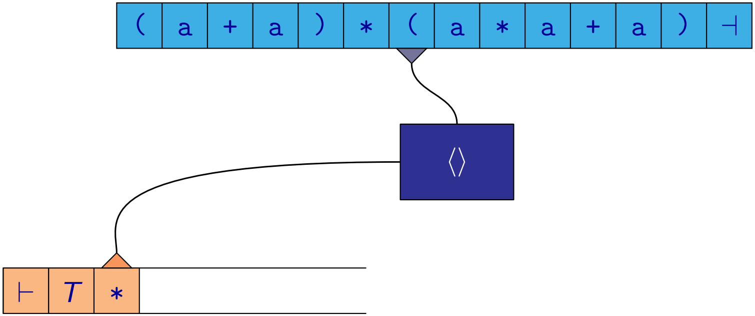
$\underline{F}$∗(a∗a+a)⊣ $\Rightarrow$ ($\underline{E}$)∗(a∗a+a)⊣ $\Rightarrow$ (E+$\underline{T}$)∗(a∗a+a)⊣ $\Rightarrow$ $\cdots$

Equivalence of CFG and PDA



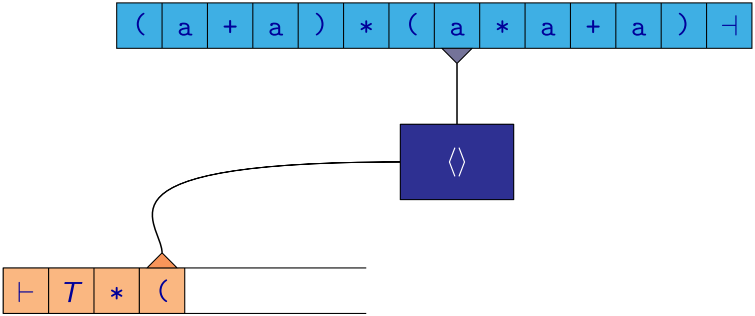
$\underline{T}$*(a*a+a)⊣ $\Rightarrow$ $\underline{F}$*(a*a+a)⊣ $\Rightarrow$ ($\underline{E}$)*(a*a+a)⊣ $\Rightarrow$ $\cdots$

Equivalence of CFG and PDA



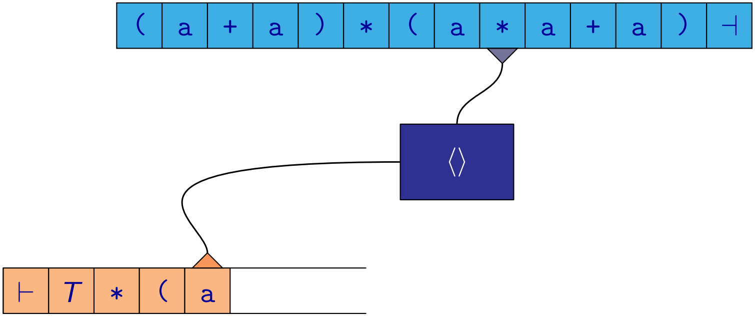
$\underline{T}$∗(a∗a+a)$\dashv$ $\Rightarrow$ $\underline{F}$∗(a∗a+a)$\dashv$ $\Rightarrow$ ($\underline{E}$)∗(a∗a+a)$\dashv$ $\Rightarrow$ $\cdots$

Equivalence of CFG and PDA



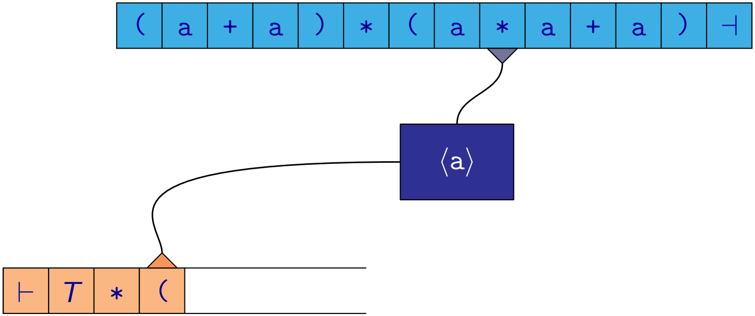
$\underline{T}$*(a*a+a)⊣ $\Rightarrow$ $\underline{F}$*(a*a+a)⊣ $\Rightarrow$ ($\underline{E}$)*(a*a+a)⊣ $\Rightarrow$ $\cdots$

Equivalence of CFG and PDA



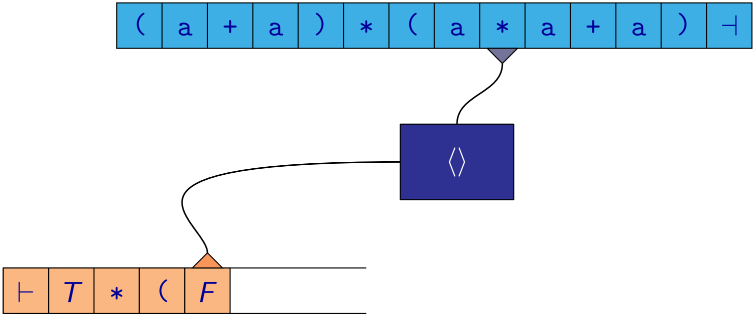
$\underline{T}*(a*a+a)\dashv \;\Rightarrow\; \underline{F}*(a*a+a)\dashv \;\Rightarrow\; (\underline{E})*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



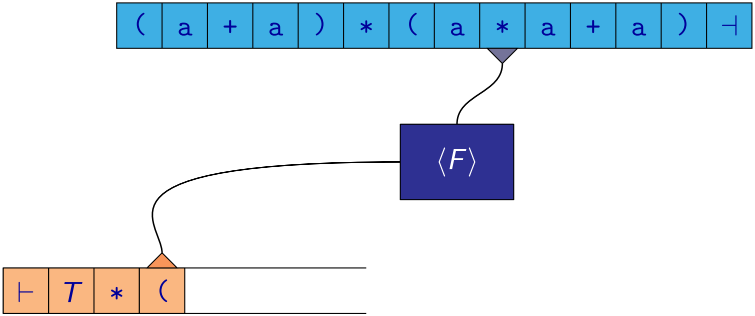
$\underline{T}$*(a*a+a)$\dashv$ $\Rightarrow$ $\underline{F}$*(a*a+a)$\dashv$ $\Rightarrow$ ($\underline{E}$)*(a*a+a)$\dashv$ $\Rightarrow$ $\cdots$

Equivalence of CFG and PDA



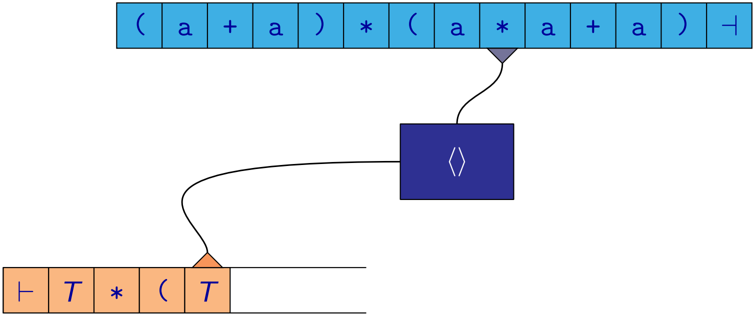
$T * (\underline{F} * a + a) \dashv \;\Rightarrow\; \underline{T} * (a * a + a) \dashv \;\Rightarrow\; \underline{F} * (a * a + a) \dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



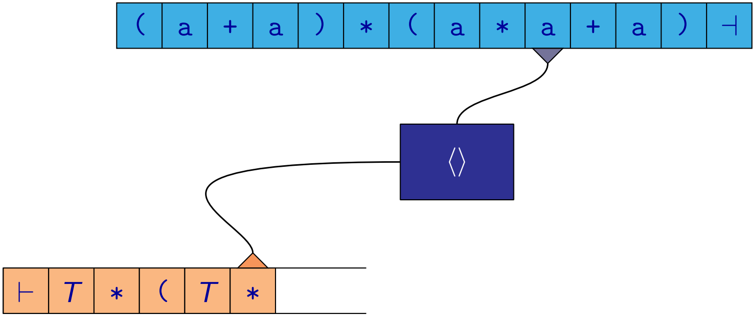
$$T*(\underline{F}*\texttt{a+a})\dashv \;\Rightarrow\; \underline{T}*(\texttt{a*a+a})\dashv \;\Rightarrow\; \underline{F}*(\texttt{a*a+a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



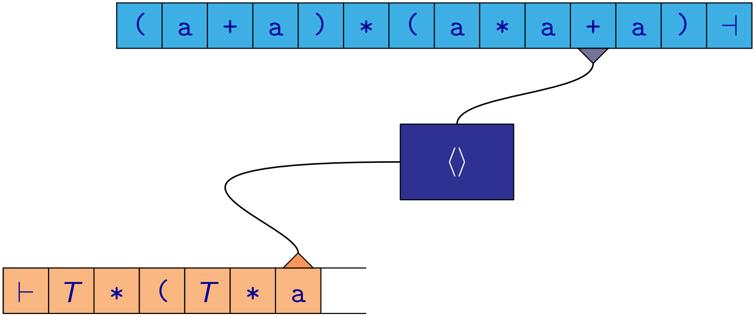
$T*(\underline{T}*a+a)\dashv \;\Rightarrow\; T*(\underline{F}*a+a)\dashv \;\Rightarrow\; \underline{T}*(a*a+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



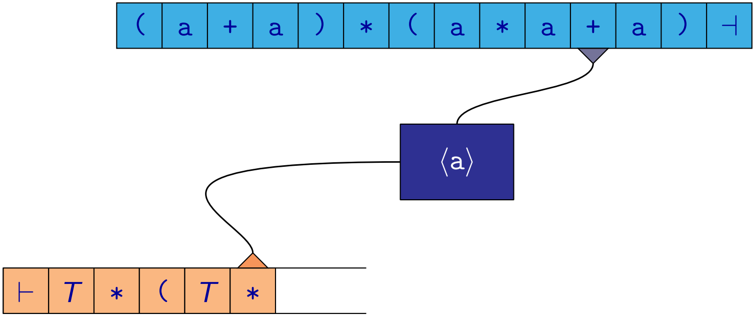
$T*(\underline{T}*\text{a+a})\dashv \;\Rightarrow\; T*(\underline{F}*\text{a+a})\dashv \;\Rightarrow\; \underline{T}*(\text{a*a+a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



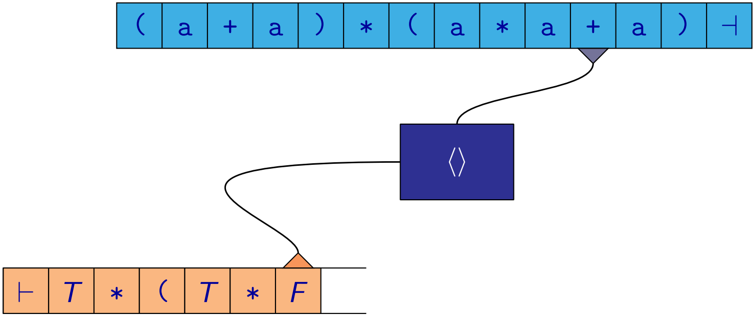
$T*(\underline{T}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{F}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; \underline{T}*(\mathtt{a}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



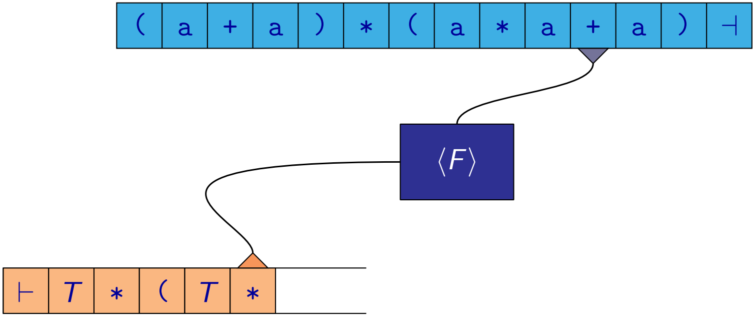
$T*(\underline{T}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{F}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; \underline{T}*(\mathtt{a}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



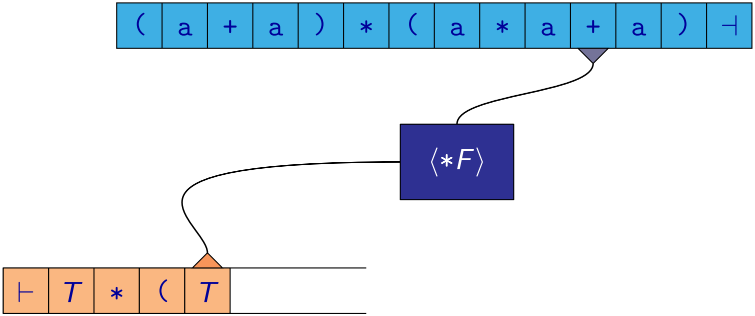
$T*(T*\underline{F}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{F}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



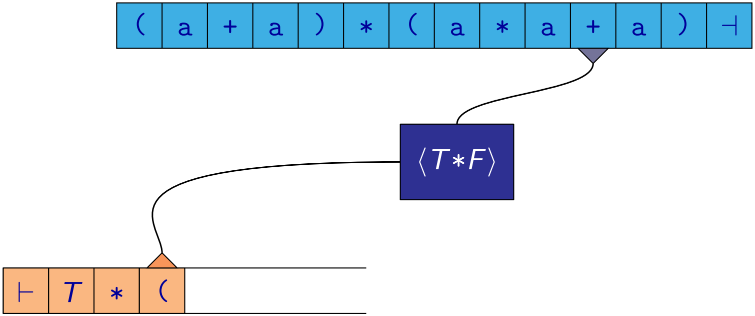
$$T*(T*\underline{F}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{F}*\mathtt{a}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



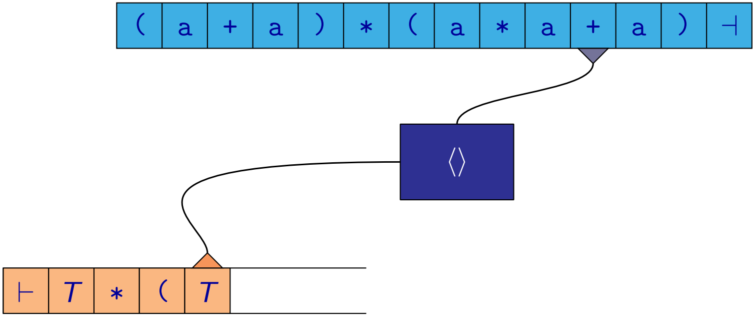
$$T*(T*\underline{F}+\mathrm{a})\dashv \;\Rightarrow\; T*(\underline{T}*\mathrm{a}+\mathrm{a})\dashv \;\Rightarrow\; T*(\underline{F}*\mathrm{a}+\mathrm{a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



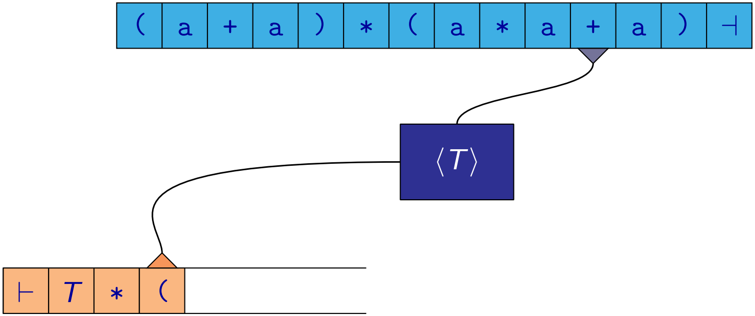
$T{*}(T{*}\underline{F}{+}\mathtt{a}){\dashv} \Rightarrow T{*}(\underline{T}{*}\mathtt{a}{+}\mathtt{a}){\dashv} \Rightarrow T{*}(\underline{F}{*}\mathtt{a}{+}\mathtt{a}){\dashv} \Rightarrow \cdots$

Equivalence of CFG and PDA



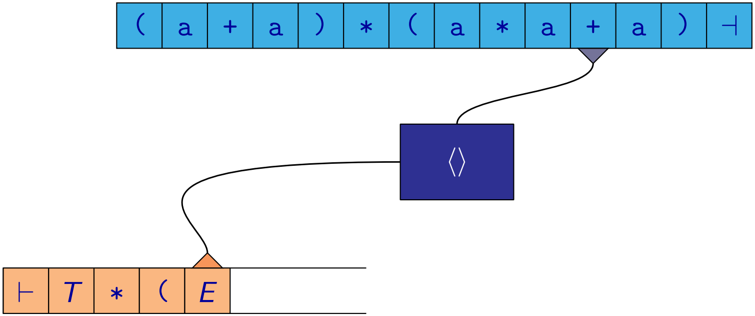
$T*(\underline{T}+\mathrm{a})\dashv \;\Rightarrow\; T*(T*\underline{F}+\mathrm{a})\dashv \;\Rightarrow\; T*(\underline{T}*\mathrm{a}+\mathrm{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



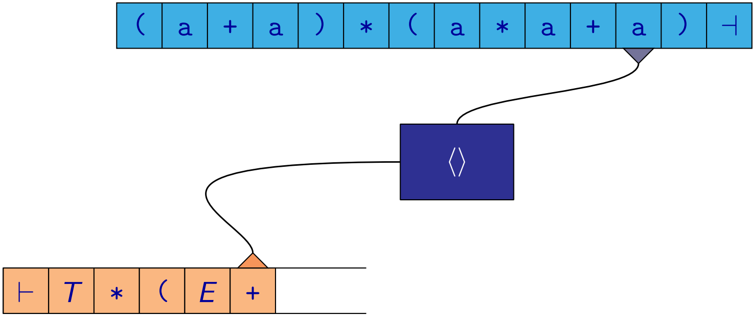
$T*(\underline{T}+\mathrm{a})\dashv \;\Rightarrow\; T*(T*\underline{F}+\mathrm{a})\dashv \;\Rightarrow\; T*(\underline{T}*\mathrm{a}+\mathrm{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



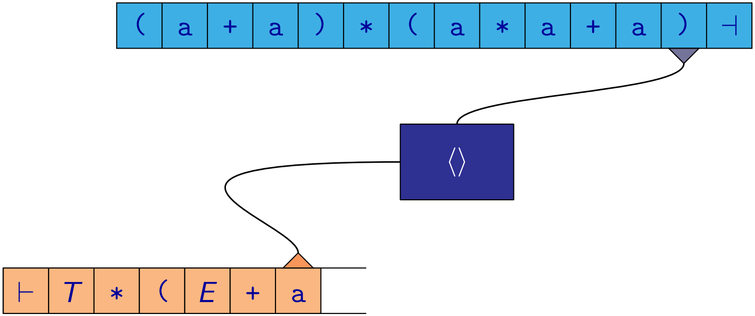
$T*(\underline{E}+a)\dashv \;\Rightarrow\; T*(\underline{T}+a)\dashv \;\Rightarrow\; T*(T*\underline{F}+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



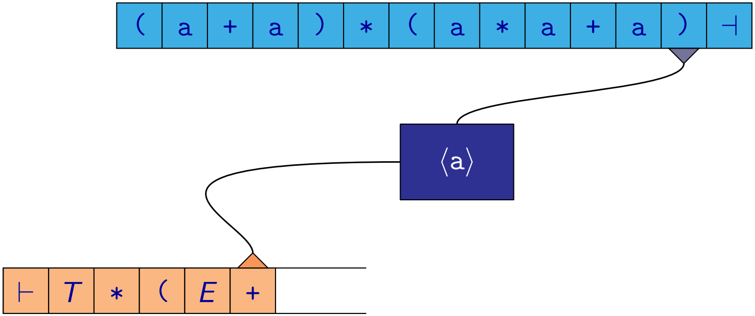
$T*(\underline{E}+a)\dashv \;\Rightarrow\; T*(\underline{T}+a)\dashv \;\Rightarrow\; T*(T*\underline{F}+a)\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



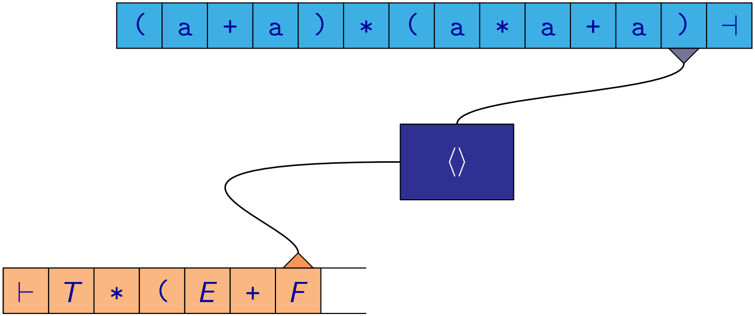
$$T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; T*(T*\underline{F}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



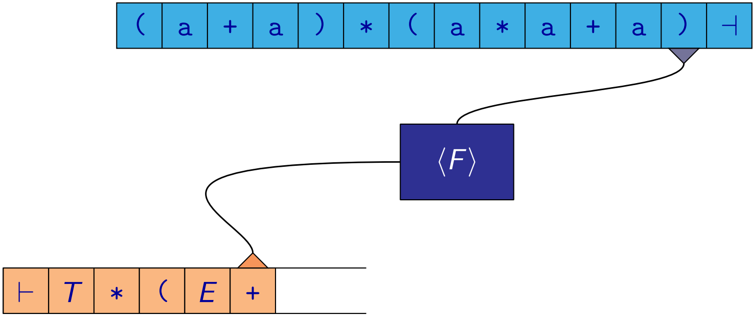
$T{*}(\underline{E}{+}\mathtt{a}){\dashv} \Rightarrow T{*}(\underline{T}{+}\mathtt{a}){\dashv} \Rightarrow T{*}(T{*}\underline{F}{+}\mathtt{a}){\dashv} \Rightarrow \cdots$

Equivalence of CFG and PDA



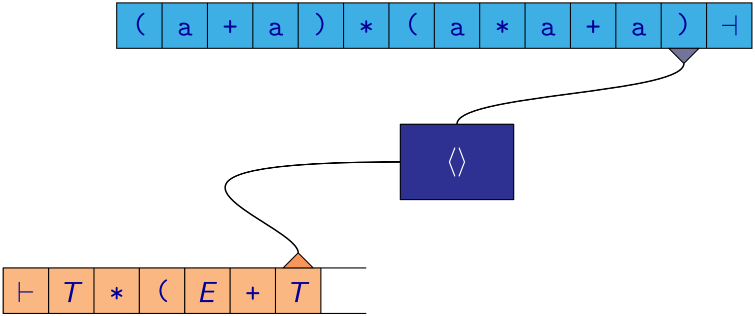
$$T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; T*(T*\underline{F}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



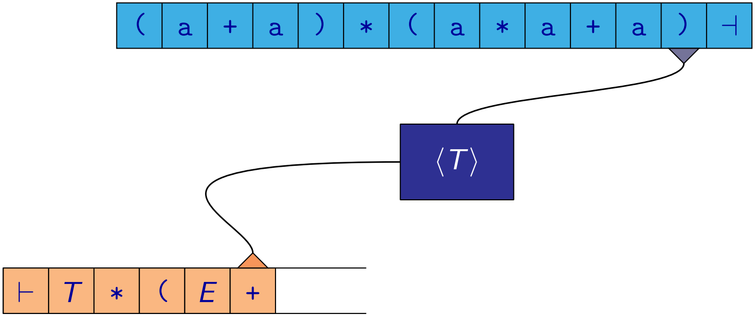
$$T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; T*(T*\underline{F}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



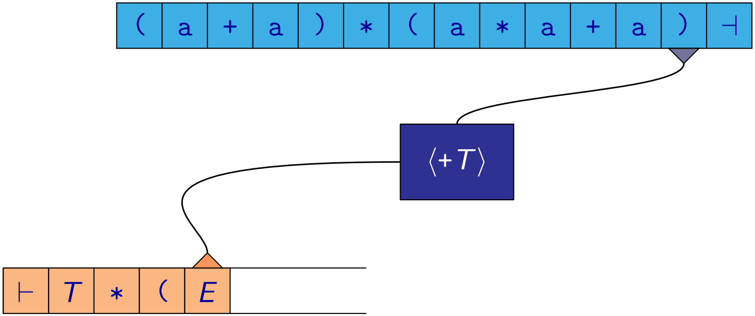
$$T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



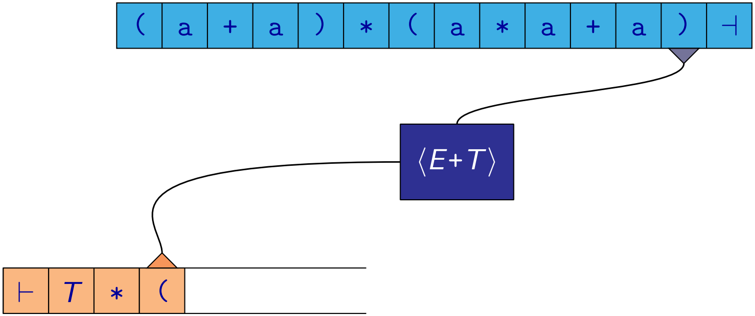
$T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



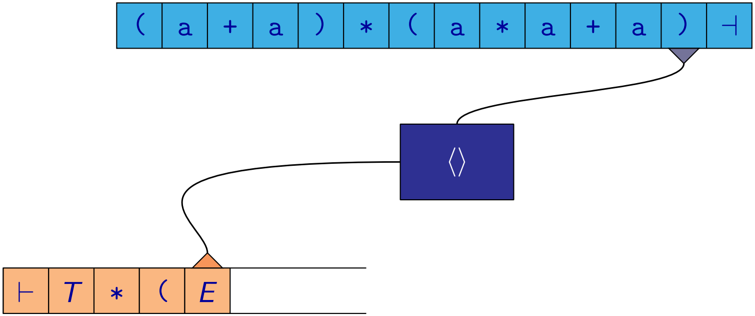
$T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



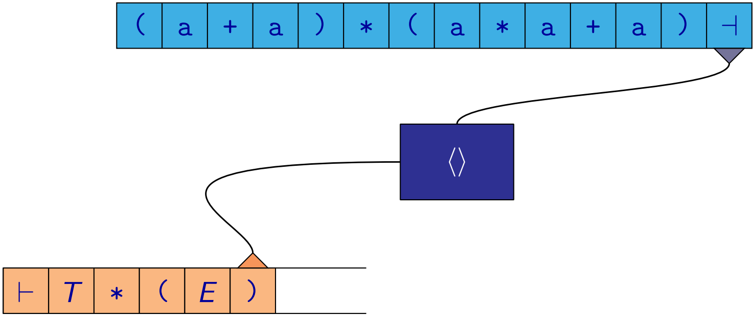
$T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; T*(\underline{T}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



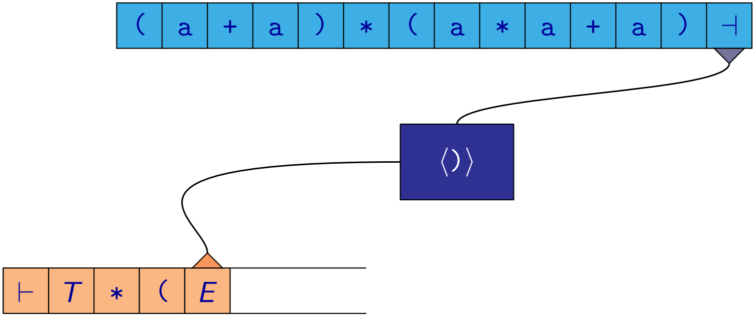
$T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; T{*}(E{+}\underline{F})\dashv \;\Rightarrow\; T{*}(\underline{E}{+}\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



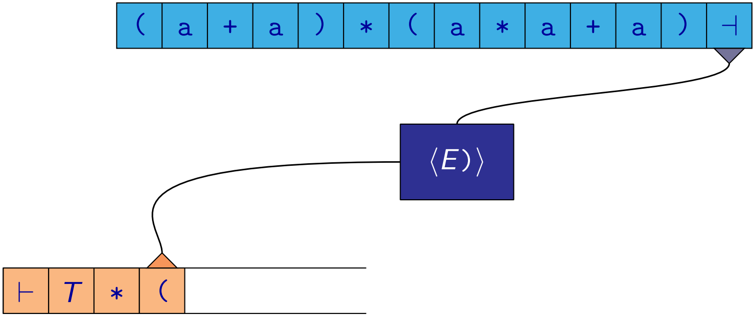
$T*(\underline{E})\dashv \;\Rightarrow\; T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



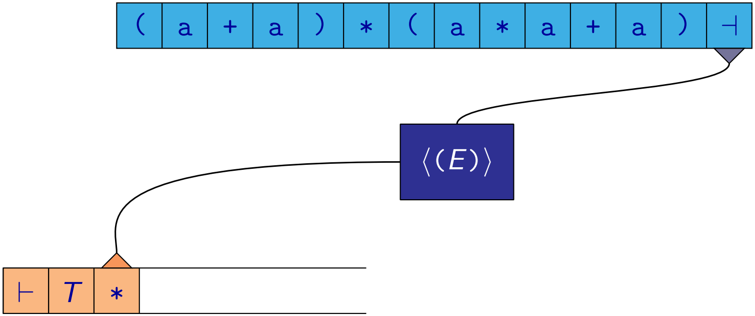
$T*(\underline{E})\dashv \;\Rightarrow\; T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathtt{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



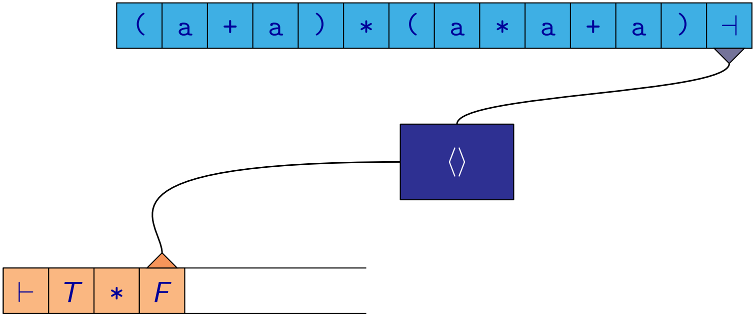
$T*(\underline{E})\dashv \;\Rightarrow\; T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; T*(\underline{E}+\mathrm{a})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



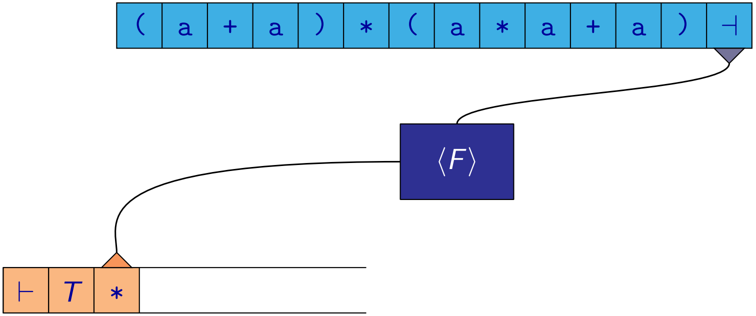
$T{*}(\underline{E}){\dashv} \Rightarrow T{*}(E{+}\underline{T}){\dashv} \Rightarrow T{*}(E{+}\underline{F}){\dashv} \Rightarrow T{*}(\underline{E}{+}\mathtt{a}){\dashv} \Rightarrow \cdots$

Equivalence of CFG and PDA



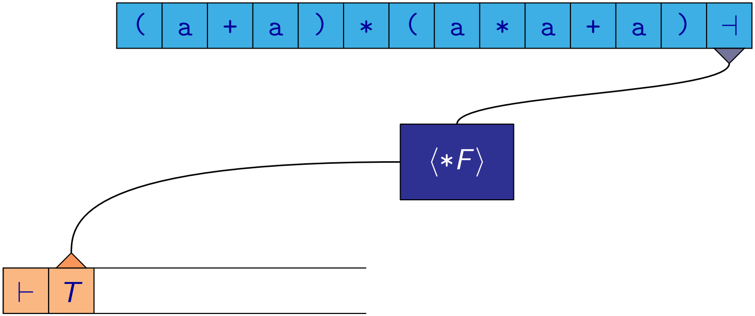
$$T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; T{*}(E{+}\underline{F})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



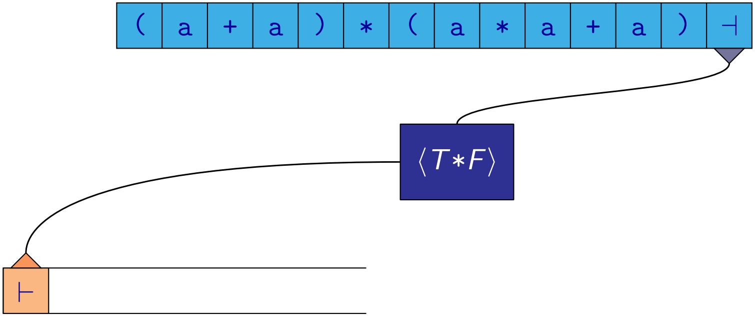
$T * \underline{F} \dashv \;\Rightarrow\; T * (\underline{E}) \dashv \;\Rightarrow\; T * (E + \underline{T}) \dashv \;\Rightarrow\; T * (E + \underline{F}) \dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



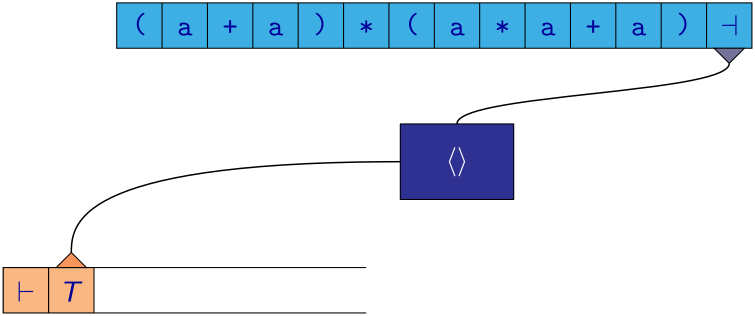
$T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; T{*}(E{+}\underline{F})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



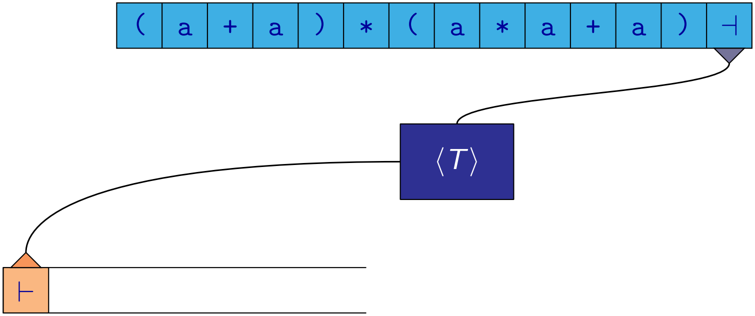
$T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; T{*}(E{+}\underline{F})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



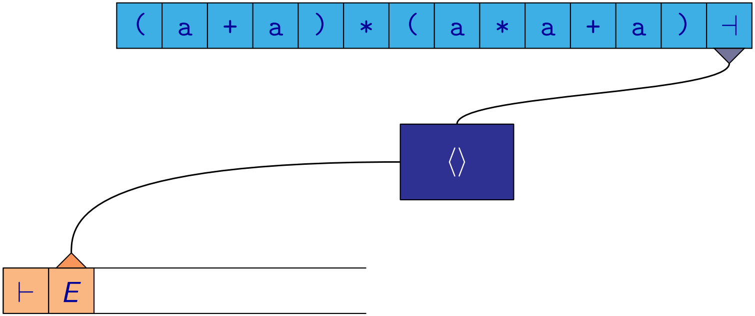
$\underline{T}\dashv \;\Rightarrow\; T*\underline{F}\dashv \;\Rightarrow\; T*(\underline{E})\dashv \;\Rightarrow\; T*(E+\underline{T})\dashv \;\Rightarrow\; T*(E+\underline{F})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



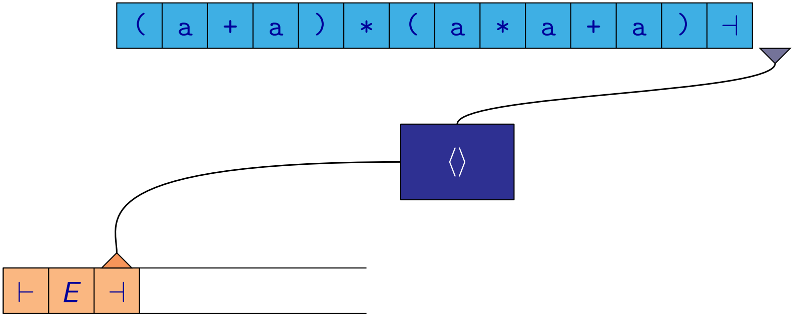
$$\underline{T}\dashv \;\Rightarrow\; T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; T{*}(E{+}\underline{F})\dashv \;\Rightarrow\; \cdots$$

Equivalence of CFG and PDA



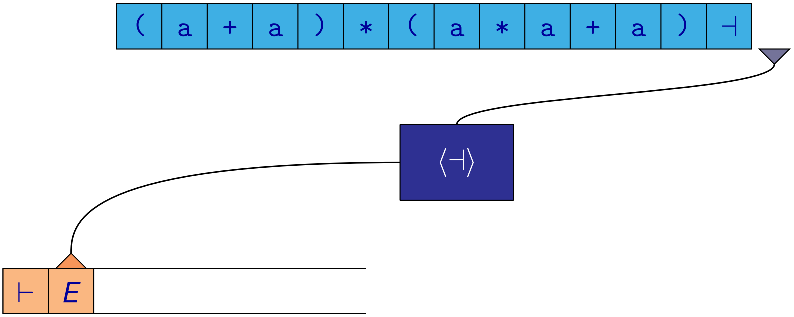
$\underline{E}\dashv \;\Rightarrow\; \underline{T}\dashv \;\Rightarrow\; T*\underline{F}\dashv \;\Rightarrow\; T*(\underline{E})\dashv \;\Rightarrow\; T*(E+\underline{T})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



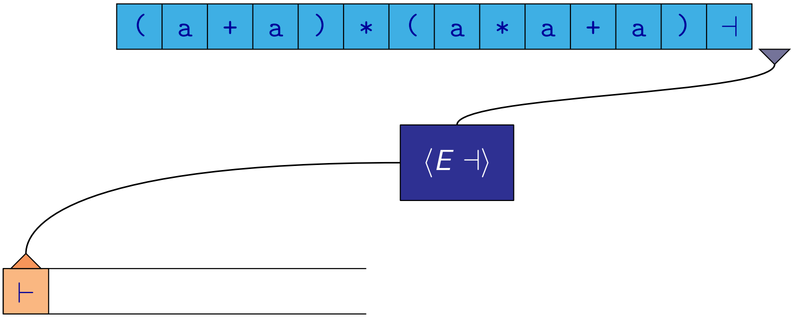
$\underline{E}\dashv \;\Rightarrow\; \underline{T}\dashv \;\Rightarrow\; T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



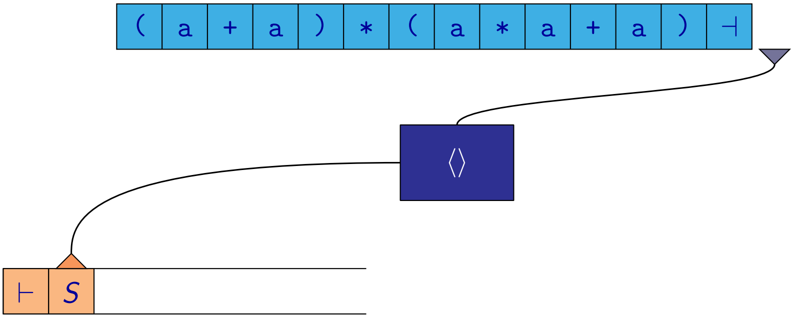
$\underline{E}\dashv \;\Rightarrow\; \underline{T}\dashv \;\Rightarrow\; T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



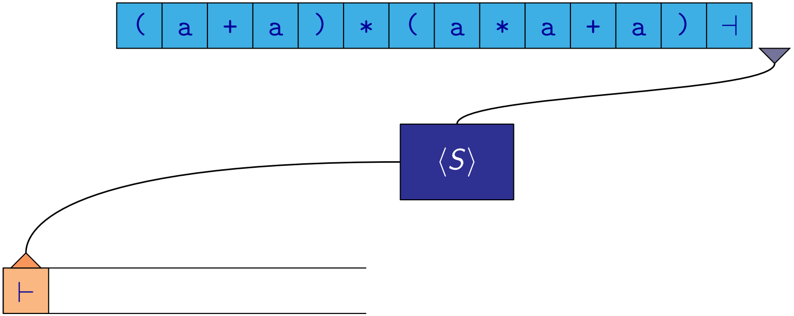
$\underline{E}\dashv \;\Rightarrow\; \underline{T}\dashv \;\Rightarrow\; T{*}\underline{F}\dashv \;\Rightarrow\; T{*}(\underline{E})\dashv \;\Rightarrow\; T{*}(E{+}\underline{T})\dashv \;\Rightarrow\; \cdots$

Equivalence of CFG and PDA



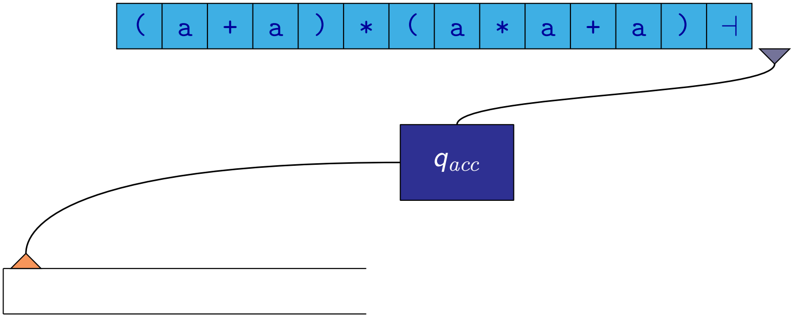
$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow T{*}\underline{F}\dashv \Rightarrow T{*}(\underline{E})\dashv \Rightarrow T{*}(E{+}\underline{T})\dashv \Rightarrow \cdots$

Equivalence of CFG and PDA



$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow T{*}\underline{F}\dashv \Rightarrow T{*}(\underline{E})\dashv \Rightarrow T{*}(E{+}\underline{T})\dashv \Rightarrow \cdots$

Equivalence of CFG and PDA



$\underline{S} \Rightarrow \underline{E}\dashv \Rightarrow \underline{T}\dashv \Rightarrow T*\underline{F}\dashv \Rightarrow T*(\underline{E})\dashv \Rightarrow T*(E+\underline{T})\dashv \Rightarrow \cdots$

Equivalence of CFG and PDA

As we can see from the previous example, the pushdown automaton $\mathcal{M}$ basically performs a **right derivation** in grammar $\mathcal{G}$ in reverse order.

Other Classes of Context-Free Grammars

There exist a lot of different classes of context-free grammars, for which it is possible to construct a corresponding pushdown automaton in such a way that this automaton is deterministic:

- **Top-down** approach — constructs a left derivation:
  - LL(0), LL(1), LL(2), . . .
- **Bottom-up** approach — constructs a right derivation in a reverse order:
  - LR(0), LR(1), LR(2), . . .
  - LALR (resp. LALR(1), . . .)
  - SLR (resp. SLR(1), . . .)

Parser Generators

Parser generators — tools that allow for a description of a context-free grammar to automatically generate a code in some programming language basically implementing behaviour of a corresponding pushdown automaton.

Examples of parser generators:

- Yacc
- Bison
- ANTLR
- JavaCC
- Menhir
- ...

Equivalence of CFG and PDA

Theorem

For every pushdown automaton $\mathcal{M}$ with one control state, there is a corresponding CFG $\mathcal{G}$ such $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{M})$.

Proof: For PDA $\mathcal{M} = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, X_0)$, where $\Sigma \cap \Gamma = \emptyset$, we construct CFG $\mathcal{G} = (\Gamma, \Sigma, X_0, P)$, where

$$(A \rightarrow a\alpha) \in P \qquad \text{iff} \qquad (q_0, \alpha) \in \delta(q_0, a, A)$$

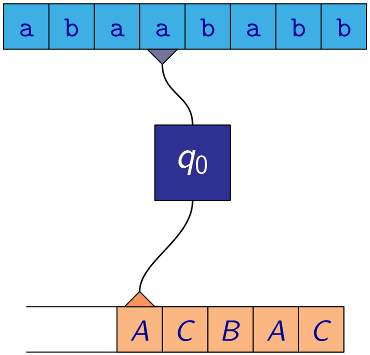
for all $A \in \Gamma$, $a \in \Sigma \cup \{\varepsilon\}$, and $\alpha \in \Gamma^*$.

It can be proved by induction that

$$X_0 \Rightarrow^* u\alpha \ \ (\text{in } \mathcal{G}) \qquad \text{iff} \qquad q_0 X_0 \xrightarrow{u} q_0\alpha \ \ (\text{in } \mathcal{M})$$

where $u \in \Sigma^*$ and $\alpha \in \Gamma^*$ (in $\mathcal{G}$, we consider only left derivations).

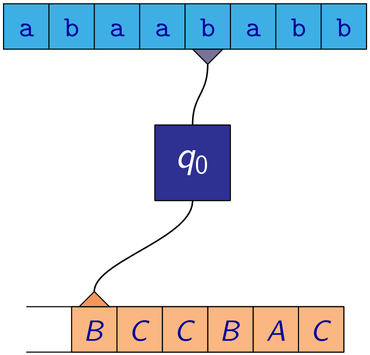
Equivalence of CFG and PDA



$\mathcal{M}$:	$\mathcal{G}$:
$\vdots$	$\vdots$
$q_0 A \xrightarrow{a} q_0 BC$	$A \longrightarrow aBC$
$q_0 B \xrightarrow{b} q_0$	$B \longrightarrow b$
$\vdots$	$\vdots$

a b a $\underline{A}\, C\, B\, A\, C$

Equivalence of CFG and PDA



$\mathcal{M}$:

$\vdots$

$q_0 A \xrightarrow{a} q_0 BC$

$q_0 B \xrightarrow{b} q_0$

$\vdots$

$\mathcal{G}$:

$\vdots$

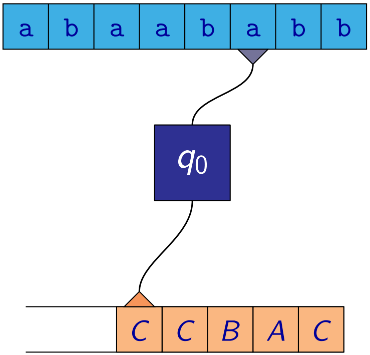
$A \rightarrow aBC$

$B \rightarrow b$

$\vdots$

$$\mathtt{a\,b\,a}\,\underline{A}\,C\,B\,A\,C$$
$$\Rightarrow\ \mathtt{a\,b\,a\,a}\,\underline{B}\,C\,C\,B\,A\,C$$

Equivalence of CFG and PDA



$\mathcal{M}$:

$\vdots$

$q_0 A \xrightarrow{a} q_0 BC$

$q_0 B \xrightarrow{b} q_0$

$\vdots$

$\mathcal{G}$:

$\vdots$

$A \longrightarrow aBC$

$B \longrightarrow b$

$\vdots$

$$\begin{aligned} & \mathtt{a\,b\,a}\, \underline{A}\, C\, B\, A\, C \\ \Rightarrow\ & \mathtt{a\,b\,a\,a}\, \underline{B}\, C\, C\, B\, A\, C \\ \Rightarrow\ & \mathtt{a\,b\,a\,a\,b}\, \underline{C}\, C\, B\, A\, C \end{aligned}$$

Equivalence of CFG and PDA

Theorem

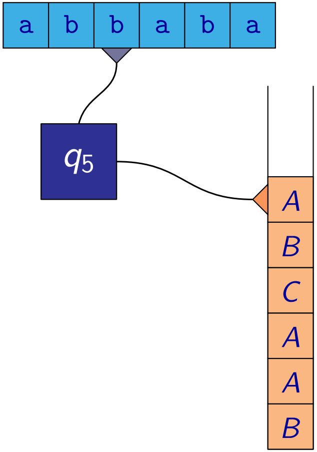
For every pushdown automaton $\mathcal{M}$ there exists a pushdown automaton $\mathcal{M}'$ with one control state such that $\mathcal{L}(\mathcal{M}') = \mathcal{L}(\mathcal{M})$.

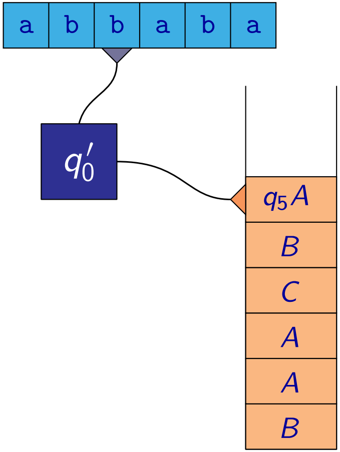
Proof idea:

- The control state of $\mathcal{M}$ is stored on the top of the stack of $\mathcal{M}'$.
- For $\delta(q, a, X) = \{(q', \varepsilon)\}$ we must ensure that the new control state on the stack of $\mathcal{M}'$ is q'. (Other cases are straightforward.)
- Stack symbols of $\mathcal{M}'$ are triples of the form (q, A, q') where q represents the control state of $\mathcal{M}$ when that symbol is on the top, A is the stack symbol of $\mathcal{M}$, and q' is the first control state in the triple below it.
- PDA $\mathcal{M}'$ nondeterministically "guesses" the control states to which $\mathcal{M}$ goes when the given stack symbols becomes the top of the stack.

Equivalence of CFG and PDA

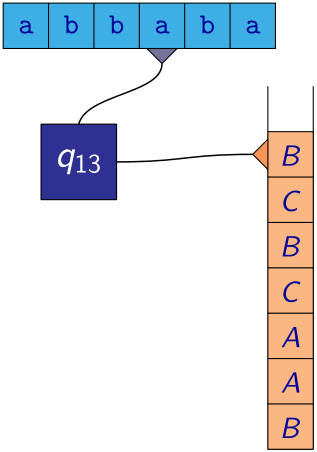
Incorrect idea:

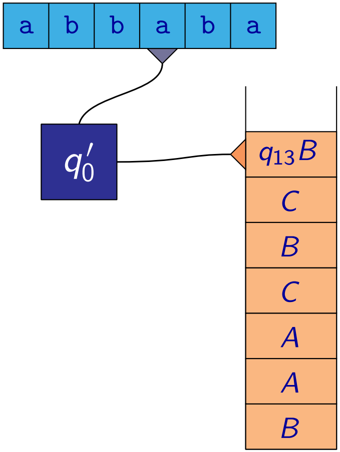




Equivalence of CFG and PDA

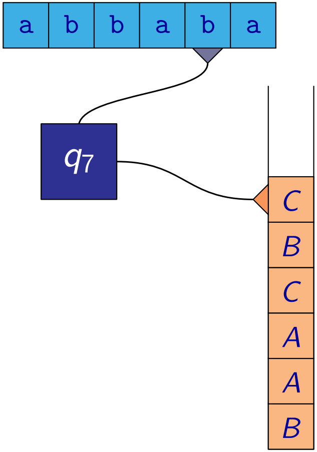
Incorrect idea:

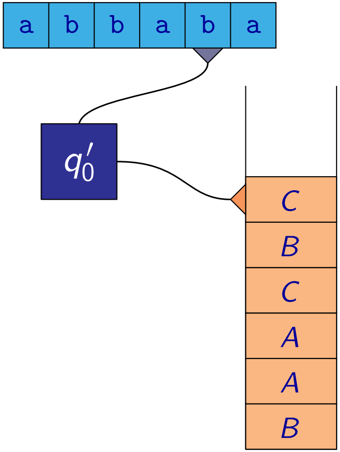




Equivalence of CFG and PDA

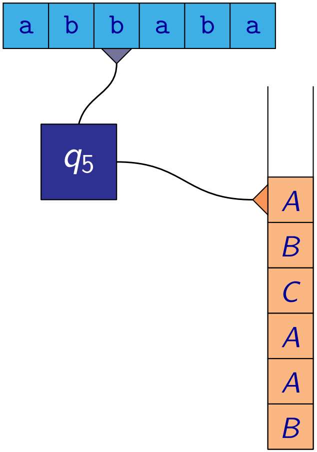
Incorrect idea:

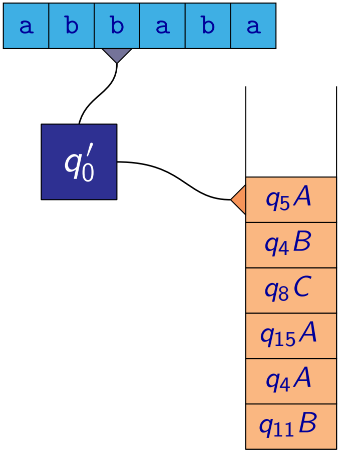




Equivalence of CFG and PDA

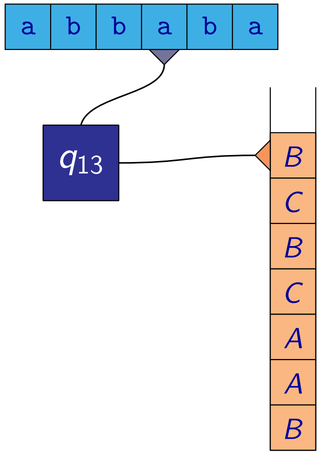
Other incorrect idea:

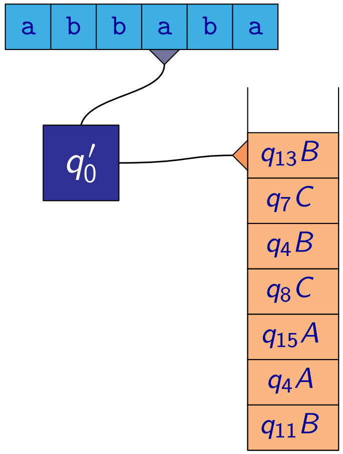




Equivalence of CFG and PDA

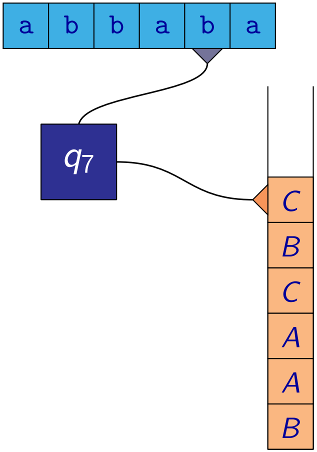
Other incorrect idea:

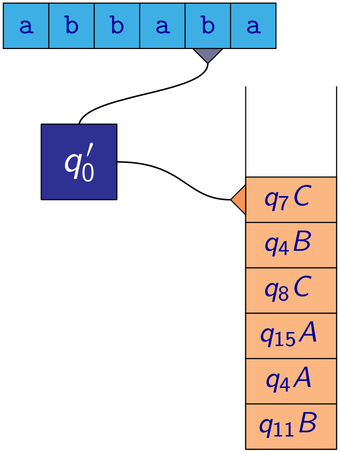




Equivalence of CFG and PDA

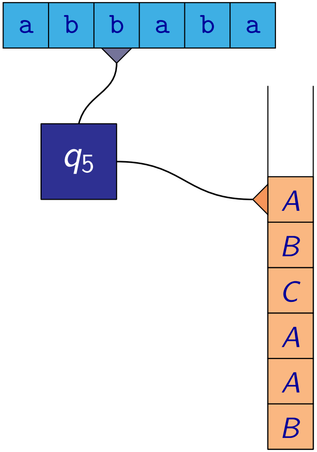
Other incorrect idea:

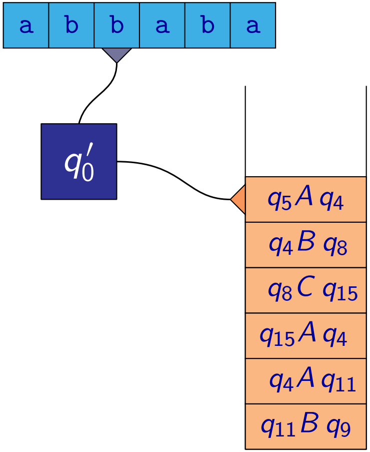




Equivalence of CFG and PDA

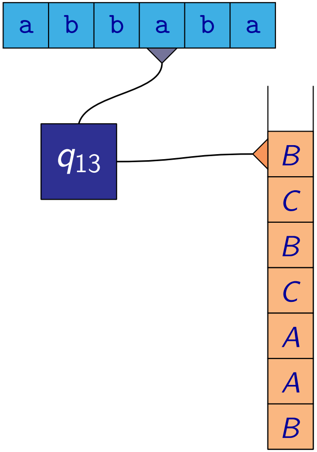
The correct construction:

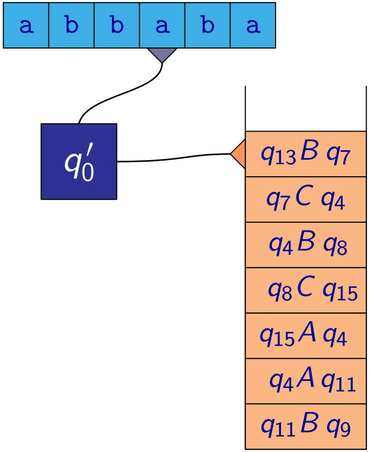




Equivalence of CFG and PDA

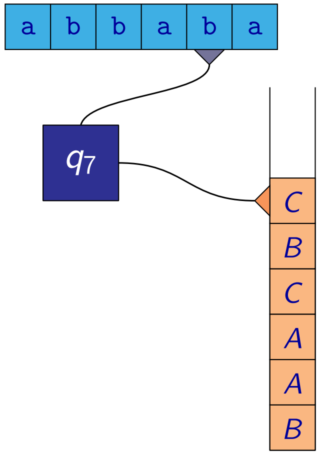
The correct construction:

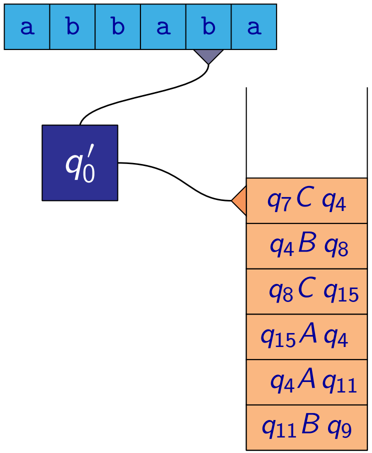




Equivalence of CFG and PDA

The correct construction:





Equivalence of CFG and PDA

Proposition

For every context-free grammar $\mathcal{G}$ there is some (nondeterministic) pushdown automaton $\mathcal{M}$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{M})$.

Proposition

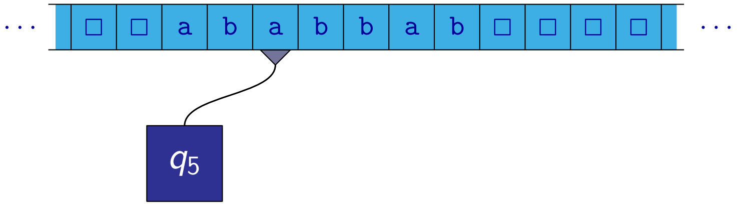
For every pushdown automaton $\mathcal{M}$ there is some context-free grammar $\mathcal{G}$ such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{G})$.

Turing Machines

Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

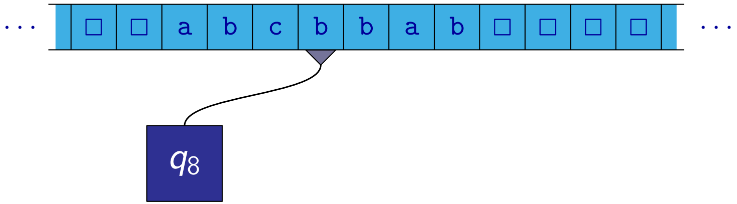
- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

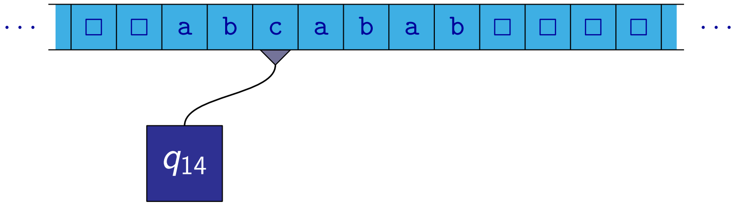
- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

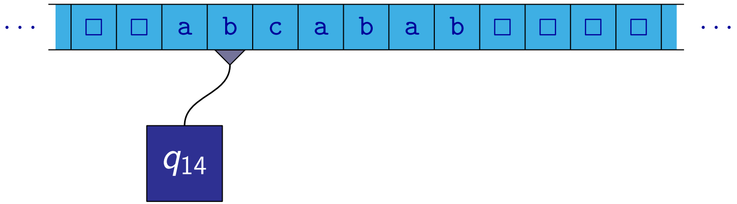
- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

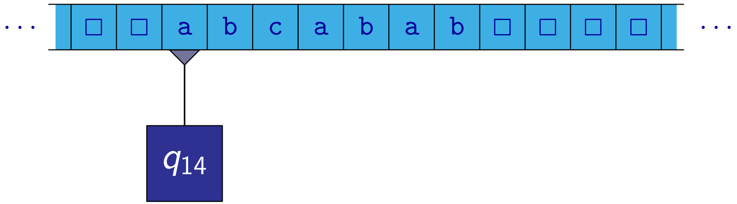
- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

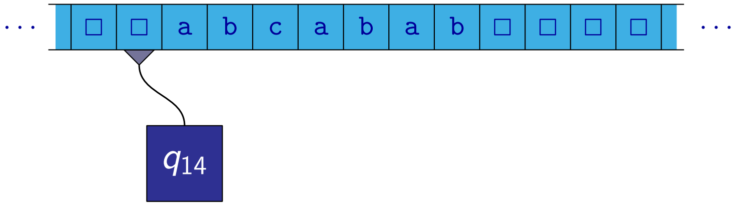
- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

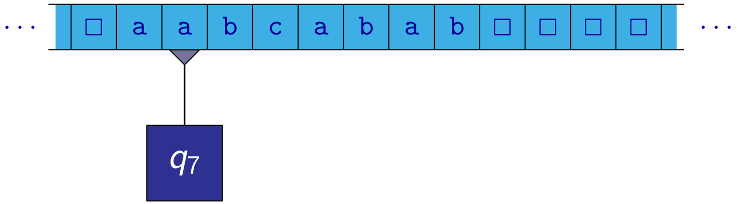
- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Turing machine — a device similar to a finite automaton with the following differences:

- the head can move in both directions
- it is possible to write on a current position of the head
- the tape is infinite



Turing Machine

Alan M. Turing, “On Computable Numbers, with an application to the Entscheidungsproblem”, *Proceedings of the London Mathematical Society*, 42 (1936), pp. 230–265, Erratum: Ibid., 43 (1937), pp. 544–546.

Turing Machine

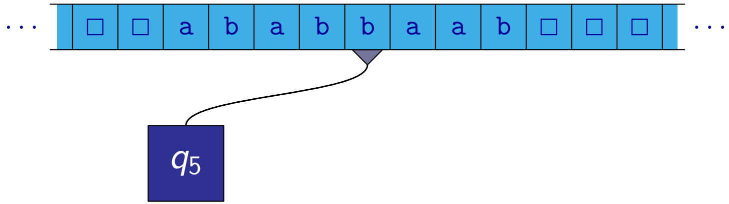
Definition

Formally, **Turing machine** is defined as a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite non-empty set of **states**
- Γ is a finite (non-empty) set of **tape symbols** (**tape alphabet**)
- $\Sigma \subseteq \Gamma$ is a finite non-empty set of **input symbols** (**input alphabet**)
- $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ is a **transition function**
- $q_0 \in Q$ is an **initial state**
- $F \subseteq Q$ is a set of **final states**

We assume that $\Gamma - \Sigma$ always contains a special element $\square$ denoting a **blank** symbol.

Configurations of a Turing Machine

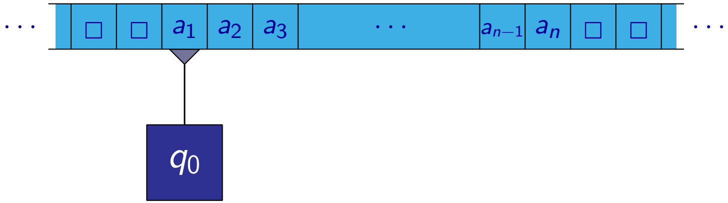


A configuration of a Turing machine is given by:

- a state of its control unit
- a content of the tape
- a position of the head

Configurations of a Turing Machine

A computation of a Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ over a word $w \in \Sigma^*$,
where $w = a_1 a_2 \cdots a_n$, starts in an **initial configuration**:



- the state of the control unit is q_0
- word w is written on the tape, remaining cells of the tape are filled with the blank symbols ($\square$)
- the head is on the first symbol of the word w (or on symbol $\square$ when $w = \varepsilon$)

Turing Machine

One step of a Turing machine:

Let us assume that:

- the state of the control unit is q
- the cell of the tape on the position of the head contains symbol b

Let us say that $\delta(q, b) = (q', b', d)$ where $d \in \{-1, 0, +1\}$.

One step of the Turing machine is performed as follows:

- the state of the control unit is changed to q'
- symbol b' is written on the tape cell on the position of the head instead of b
- The head is moved depending on d:
    - for $d = -1$ the head is moved one cell left
    - for $d = +1$ the head is moved one cell right
    - for $d = 0$ the position of the head is not changed

Turing Machine

- A Turing machine performs these steps until a state of its control unit is a state from the set F.
- Those configurations where a state of the control unit belongs to set F are **final configurations**.
- A computation ends in a final configuration.
- A computation of a machine $\mathcal{M}$ over a word w can be infinite.

Turing Machine

We often choose the set of final states $F = \{q_{acc}, q_{rej}\}$.

Then we can define for a word $w \in \Sigma^*$ if a given Turing machine accepts it:

- If the state of the control unit after the computation over the word w is q_{acc}, the machine accepts the word w.
- If the state of the control unit after the computation over the word w is q_{rej}, the machine does not accept the word w.
- The computation of the machine over the word w can be infinite. In this case the machine does not accept the word w.

The language $\mathcal{L}(\mathcal{M})$ of a Turing machine $\mathcal{M}$ is the set of all words accepted by $\mathcal{M}$.

Turing Machine

A language $L \subseteq \Sigma^*$ is **accepted** by a Turing machine $\mathcal{M}$ if:

- for each word $w \in \Sigma^*$ it holds that $w \in L$ iff the computation of $\mathcal{M}$ over w ends in final state q_{acc}.

(So computations over words that do not belong to L can end in state q_{rej} or be infinite.)

Language $L \subseteq \Sigma^*$ is **recognized** by a Turing machine $\mathcal{M}$ if:

- for each word $w \in L$ the computation of machine $\mathcal{M}$ over w ends in final state q_{acc}.
- for every word $w \in (\Sigma^* - L)$ the computation of machine $\mathcal{M}$ over w ends in final state q_{rej}.

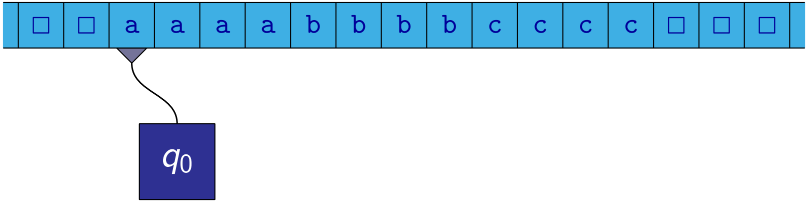
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\qquad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



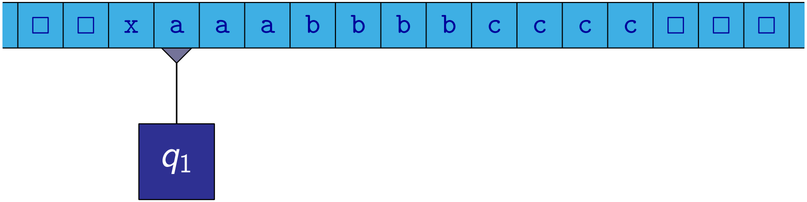
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\quad \Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



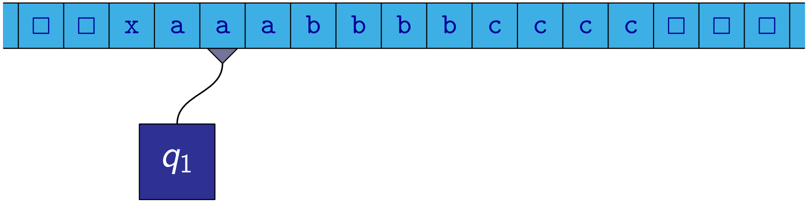
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



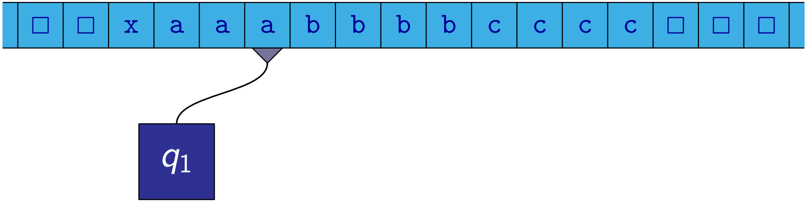
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{a, b, c\}$ $\Gamma = \{\square, a, b, c, x\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, x, +1)$	$(q_{rej}, b, 0)$	$(q_{rej}, c, 0)$	$(q_0, x, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, a, +1)$	$(q_2, x, +1)$	$(q_{rej}, c, 0)$	$(q_1, x, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, a, 0)$	$(q_2, b, +1)$	$(q_3, x, +1)$	$(q_2, x, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, a, 0)$	$(q_{rej}, b, 0)$	$(q_3, c, +1)$	$(q_3, x, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, a, -1)$	$(q_4, b, -1)$	$(q_4, c, -1)$	$(q_4, x, -1)$



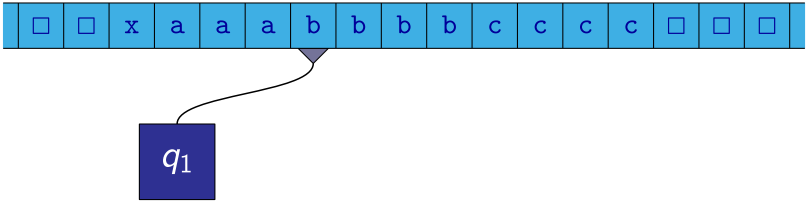
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



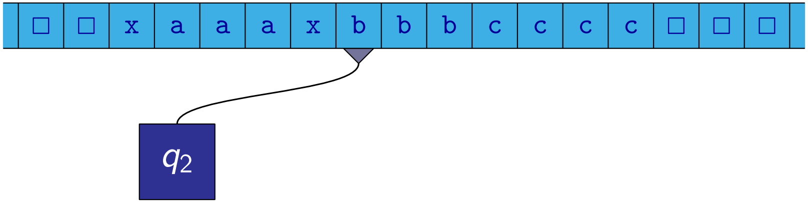
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



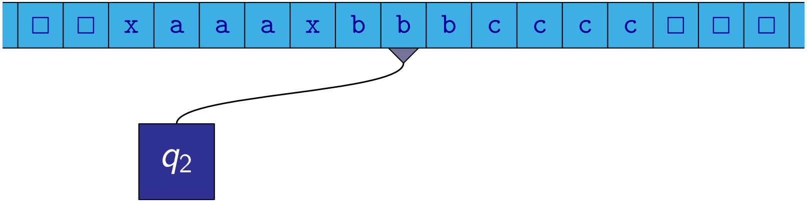
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\quad \Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



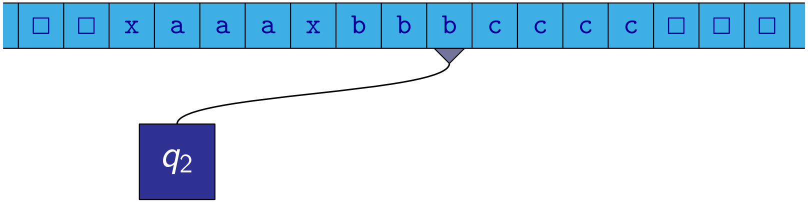
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



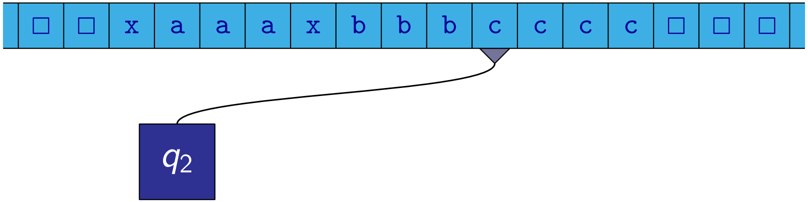
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



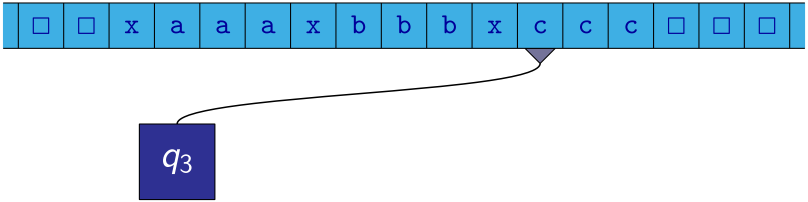
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



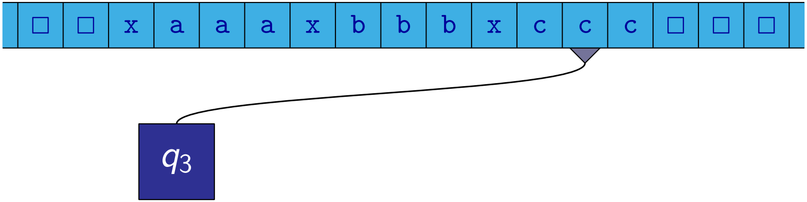
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\qquad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



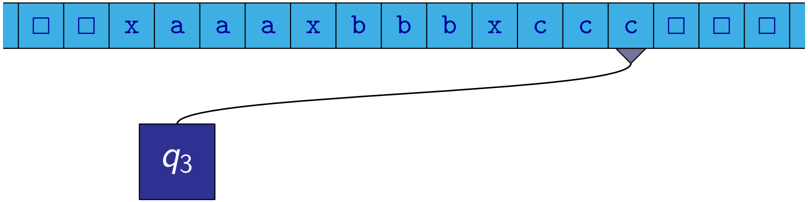
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



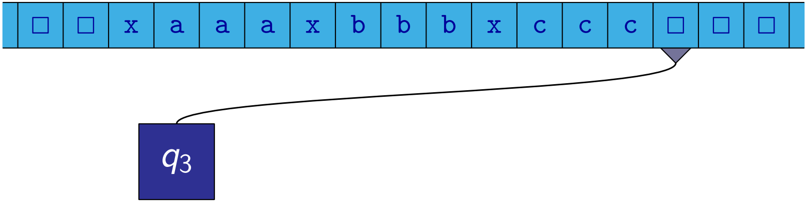
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



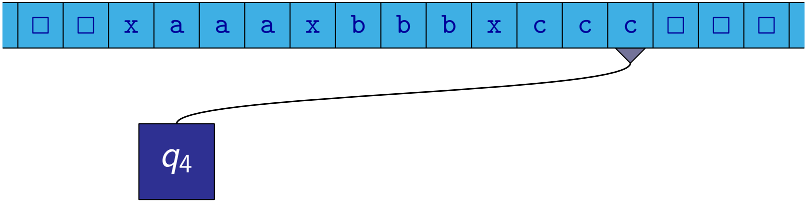
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



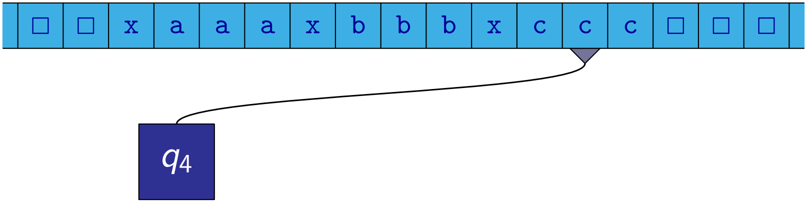
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\qquad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



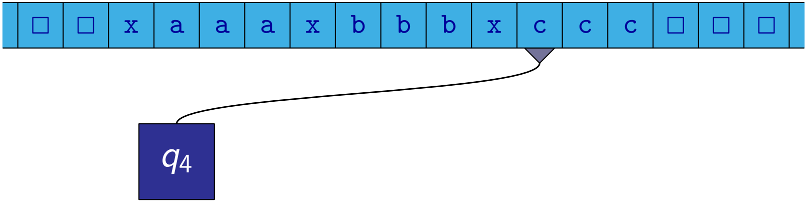
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



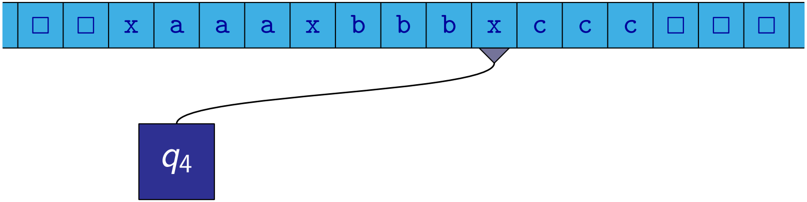
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



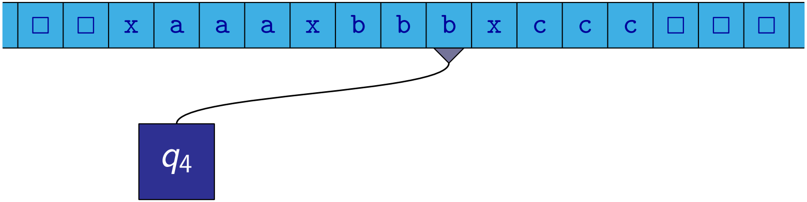
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



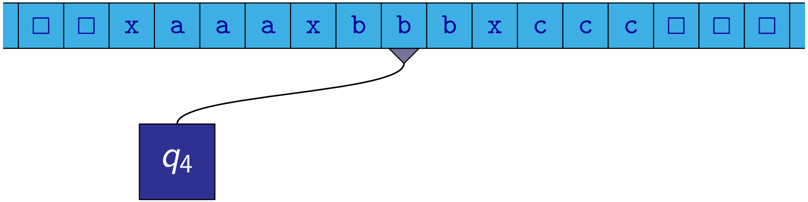
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



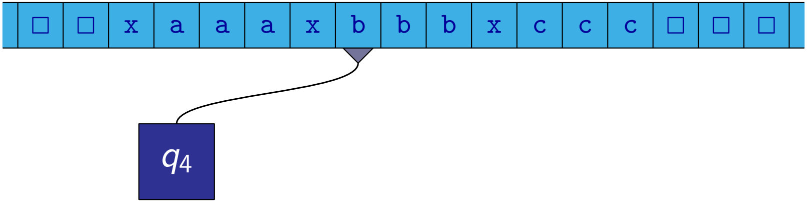
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



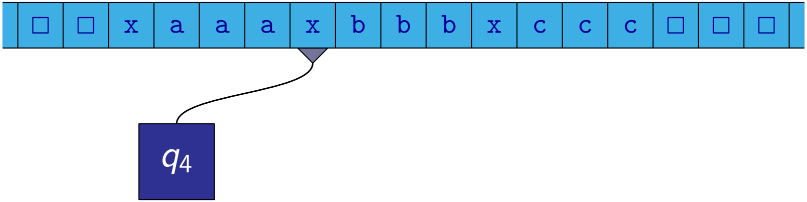
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



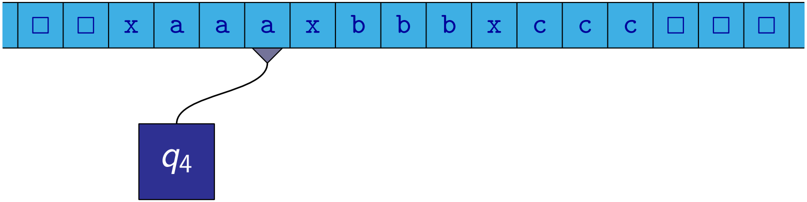
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



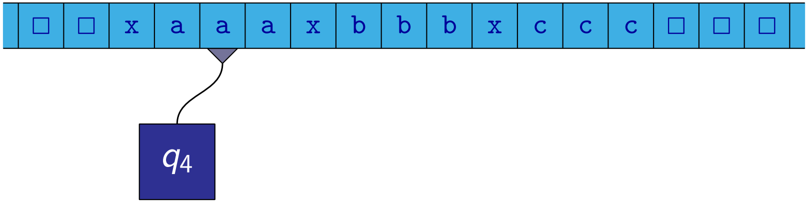
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



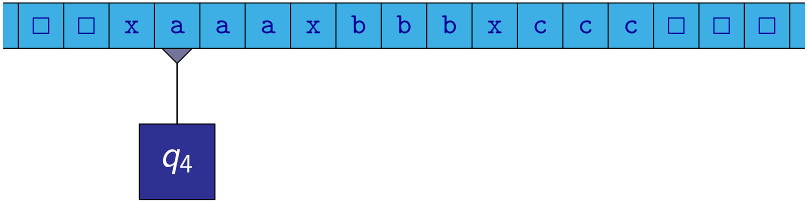
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\qquad \Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



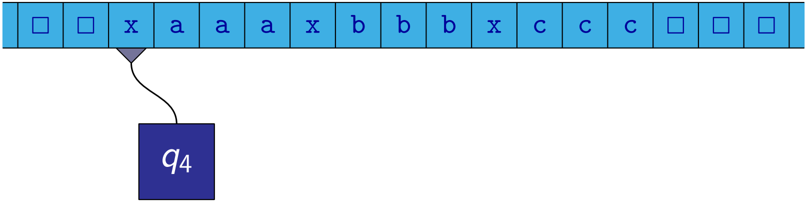
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



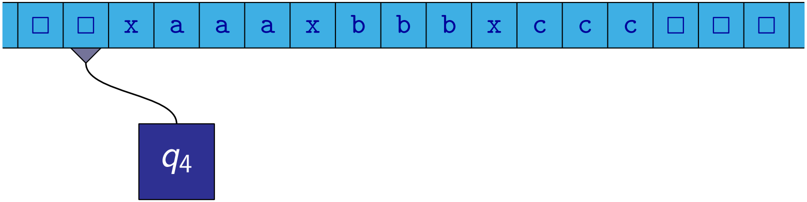
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



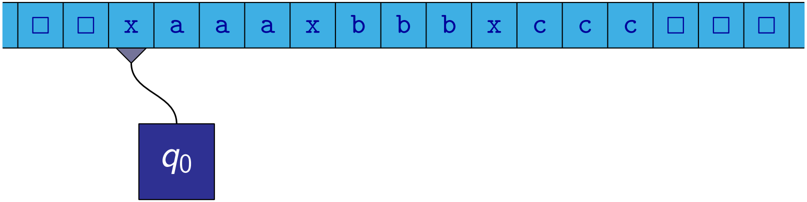
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



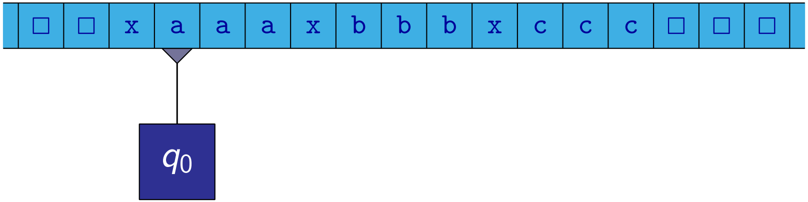
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



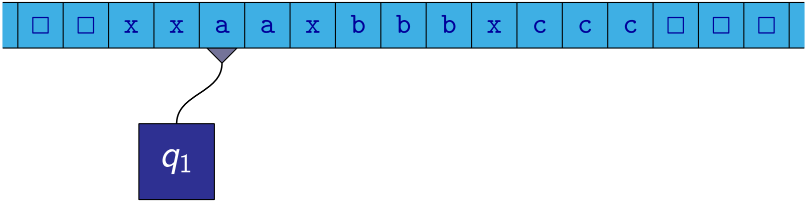
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



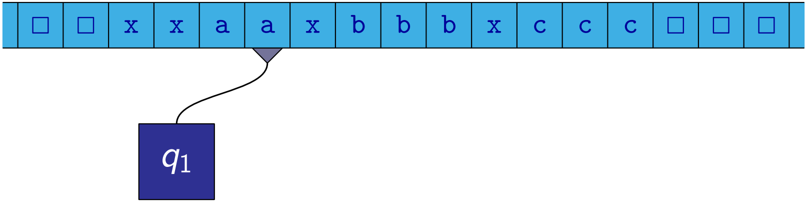
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



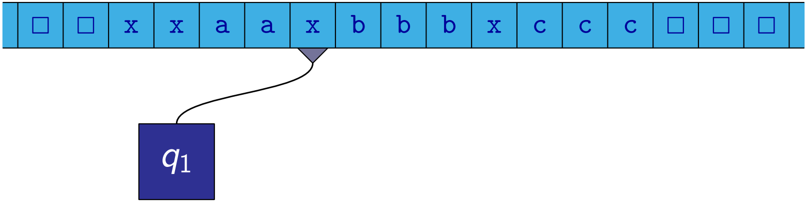
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



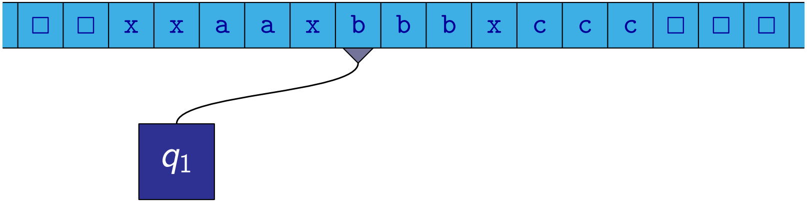
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



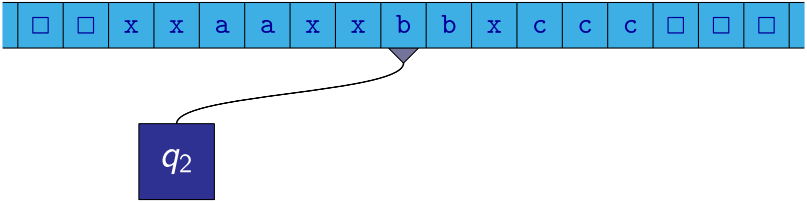
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



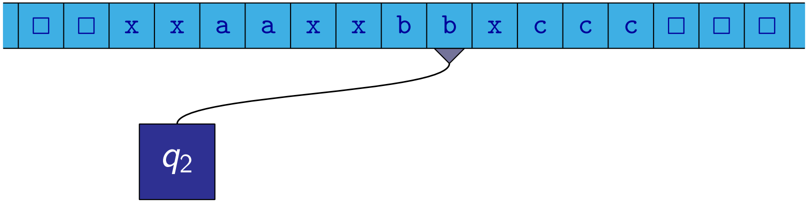
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



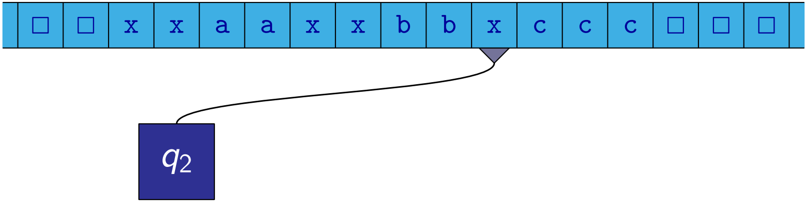
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



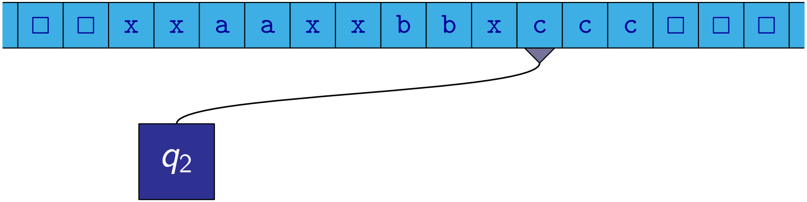
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\qquad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



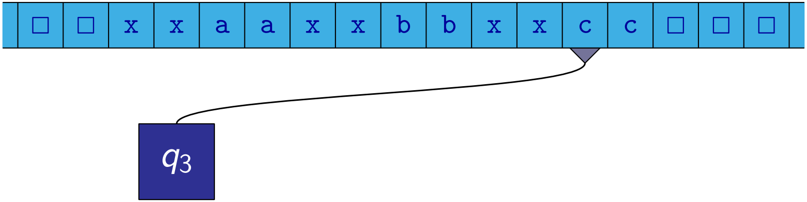
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



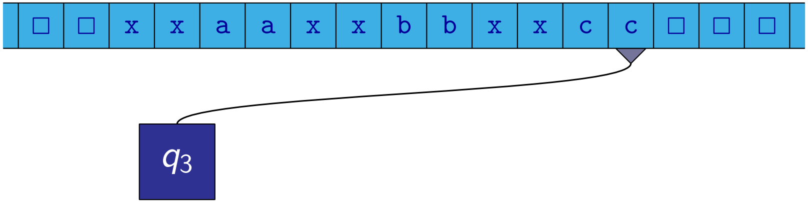
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



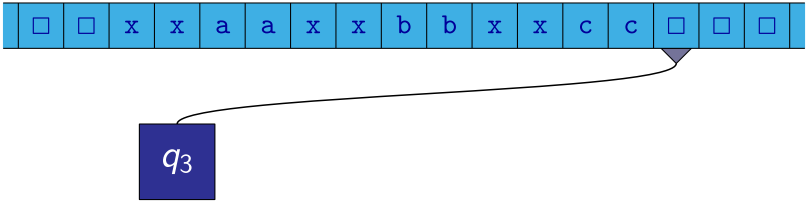
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



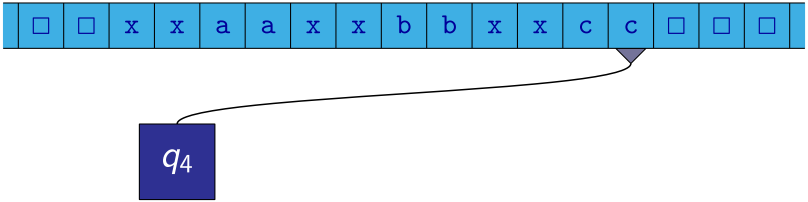
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



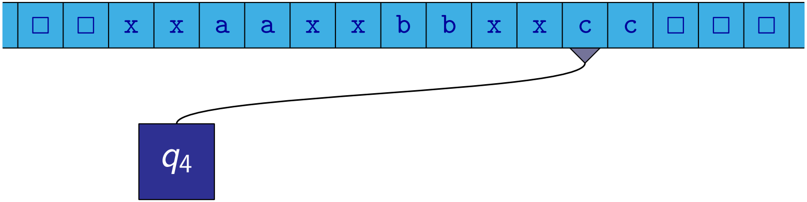
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\quad \Gamma = \{\Box, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



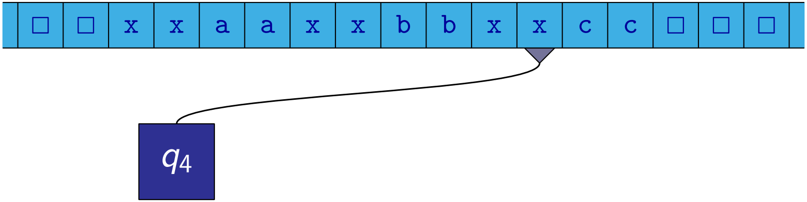
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



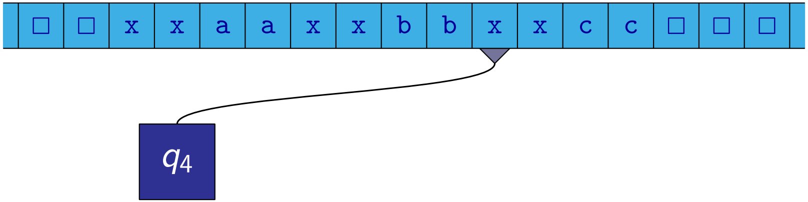
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



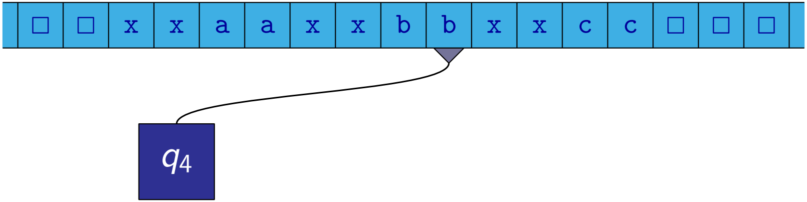
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



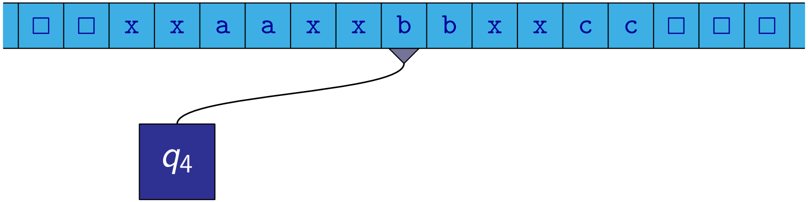
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



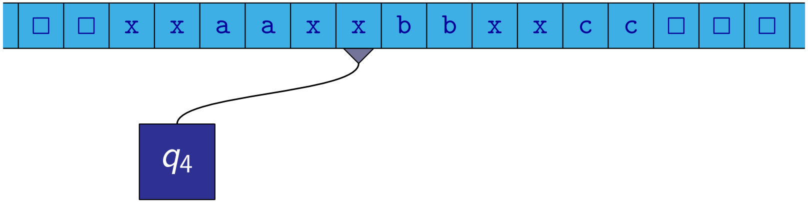
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



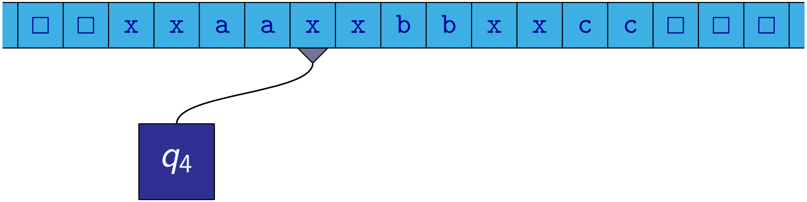
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



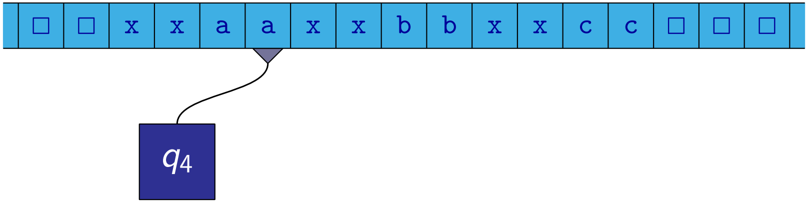
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



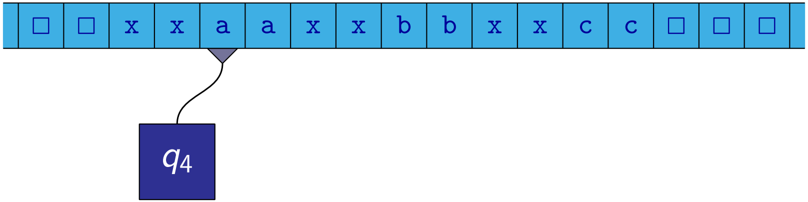
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



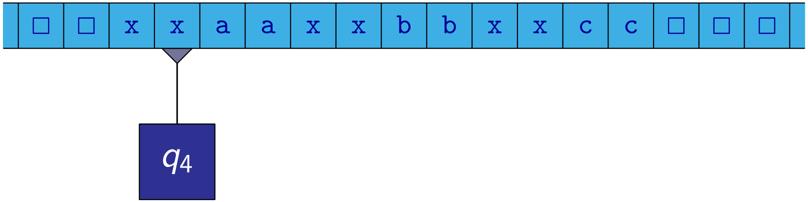
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



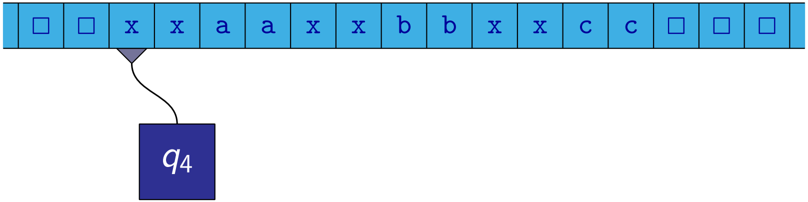
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



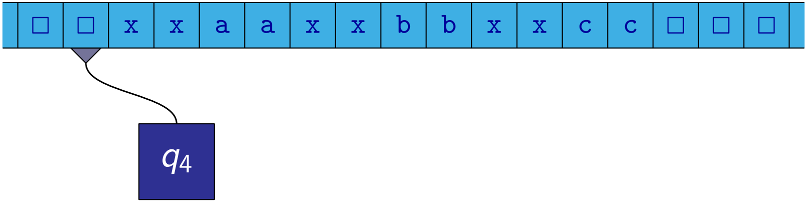
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



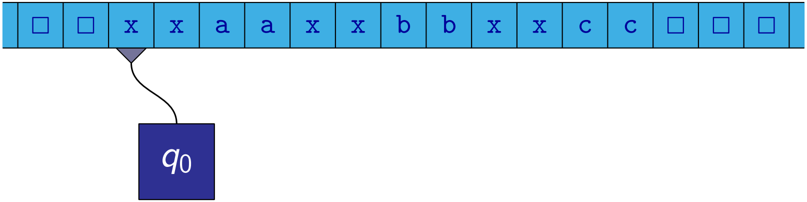
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



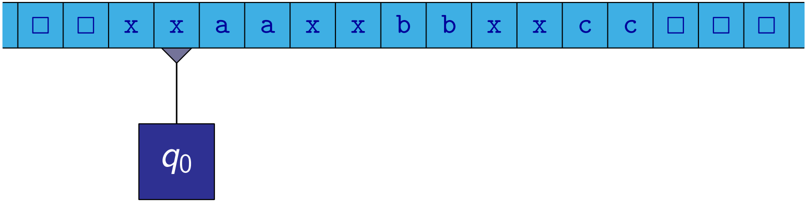
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



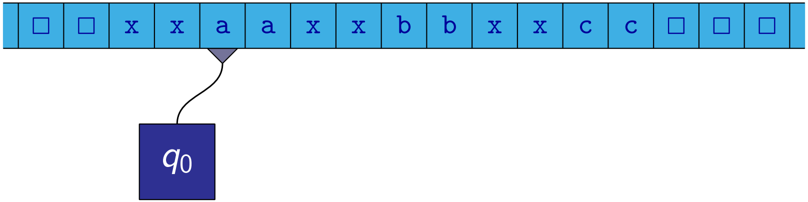
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



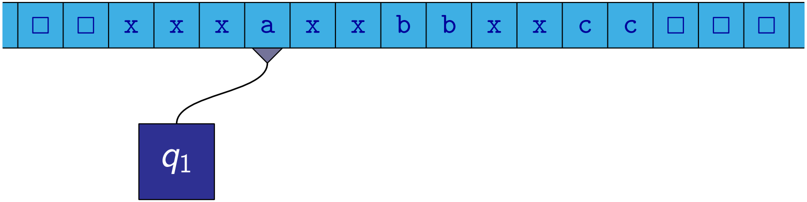
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



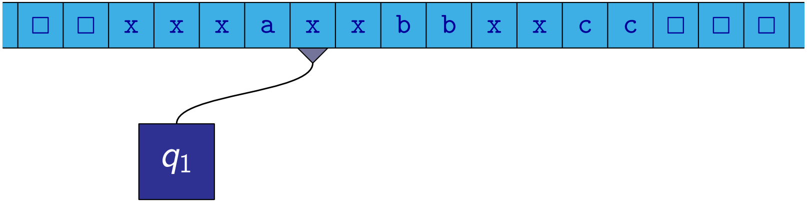
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



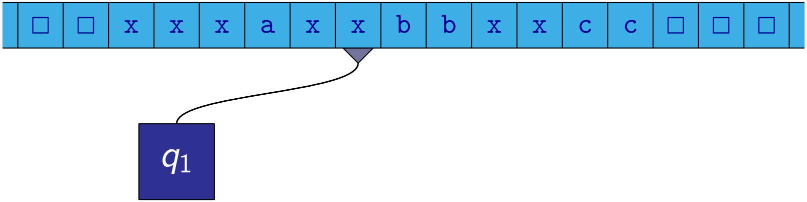
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



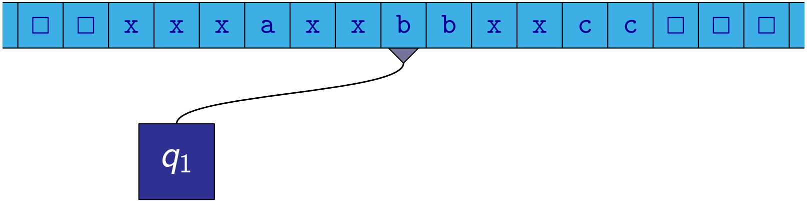
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



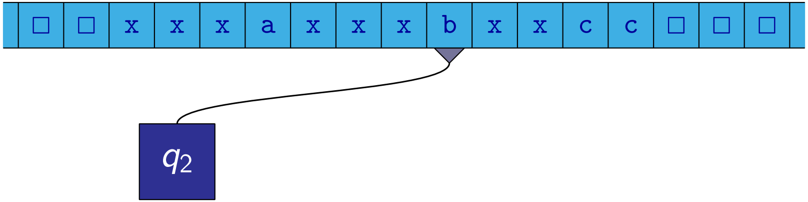
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\quad \Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



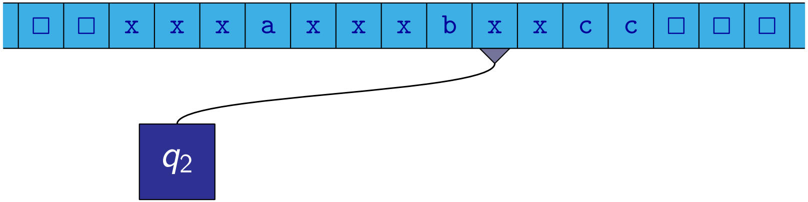
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



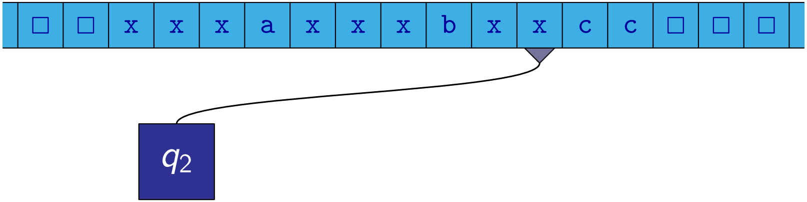
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



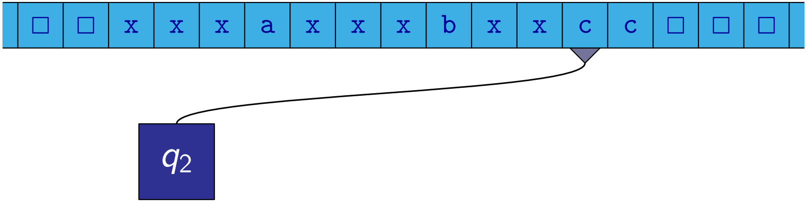
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



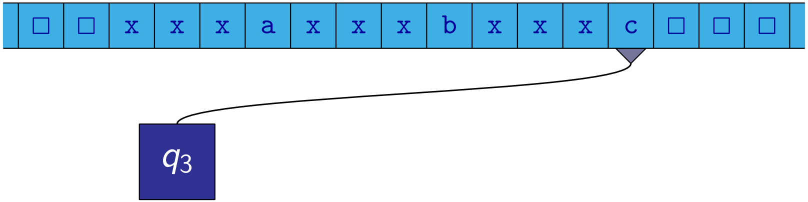
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



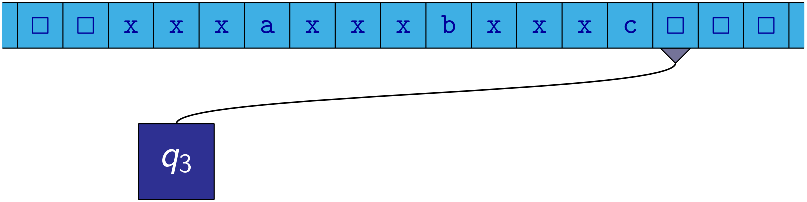
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



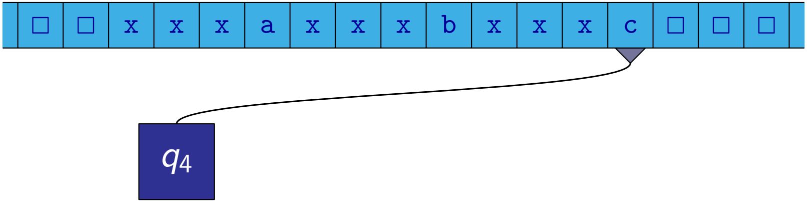
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



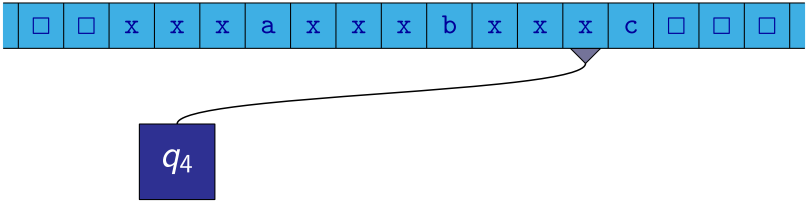
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\qquad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



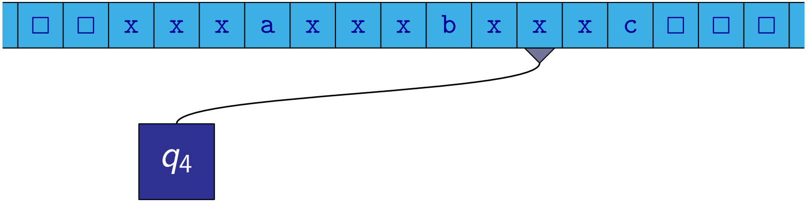
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\qquad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



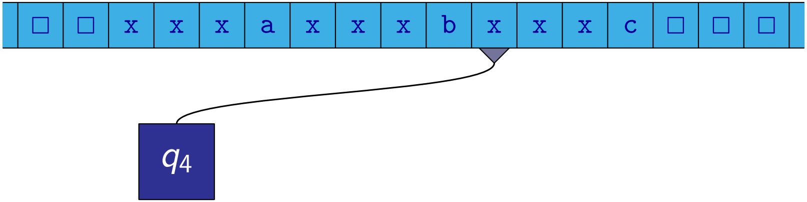
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



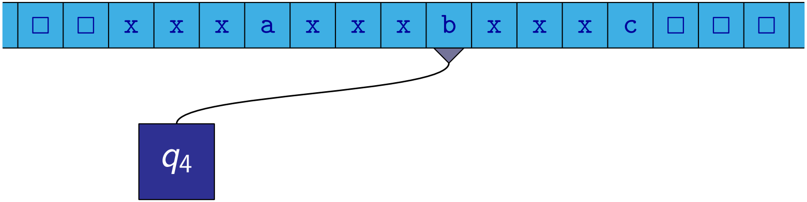
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



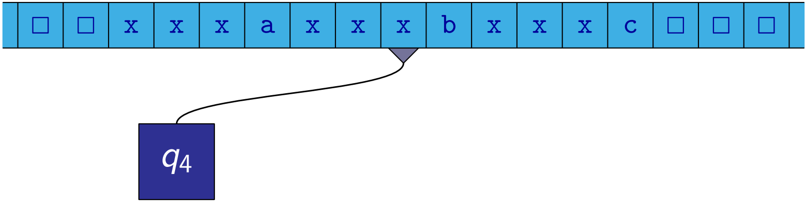
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



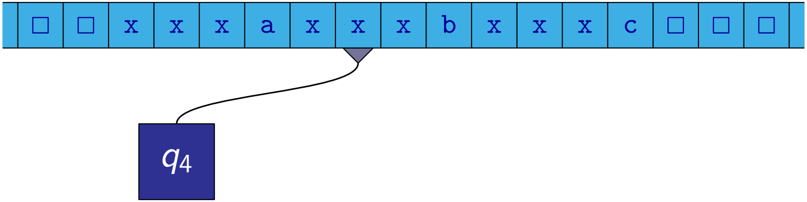
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



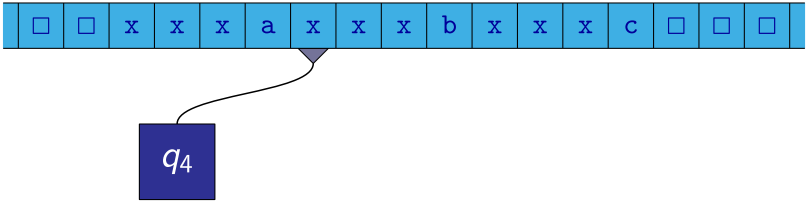
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



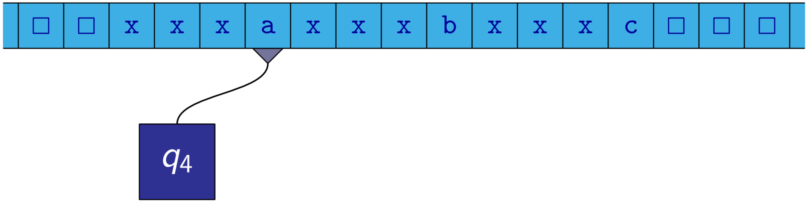
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



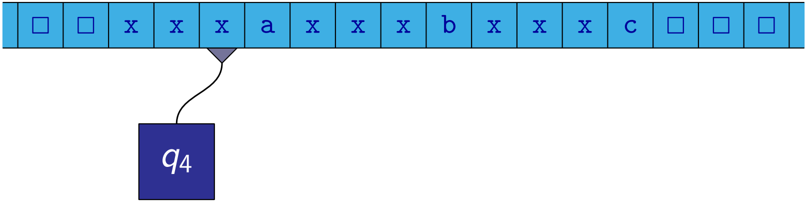
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



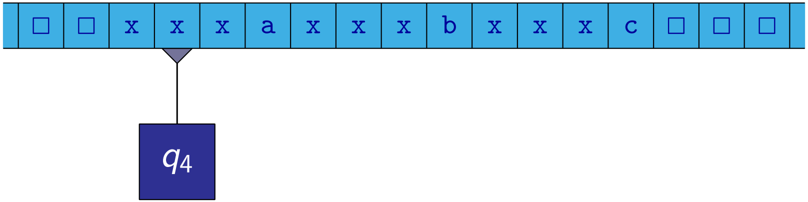
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



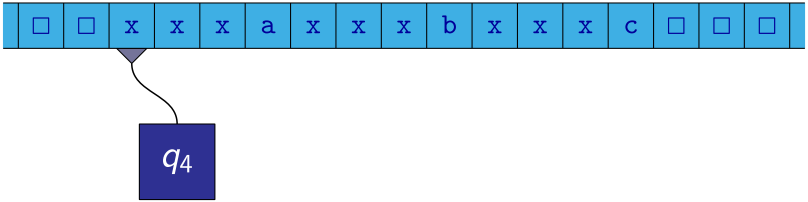
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



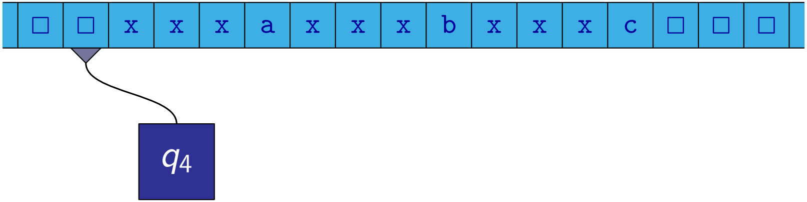
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



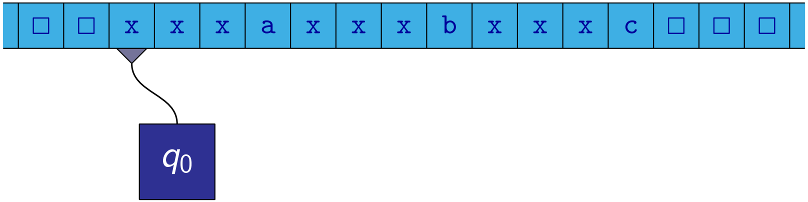
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\quad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



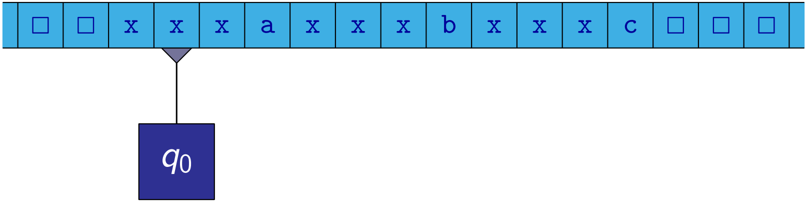
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



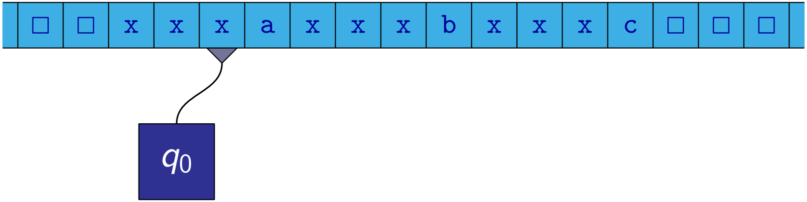
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\quad \Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



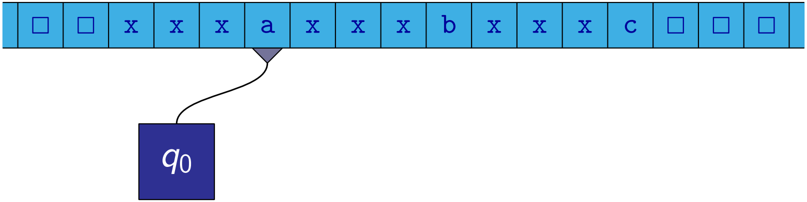
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



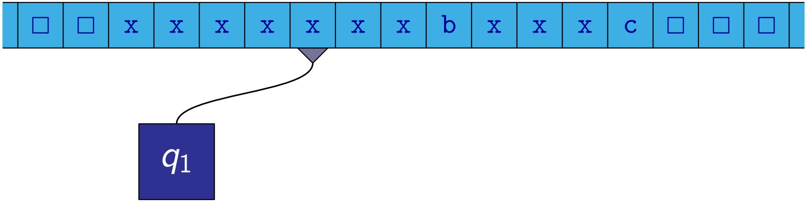
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



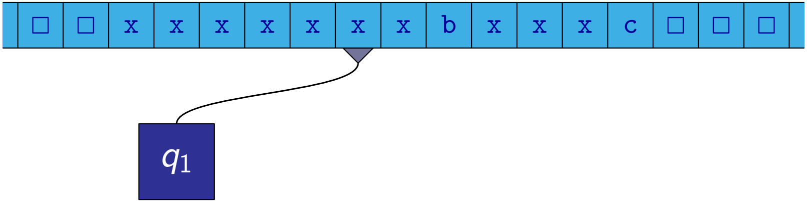
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



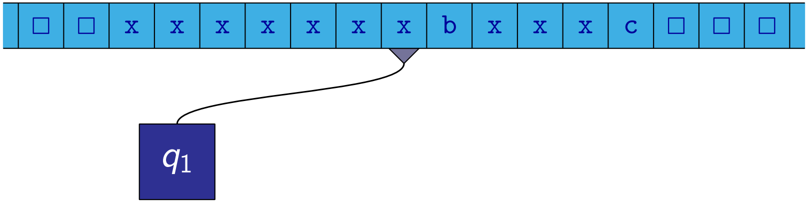
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



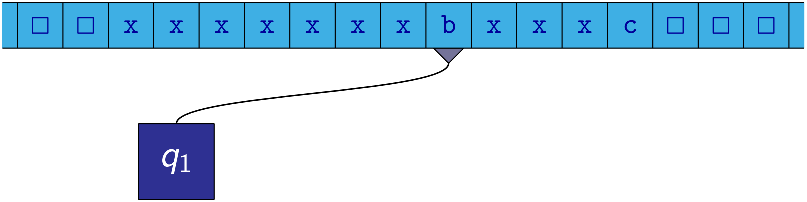
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



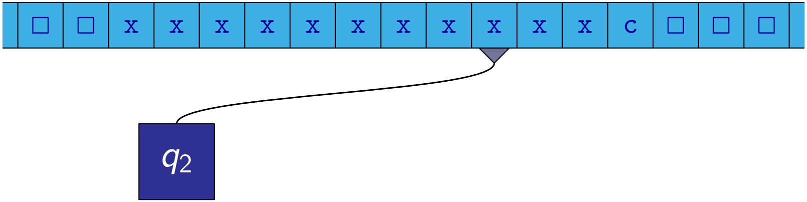
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\quad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



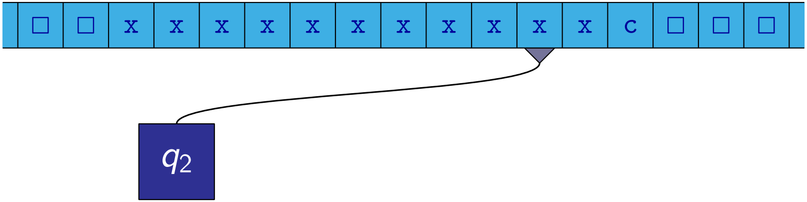
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



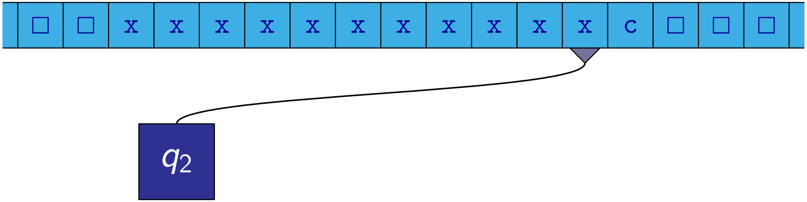
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\Box, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



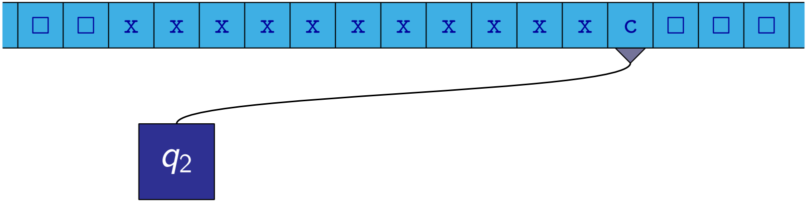
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{a, b, c\}$ $\Gamma = \{\square, a, b, c, x\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, x, +1)$	$(q_{rej}, b, 0)$	$(q_{rej}, c, 0)$	$(q_0, x, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, a, +1)$	$(q_2, x, +1)$	$(q_{rej}, c, 0)$	$(q_1, x, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, a, 0)$	$(q_2, b, +1)$	$(q_3, x, +1)$	$(q_2, x, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, a, 0)$	$(q_{rej}, b, 0)$	$(q_3, c, +1)$	$(q_3, x, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, a, -1)$	$(q_4, b, -1)$	$(q_4, c, -1)$	$(q_4, x, -1)$



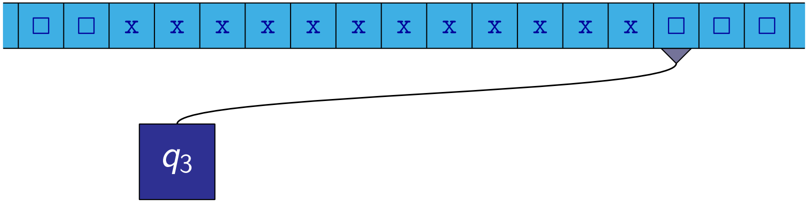
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\qquad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



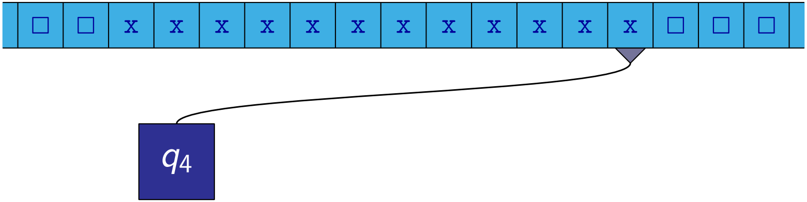
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\quad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



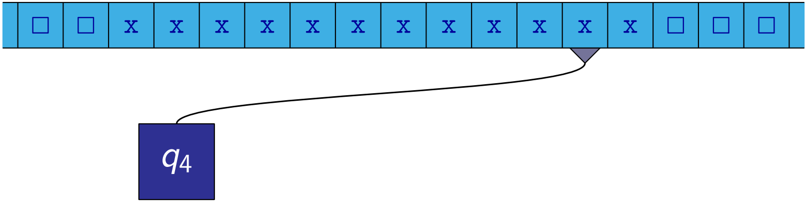
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



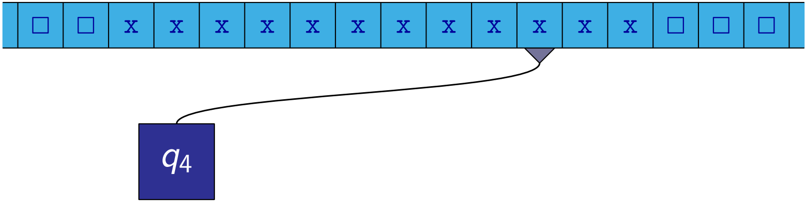
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\qquad \Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



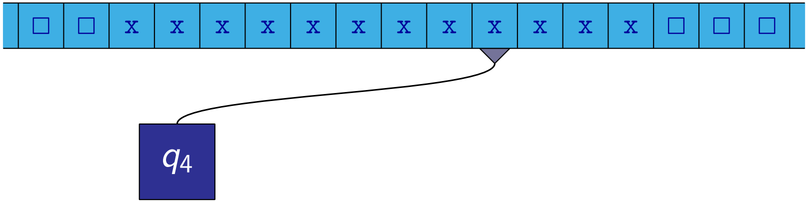
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



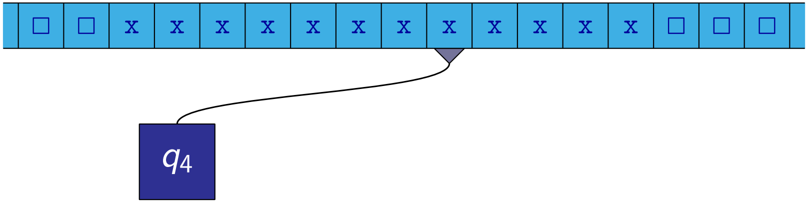
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



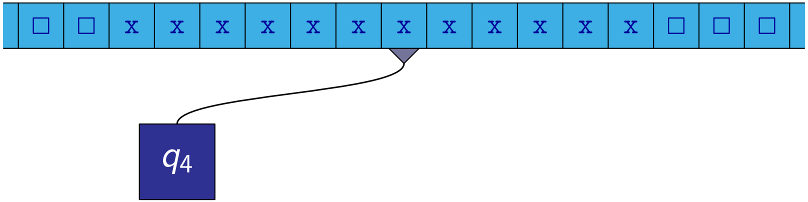
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



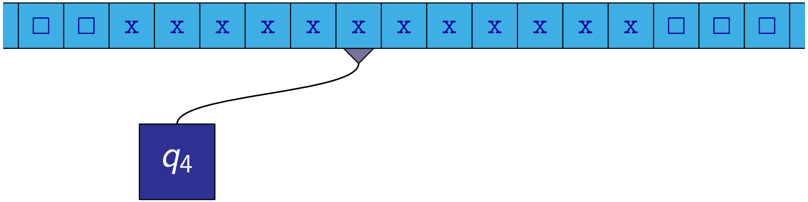
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



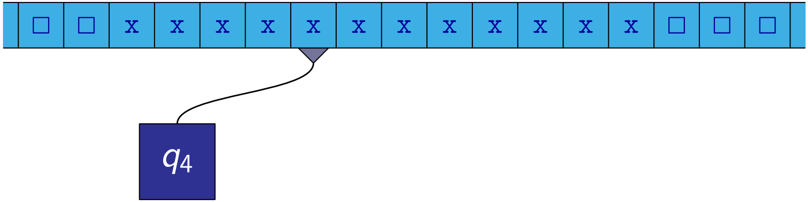
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



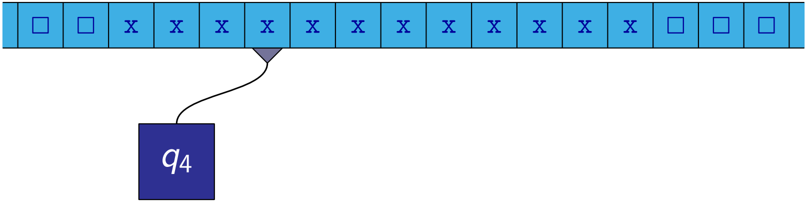
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



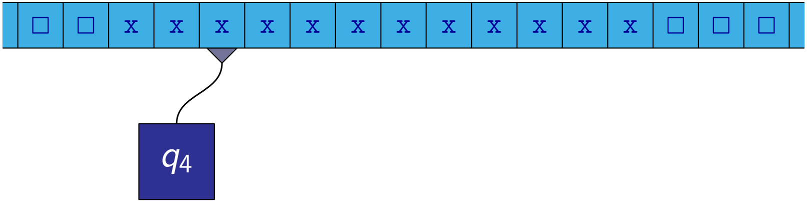
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



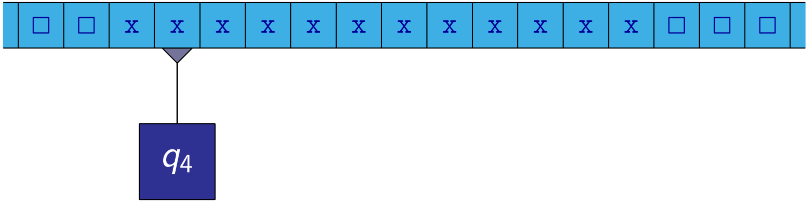
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



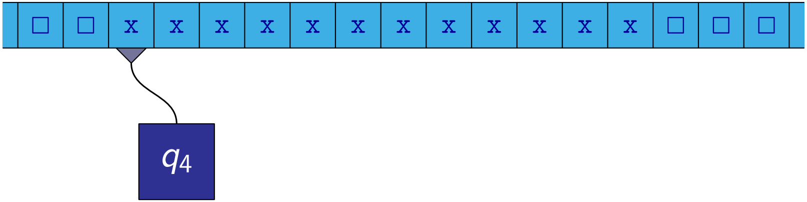
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\quad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



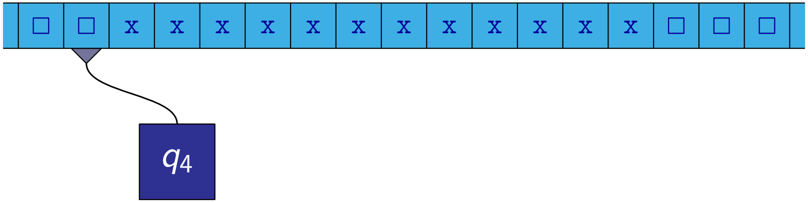
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



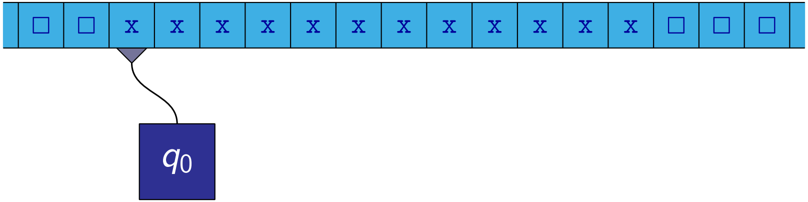
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\qquad \Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



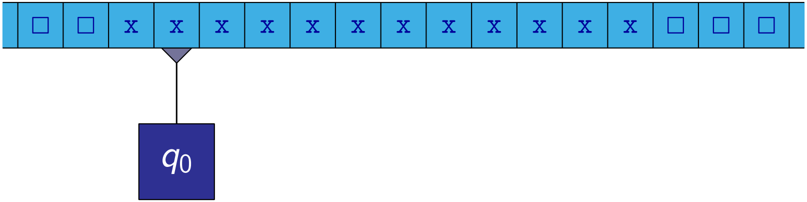
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\Box, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



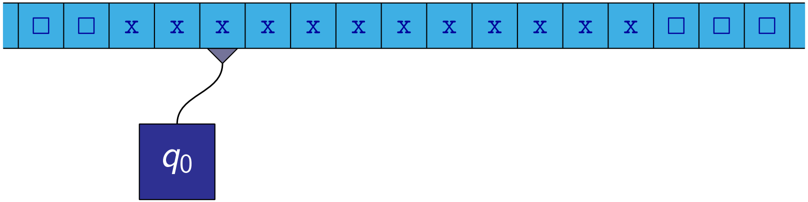
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



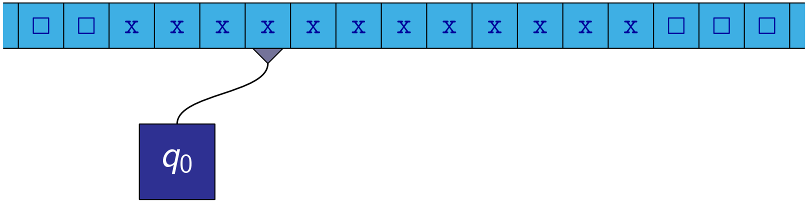
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



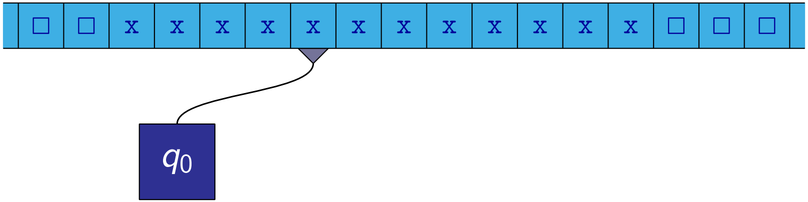
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



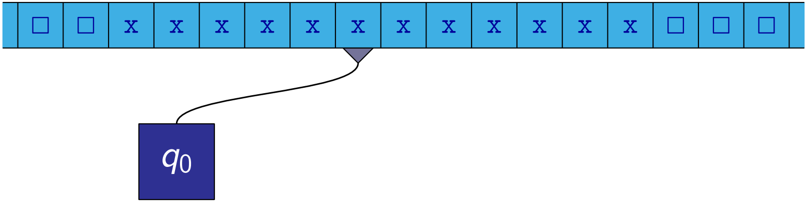
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\text{a}, \text{b}, \text{c}\}$ $\Gamma = \{\square, \text{a}, \text{b}, \text{c}, \text{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \text{x}, +1)$	$(q_{rej}, \text{b}, 0)$	$(q_{rej}, \text{c}, 0)$	$(q_0, \text{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \text{a}, +1)$	$(q_2, \text{x}, +1)$	$(q_{rej}, \text{c}, 0)$	$(q_1, \text{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \text{a}, 0)$	$(q_2, \text{b}, +1)$	$(q_3, \text{x}, +1)$	$(q_2, \text{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \text{a}, 0)$	$(q_{rej}, \text{b}, 0)$	$(q_3, \text{c}, +1)$	$(q_3, \text{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \text{a}, -1)$	$(q_4, \text{b}, -1)$	$(q_4, \text{c}, -1)$	$(q_4, \text{x}, -1)$



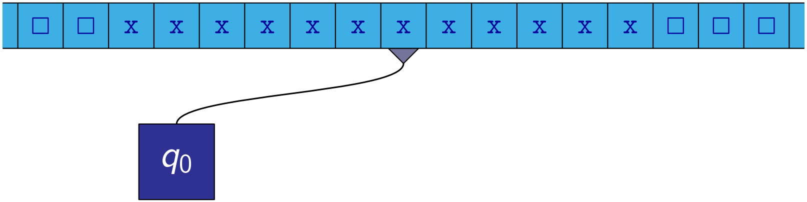
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



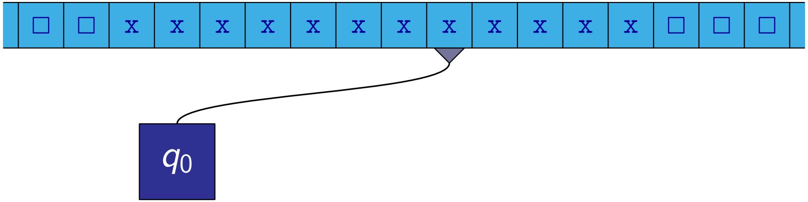
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



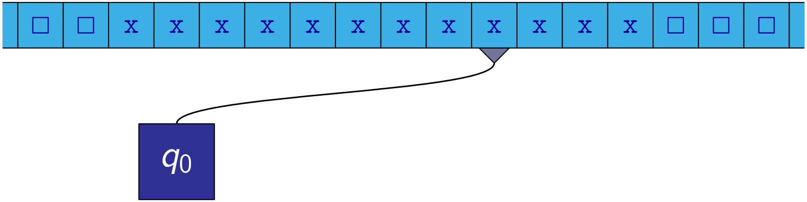
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\Box, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



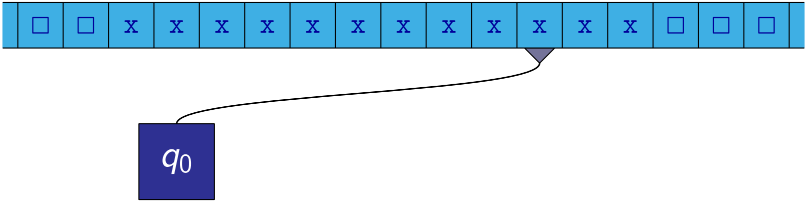
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



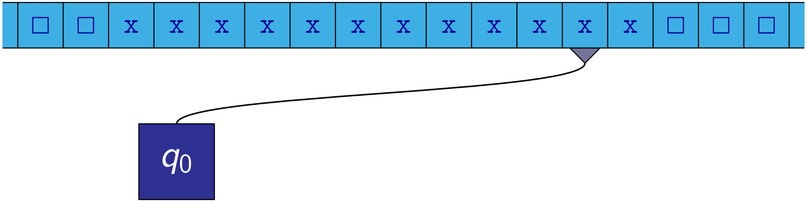
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



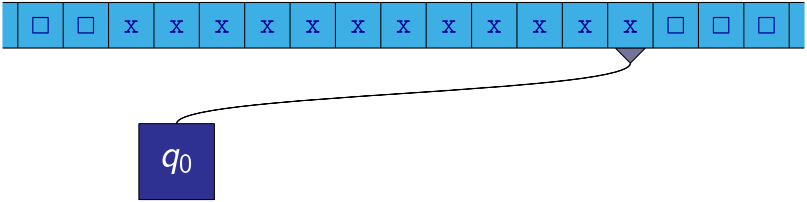
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\Gamma = \{\square, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathrm{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathrm{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathrm{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathrm{x}, +1)$	$(q_2, \mathrm{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathrm{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathrm{x}, -1)$



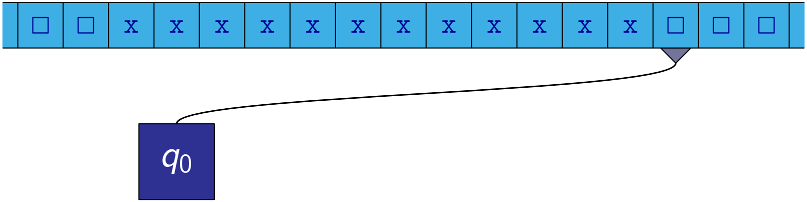
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\qquad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\qquad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{acc}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \square, 0)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{rej}, \mathtt{a}, 0)$	$(q_{rej}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



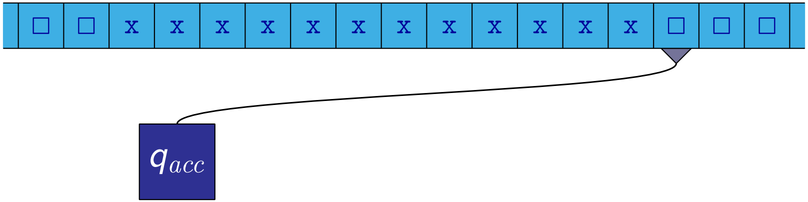
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

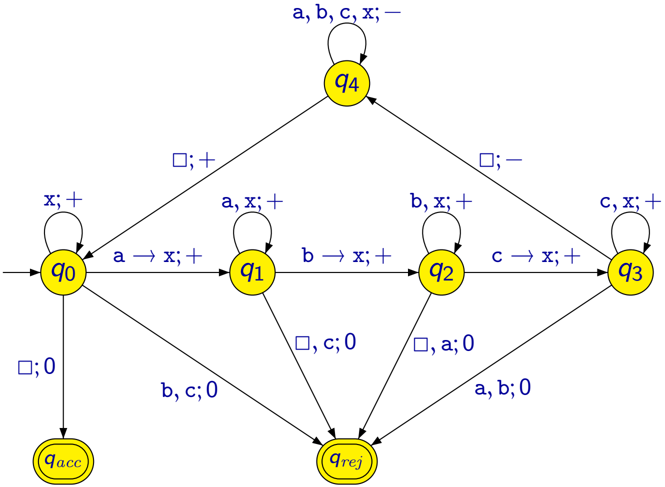
$Q = \{q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej}\}$ $\quad F = \{q_{acc}, q_{rej}\}$

$\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ $\quad \Gamma = \{\Box, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathtt{x}\}$

δ	$\Box$	a	b	c	x
q_0	$(q_{acc}, \Box, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{rej}, \Box, 0)$	$(q_1, \mathrm{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{rej}, \mathrm{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{rej}, \Box, 0)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_2, \mathrm{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \Box, -1)$	$(q_{rej}, \mathrm{a}, 0)$	$(q_{rej}, \mathrm{b}, 0)$	$(q_3, \mathrm{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \Box, +1)$	$(q_4, \mathrm{a}, -1)$	$(q_4, \mathrm{b}, -1)$	$(q_4, \mathrm{c}, -1)$	$(q_4, \mathtt{x}, -1)$



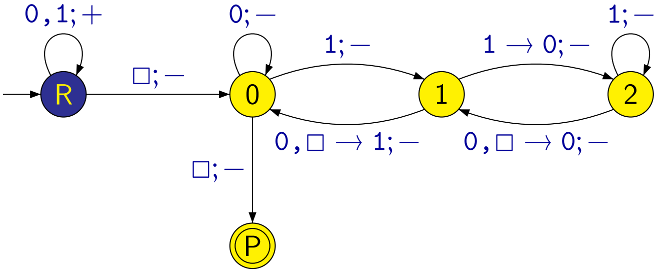
Turing Machine

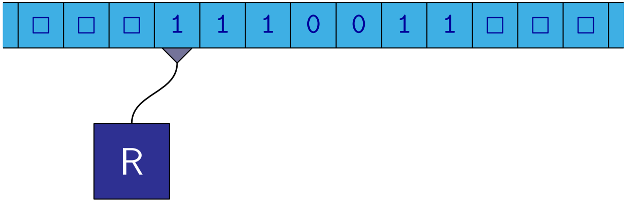


Turing Machine

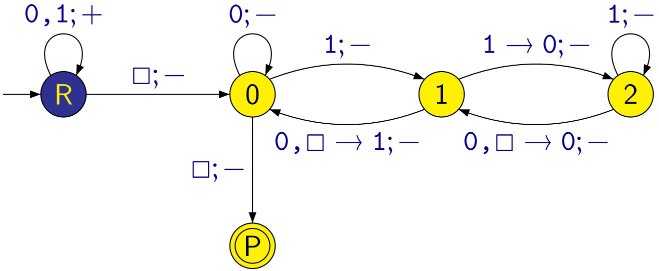
- A Turing machine can give not only answers $\textsc{Yes}$ or $\textsc{No}$ but it can also compute a function that assigns to each word from Σ^* some other word (from Γ^*).
- A word assigned to a word w is the word that remains on the tape after the computation over the word w when we remove all symbols $\square$.

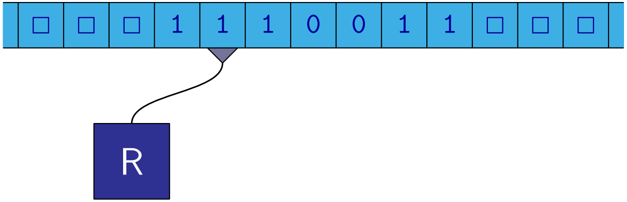
Turing Machine – Multiplication by Three



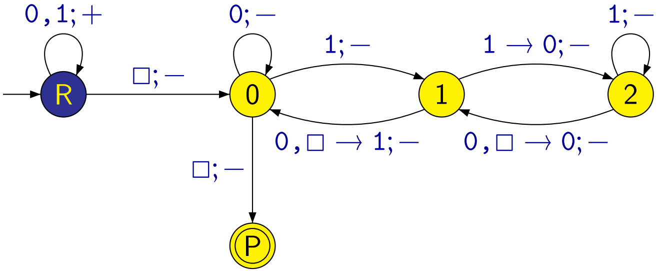


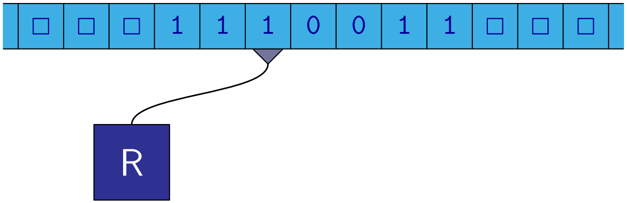
Turing Machine – Multiplication by Three



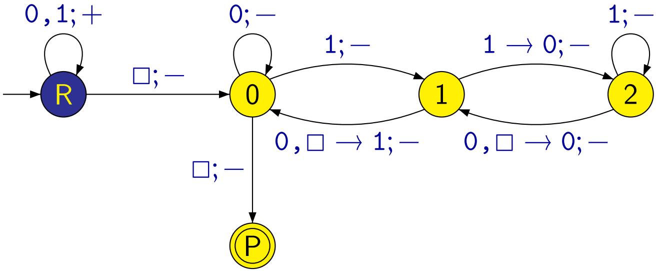


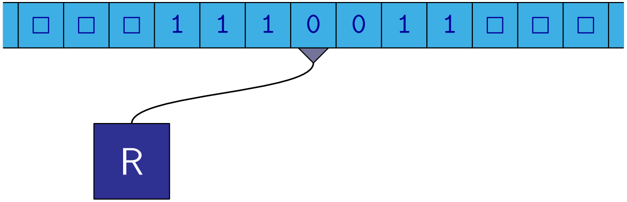
Turing Machine – Multiplication by Three



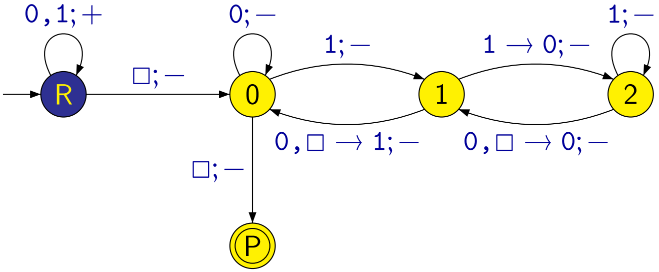


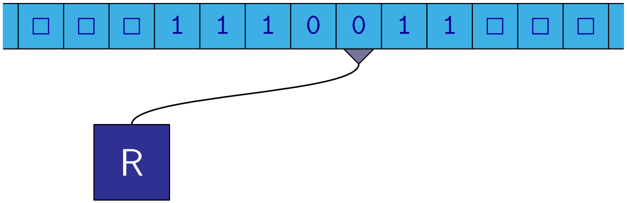
Turing Machine – Multiplication by Three



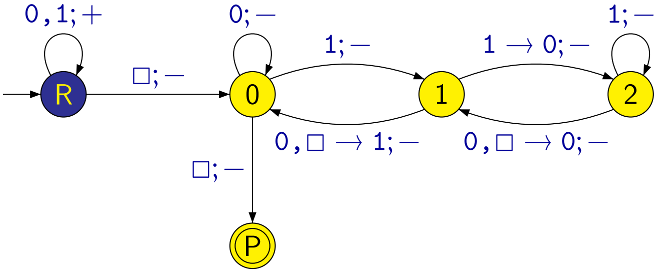


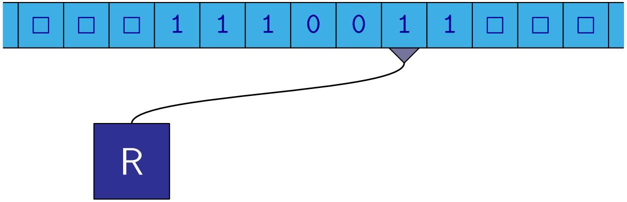
Turing Machine – Multiplication by Three



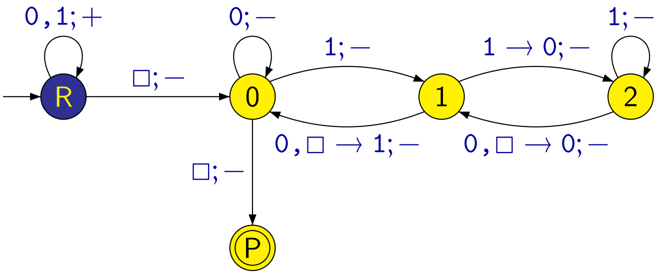


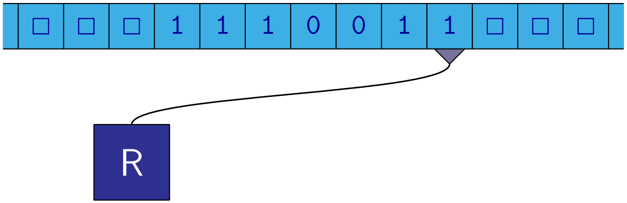
Turing Machine – Multiplication by Three



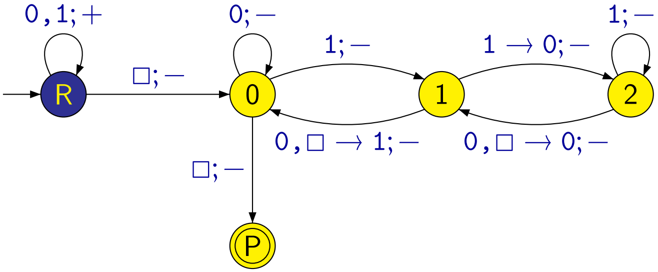


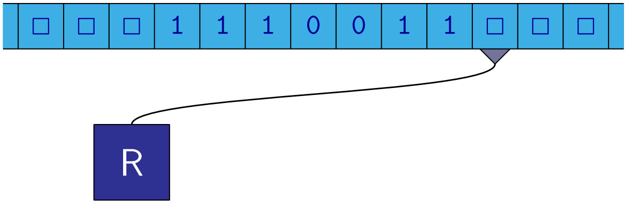
Turing Machine – Multiplication by Three



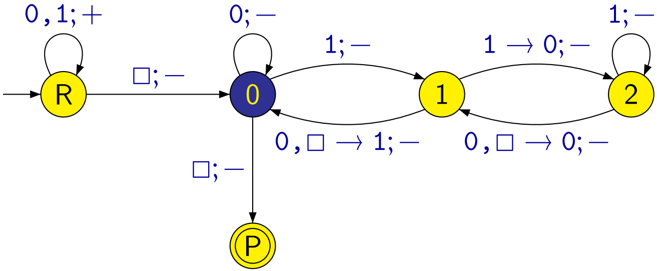


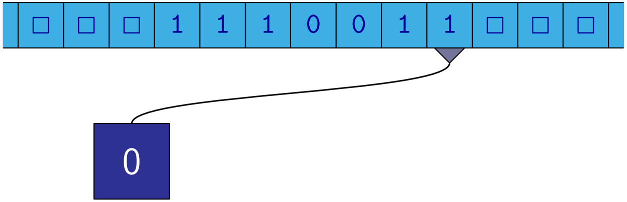
Turing Machine – Multiplication by Three



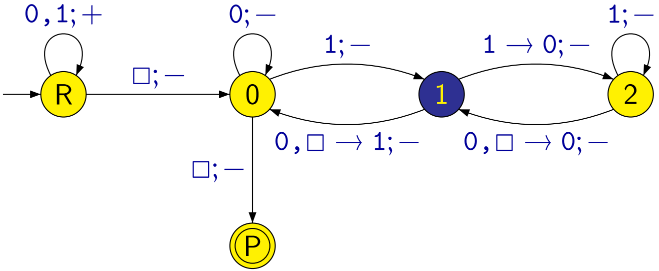


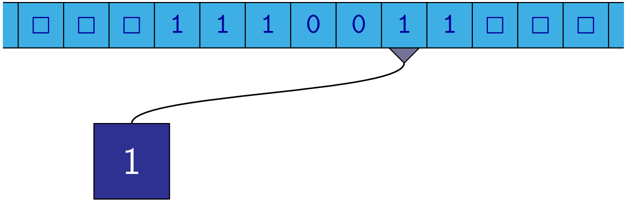
Turing Machine – Multiplication by Three



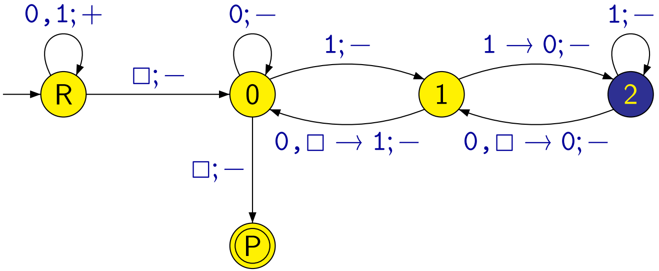


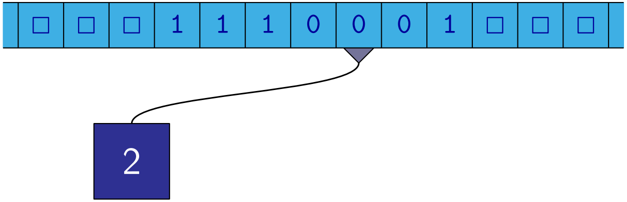
Turing Machine – Multiplication by Three



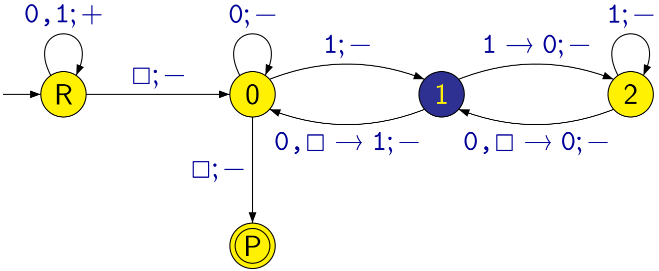


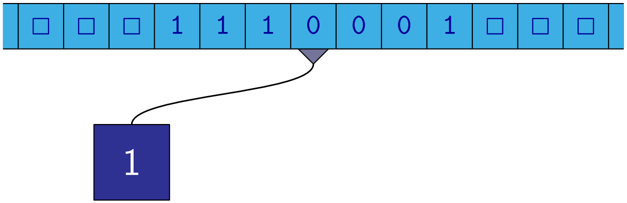
Turing Machine – Multiplication by Three



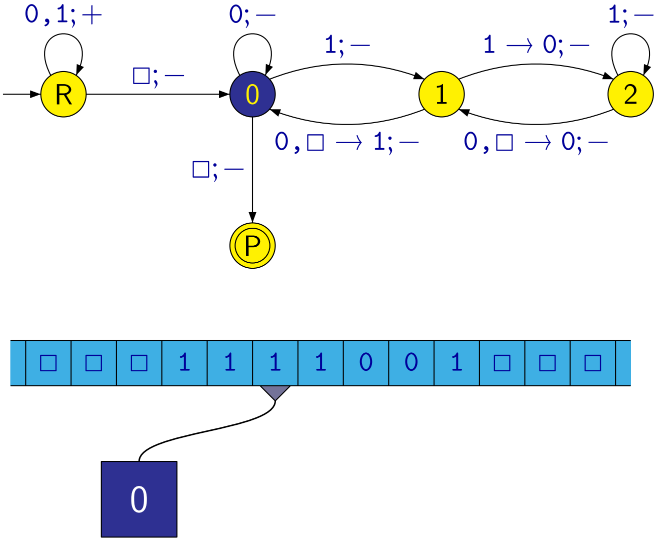


Turing Machine – Multiplication by Three

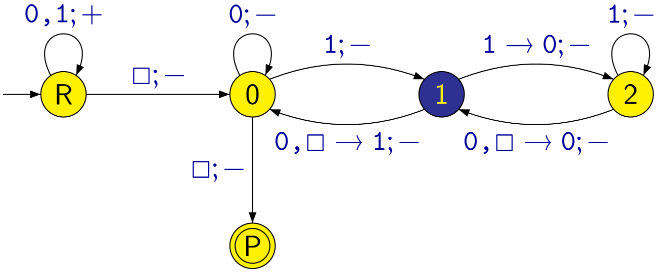


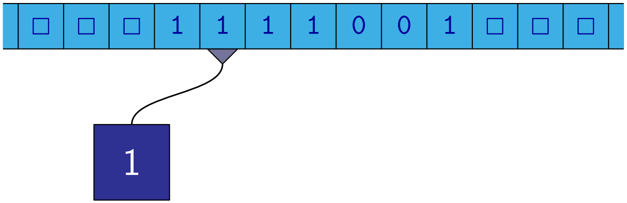


Turing Machine – Multiplication by Three

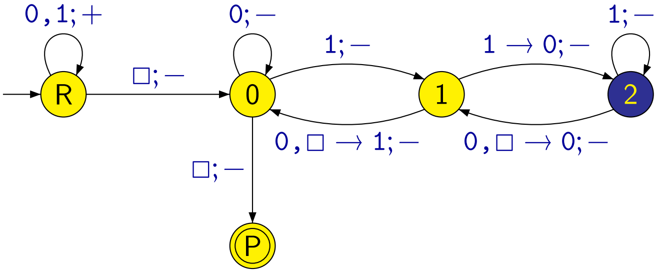


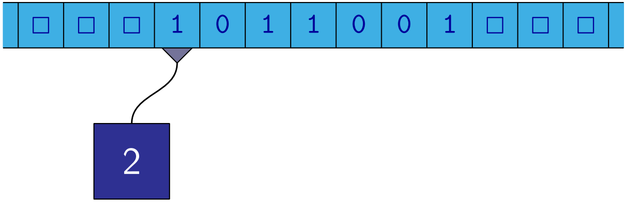
Turing Machine – Multiplication by Three



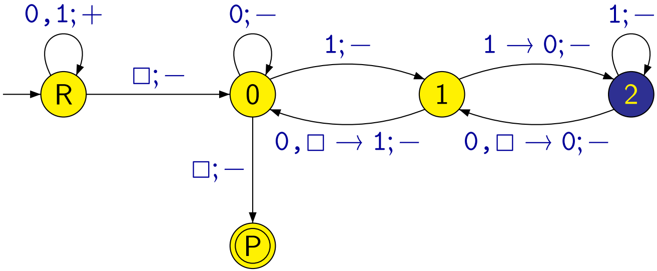


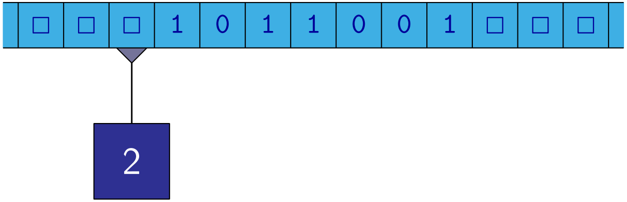
Turing Machine – Multiplication by Three



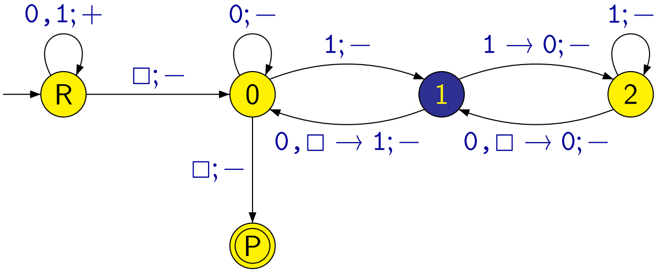


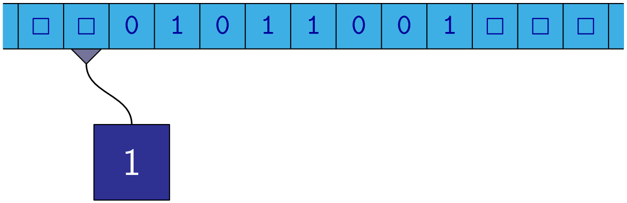
Turing Machine – Multiplication by Three



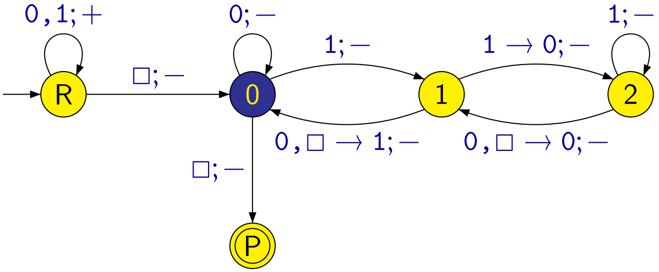


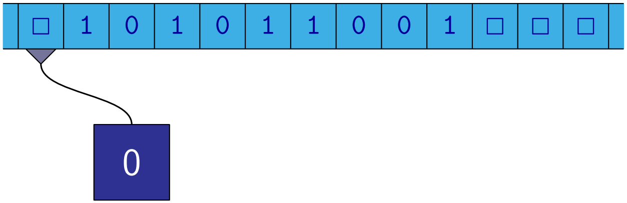
Turing Machine – Multiplication by Three



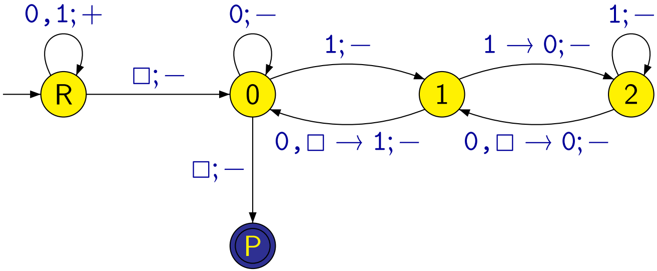


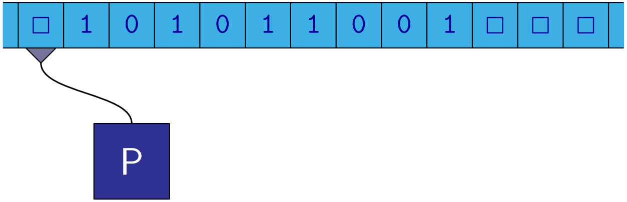
Turing Machine – Multiplication by Three





Turing Machine – Multiplication by Three





Nondeterministic Turing Machines

We can also consider **nondeterministic Turing machines** where for every state q and symbol b the transition function $\delta(q, b)$ specifies several different triples (q', b', d).

The machine can choose any of them.

The machine accepts a word w iff it has at least one computation where w is accepted.

Remark: For every nondeterministic Turing machine, there can be constructed an equivalent deterministic Turing machine.

Nondeterministic Turing Machines

Formally, the only difference in the definition of a deterministic and a nondeterministic Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is the definition of the transition function δ:

- **Deterministic** Turing machine:

$$\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$$

- **Nondeterministic Turing machine**:

$$\delta : (Q - F) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{-1, 0, +1\})$$

Remark: For nondeterministic Turing machines, it makes a little sense to consider other set of final state than $F = \{q_{acc}, q_{rej}\}$.

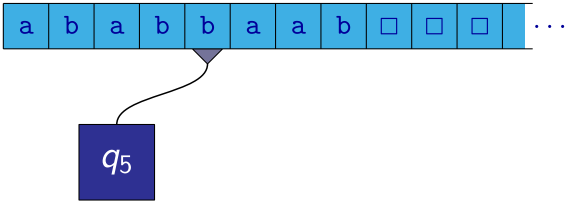
Variants of Turing Machines

- The definition of Turing machine given before is just one of many variants.
- Here we give several examples of differences between different variants of Turing machines.
- Almost all these variants of Turing machines are able to accept or recognize the same languages and to compute the same functions.
- There can be (but need not be) big differences between variants with respect to their running time and an amount of used memory.
- All these variants can be considered in a deterministic and a nondeterministic version.

Variants of Turing Machines

One-sided or **two-sided** infinite tape:

- In the previous definition, we have considered a tape that is infinite in both directions — to the left and to the right.
- Instead, it is sometimes considered a tape that is infinite only to the right.

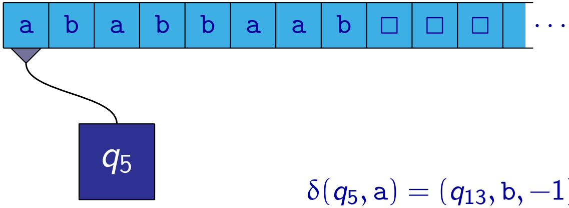


Variants of Turing Machines

It is necessary to define what should happen when the head is on the leftmost cell of the tape and, according to the transition function, it should move to the left.

Two most common possibilities:

- An "error" occurs and the computation is (unsuccesfully) ended:

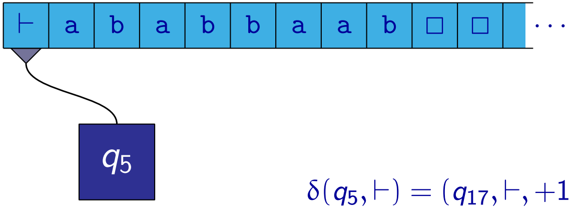


$\delta(q_5, \mathtt{a}) = (q_{13}, \mathtt{b}, -1)$

Variants of Turing Machines

- The left end of the tape contains a "marker" represented by a special symbol $\vdash \in (\Gamma - \Sigma)$.

  This marker can not be overwritten and a move to the left is forbidden on this symbol, i.e., for each $q \in Q$ it holds that if $\delta(q, \vdash) = (q', b, d)$ then $b = \vdash$ a $d \in \{0, +1\}$.



$$\delta(q_5, \vdash) = (q_{17}, \vdash, +1)$$

Variants of Turing Machines

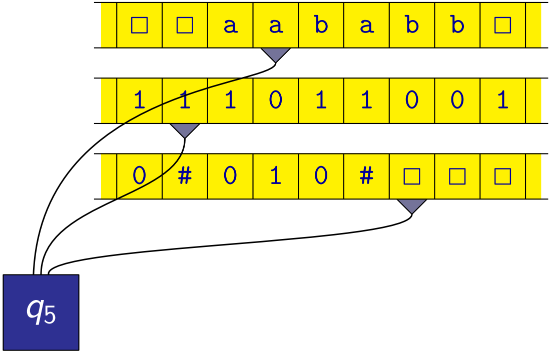
Remark: The possibility that a computation can end unsuccessfully because of an error where it is no possible to continue from the given configuration is quite common also for other types of machines we will consider.

Generally, the following possibilities can happen in a computation:

- The computation ends successfully in a final configuration that corresponds to a correct halting.
- The computation is stuck in a configuration that is not final but it is not possible to continue there — this is considered as a computation ending with an error.
- The computation never halts.

Variants of Turing Machines

Multitape Turing machines are often considered.



Variants of Turing Machines

In the case of a multitape machines:

- Each of k tapes has its own alphabet, i.e., we have tape alphabets Γ_1, Γ_2, ..., Γ_k.
- The transition function δ is of the type

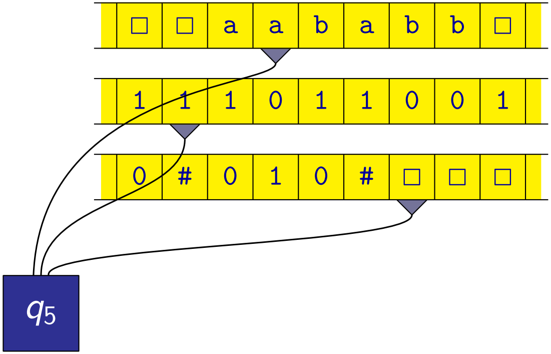
$$(Q - F) \times \Gamma_1 \times \cdots \times \Gamma_k \;\rightarrow\; Q \times \Gamma_1 \times \{-1, 0, +1\} \times \cdots \times \Gamma_k \times \{-1, 0, +1\}$$

Example:

$$\delta(q_5, \mathrm{a}, 1, \Box) \;=\; (q_{12}, \mathrm{a}, -1, \mathrm{x}, 0, 1, +1)$$

Variants of Turing Machines

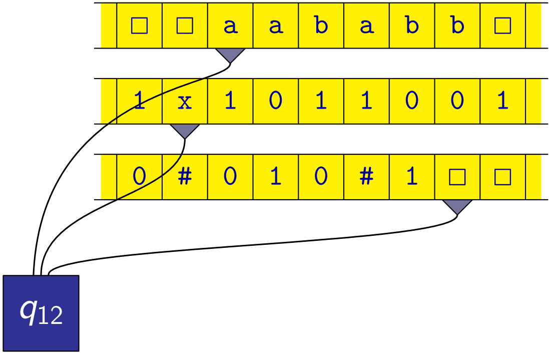
Example:



$$\delta(q_5, \mathrm{a}, 1, \square) = (q_{12}, \mathrm{a}, -1, \mathrm{x}, 0, 1, +1)$$

Variants of Turing Machines

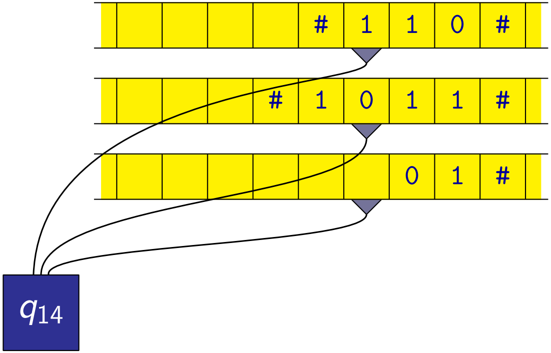
Example:



$$\delta(q_5, \mathrm{a}, 1, \square) = (q_{12}, \mathrm{a}, -1, \mathrm{x}, 0, 1, +1)$$

Variants of Turing Machines

Example: A machine that gets as an input two natural numbers written in binary and separated by symbols # (e.g., number 6 and 11 will be written as "#110#" a "#1011#").

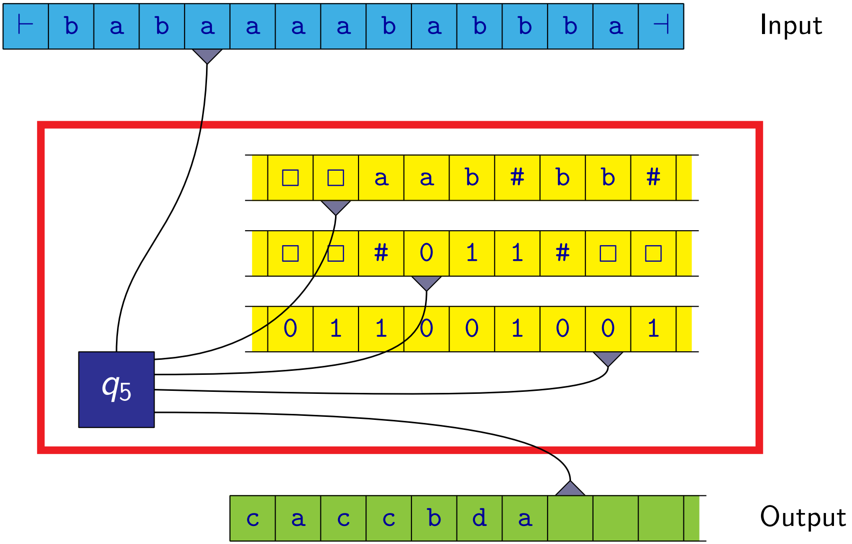


Variants of Turing Machines

Multitape machines often use one of its tapes as an input tape and one of its tapes as an output tape. Other tapes are used as working tapes:

- **Input tape** — it contains an input word, the machine can not write on it (it is read-only), it is not infinite
- **Working tapes** — the machine can read from them and write on them (they are read/write), at the beginning of a computation they are empty (they contain only symbols $\square$)
- **Output tape** — the machine can only write on it (it is write-only), it can not read from it, it is empty at the beginning of a computation, the head can move only from the left to the right

Variants of Turing Machines



Variants of Turing Machines

If a machine has a special separate input tape (which is read-only), the following two variants are typically used:

- The head on this tape can move to the left and to the right.

  In this case, an input word $w \in \Sigma^*$ is bounded from the left and from the right using "endmarkers", i.e., special symbols $\vdash, \dashv \in (\Gamma - \Sigma)$.

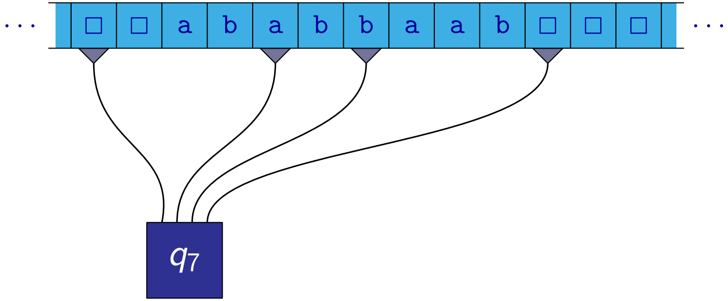
- The head can move only from left to right.

Remark: The variant with possible movement in both directions and endmarkers is more common.

If it is not specified otherwise, we will consider this variant.

Variants of Turing Machines

Instead of several tapes, we can consider **several heads** on one tape:



Variants of Turing Machines

In the variant with several heads on one tape, it is necessary to specify:

- If there can be more than one head in the same time on one tape cell.
- If this is the case, what is the behaviour of the machine if several heads occurring on the same cell want to write different symbols on this cell.
- Whether the given machine can detect the situation when several head are on the same cell.

Remark: Of course, in general we can consider machines with several tapes where each of these tapes is equipped with several heads.

Variants of Turing Machines

Consider a machine with several tapes and with arbitrary number of heads on each tape.

Instead of describing a transition function that works with all heads in each step, we can alternatively describe the behaviour of the machine by a **program** consisting of simpler instructions of the following types:

- to move a given head by one cell to the left
- to move a given head by one cell to the right
- to write a specified symbol on the given position of a specified head on a tape
- to read one symbol from a position of a given head and to branch the program according this symbol (i.e., to go to different states of the control unit)

Variants of Turing Machines

So far we considered only **linear** (one-dimensitional) tapes.

Instead, the memory with cells (where every cell contains one symbol from some alphabet) can have some other structure.

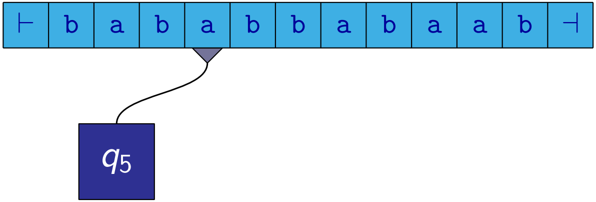
For example:

- two-dimensional **square grid**
  — a movement of a head into four directions: left, right, up, down
- d-dimensional memory for some $d = 3, 4, \ldots$
  (three-dimensional, four-dimensional, etc.)
- a memory organized in a form of an (infinite) tree
- ...

Linear Bounded Automaton

Linear bounded automaton (**LBA**):

- A nondeterministic Turing machine that can use only the part of the tape where its input word is written.
- Cells of the tape, which at the beginning contain symbols of an input word, can be arbitrarily overwritten during a computation.
- Left and right endmarkers around the word. These endmarkers can not be overwritten.
- It is not possible to move the head to the left of the left endmarker and to the right of the right endmarker.



Linear Bounded Automaton

- Linear bounded automata can be considered in both **deterministic** and **nondeterministic** version.
- The nondeterministic version is considered as the default (i.e., if it is not specified otherwise).
- The question whether every language that can be recognized by a nondeterministic LBA can be also also recognized by a deterministic LBA is an open problem.

Remark: From the point of view of languages that they are able to accept or recognize and from the point view of functions that they can compute, linear bounded automata are considerably weaker than Turing machines that can use memory of unbounded size (in the form of an infinite tape).

Chomsky Hierarchy

Generative Grammars

Definition

A **generative grammar** is a tuple $\mathcal{G} = (\Pi, \Sigma, S, P)$, where

- Π is a finite set of nonterminals
- Σ is a finite set of terminals, $\Pi \cap \Sigma = \emptyset$
- $S \in \Pi$ is the initial nonterminal
- P is a finite set of rules of the form $\alpha \rightarrow \beta$, where $\alpha \in (\Pi \cup \Sigma)^*\Pi(\Pi \cup \Sigma)^*$ and $\beta \in (\Pi \cup \Sigma)^*$.

Example of a rule:

$$CaECb \rightarrow bDFbBDaC$$

Remark: This type of grammar is also called **type-0** grammars, **unrestricted** grammars, or **phrase structure grammars**.

Generative Grammars

Let us assume that we have a generative grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$.

Relation $\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$:

- $\mu_1 \alpha \mu_2 \Rightarrow \mu_1 \beta \mu_2$ if $\alpha \rightarrow \beta$ is a rule from P

Example: If $(BcE \rightarrow DDaBb) \in P$ then

$$CaBC\,BcE\,AccABb \Rightarrow CaBC\,DDaBb\,AccABb$$

A **language** $\mathcal{L}(\mathcal{G})$ generated by a grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is the set of all words over alphabet Σ that can be derived by some derivation from the initial nonterminal S using rules from P, i.e.,

$$\mathcal{L}(\mathcal{G}) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Generative Grammars

Let us assume that we have a generative grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$.

Relation $\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$:

- $\mu_1 \alpha \mu_2 \ \Rightarrow\ \mu_1 \beta \mu_2$ if $\alpha \ \rightarrow\ \beta$ is a rule from P

Example: If $(BcE \ \rightarrow\ DDaBb) \in P$ then

$$CaBC\underline{BcE}AccABb \ \Rightarrow\ CaBC\underline{DDaBb}AccABb$$

A **language** $\mathcal{L}(\mathcal{G})$ generated by a grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is the set of all words over alphabet Σ that can be derived by some derivation from the initial nonterminal S using rules from P, i.e.,

$$\mathcal{L}(\mathcal{G}) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

S

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$S \Rightarrow aSQ$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S \Rightarrow\ & aSQ \\ \Rightarrow\ & aaSQQ \\ \Rightarrow\ & aaaSQQQ \\ \Rightarrow\ & aaaaSQQQQ \\ \Rightarrow\ & aaaaabcQQQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word *aaaaabbbbbccccc*:

$$\begin{aligned} S \Rightarrow\ & aSQ \\ \Rightarrow\ & aaSQQ \\ \Rightarrow\ & aaaSQQQ \\ \Rightarrow\ & aaaaSQQQQ \\ \Rightarrow\ & aaaaabcQQQQ \\ \Rightarrow\ & aaaaabQcQQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$
\begin{aligned}
S &\longrightarrow aSQ \\
S &\longrightarrow abc \\
cQ &\longrightarrow Qc \\
bQc &\longrightarrow bbcc
\end{aligned}
$$

A derivation of word $aaaaabbbbbccccc$:

$$
\begin{aligned}
S &\Rightarrow aSQ \\
&\Rightarrow aaSQQ \\
&\Rightarrow aaaSQQQ \\
&\Rightarrow aaaaSQQQQ \\
&\Rightarrow aaaaabcQQQQ \\
&\Rightarrow aaaaabQcQQQ \\
&\Rightarrow aaaaabbccQQQ
\end{aligned}
$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S \Rightarrow\ & aSQ \\ \Rightarrow\ & aaSQQ \\ \Rightarrow\ & aaaSQQQ \\ \Rightarrow\ & aaaaSQQQQ \\ \Rightarrow\ & aaaaabcQQQQ \\ \Rightarrow\ & aaaaabQcQQQ \\ \Rightarrow\ & aaaaabbccQQQ \\ \Rightarrow\ & aaaaabbcQcQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abc \\ cQ &\rightarrow Qc \\ bQc &\rightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \end{aligned}$$

$$\Rightarrow aaaaabbQccQQ$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abc \\ cQ &\rightarrow Qc \\ bQc &\rightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \\ &\Rightarrow aaaaabbbccQcQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$
\begin{aligned}
S &\rightarrow aSQ \\
S &\rightarrow abc \\
cQ &\rightarrow Qc \\
bQc &\rightarrow bbcc
\end{aligned}
$$

A derivation of word $aaaaabbbbbccccc$:

$$
\begin{aligned}
S &\Rightarrow aSQ \\
&\Rightarrow aaSQQ \\
&\Rightarrow aaaSQQQ \\
&\Rightarrow aaaaSQQQQ \\
&\Rightarrow aaaaabcQQQQ \\
&\Rightarrow aaaaabQcQQQ \\
&\Rightarrow aaaaabbccQQQ \\
&\Rightarrow aaaaabbcQcQQ \\
&\Rightarrow aaaaabbQccQQ \\
&\Rightarrow aaaaabbbcccQQ \\
&\Rightarrow aaaaabbbccQcQ \\
&\Rightarrow aaaaabbbcQccQ
\end{aligned}
$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abc \\ cQ &\rightarrow Qc \\ bQc &\rightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \\ &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \\ &\Rightarrow aaaaabbbccQcQ \\ &\Rightarrow aaaaabbbcQccQ \\ &\Rightarrow aaaaabbbQcccQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word *aaaaabbbbbccccc*:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \\ &\Rightarrow aaaaabbbccQcQ \\ &\Rightarrow aaaaabbbcQccQ \\ &\Rightarrow aaaaabbbQcccQ \\ &\Rightarrow aaaaabbbbccccQ \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \\ &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \\ &\Rightarrow aaaaabbbccQcQ \\ &\Rightarrow aaaaabbbcQccQ \\ &\Rightarrow aaaaabbbQcccQ \\ &\Rightarrow aaaaabbbbccccQ \\ &\Rightarrow aaaaabbbbcccQc \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word *aaaaabbbbbccccc*:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabcQQQQ \\ &\Rightarrow aaaaabQcQQQ \\ &\Rightarrow aaaaabbccQQQ \\ &\Rightarrow aaaaabbcQcQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \\ &\Rightarrow aaaaabbbccQcQ \\ &\Rightarrow aaaaabbbcQccQ \\ &\Rightarrow aaaaabbbQcccQ \\ &\Rightarrow aaaaabbbbccccQ \\ &\Rightarrow aaaaabbbbcccQc \\ &\Rightarrow aaaaabbbbccQcc \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$
\begin{aligned}
S &\longrightarrow aSQ \\
S &\longrightarrow abc \\
cQ &\longrightarrow Qc \\
bQc &\longrightarrow bbcc
\end{aligned}
$$

A derivation of word $aaaaabbbbbccccc$:

$\Rightarrow\ aaaaabbbbcQccc$

$\Rightarrow\ aaaaabbQccQQ$
$\Rightarrow\ aaaaabbbcccQQ$
$\Rightarrow\ aaaaabbbccQcQ$
$\Rightarrow\ aaaaabbbcQccQ$
$\Rightarrow\ aaaaabbbQcccQ$
$\Rightarrow\ aaaaabbbbccccQ$
$\Rightarrow\ aaaaabbbbcccQc$
$\Rightarrow\ aaaaabbbbccQcc$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abc \\ cQ &\longrightarrow Qc \\ bQc &\longrightarrow bbcc \end{aligned}$$

A derivation of word *aaaaabbbbbccccc*:

$$\begin{aligned} &\Rightarrow aaaaabbbbcQccc \\ &\Rightarrow aaaaabbbbQcccc \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQccQQ \\ &\Rightarrow aaaaabbbcccQQ \\ &\Rightarrow aaaaabbbccQcQ \\ &\Rightarrow aaaaabbbcQccQ \\ &\Rightarrow aaaaabbbQcccQ \\ &\Rightarrow aaaaabbbbccccQ \\ &\Rightarrow aaaaabbbbcccQc \\ &\Rightarrow aaaaabbbbccQcc \end{aligned}$$

Generative Grammars

Example: A grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$
\begin{aligned}
S &\longrightarrow aSQ \\
S &\longrightarrow abc \\
cQ &\longrightarrow Qc \\
bQc &\longrightarrow bbcc
\end{aligned}
$$

A derivation of word $aaaaabbbbbccccc$:

$$
\begin{aligned}
&\Rightarrow aaaaabbbbcQccc \\
&\Rightarrow aaaaabbbbQcccc \\
&\Rightarrow aaaaabbbbbccccc
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow aaaaabbQccQQ \\
&\Rightarrow aaaaabbbcccQQ \\
&\Rightarrow aaaaabbbccQcQ \\
&\Rightarrow aaaaabbbcQccQ \\
&\Rightarrow aaaaabbbQcccQ \\
&\Rightarrow aaaaabbbbccccQ \\
&\Rightarrow aaaaabbbbcccQc \\
&\Rightarrow aaaaabbbbccQcc
\end{aligned}
$$

Context-sensitive Grammars

Context-sensitive grammars, also called **type-1** grammars, are a special case of generative grammars.

A grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is called **context-sensitive** if all its rules (with one exception given below) are of the form

$$\alpha X \beta \rightarrow \alpha \gamma \beta$$

where $X \in \Pi$, $\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*$, with $|\gamma| \geq 1$.

The only exception is that the grammar can contain the rule $S \rightarrow \varepsilon$.

If $\mathcal{G}$ contains this rule then the initial nonterminal S can not occur on the right-hand side of any rule.

An example of a rule:

$$BaEC \rightarrow BaDAcBC$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

S

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$S \Rightarrow aSQ$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad\qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} S \Rightarrow\ & aSQ \\ \Rightarrow\ & aaSQQ \\ \Rightarrow\ & aaaSQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad\qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\to aSQ \\ S &\to abC \\ bQC &\to bbCC \\ C &\to c \end{aligned} \qquad\qquad \begin{aligned} CQ &\to XQ \\ XQ &\to XY \\ XY &\to QY \\ QY &\to QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abC \\ bQC &\longrightarrow bbCC \\ C &\longrightarrow c \end{aligned} \qquad \begin{aligned} CQ &\longrightarrow XQ \\ XQ &\longrightarrow XY \\ XY &\longrightarrow QY \\ QY &\longrightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \\ &\Rightarrow aaaaabQYQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S \Rightarrow\ & aSQ \\ \Rightarrow\ & aaSQQ \\ \Rightarrow\ & aaaSQQQ \\ \Rightarrow\ & aaaaSQQQQ \\ \Rightarrow\ & aaaaabCQQQQ \\ \Rightarrow\ & aaaaabXQQQQ \\ \Rightarrow\ & aaaaabXYQQQ \\ \Rightarrow\ & aaaaabQYQQQ \end{aligned} \qquad \begin{aligned} \Rightarrow\ & aaaaabQCQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\to aSQ \\ S &\to abC \\ bQC &\to bbCC \\ C &\to c \end{aligned} \qquad\qquad \begin{aligned} CQ &\to XQ \\ XQ &\to XY \\ XY &\to QY \\ QY &\to QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \\ &\Rightarrow aaaaabQYQQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \\ &\Rightarrow aaaaabQYQQQ \\ &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \\ &\Rightarrow aaaaabQYQQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S \Rightarrow\ & aSQ \\ \Rightarrow\ & aaSQQ \\ \Rightarrow\ & aaaSQQQ \\ \Rightarrow\ & aaaaSQQQQ \\ \Rightarrow\ & aaaaabCQQQQ \\ \Rightarrow\ & aaaaabXQQQQ \\ \Rightarrow\ & aaaaabXYQQQ \\ \Rightarrow\ & aaaaabQYQQQ \end{aligned} \qquad \begin{aligned} \Rightarrow\ & aaaaabQCQQQ \\ \Rightarrow\ & aaaaabbCCQQQ \\ \Rightarrow\ & aaaaabbCXQQQ \\ \Rightarrow\ & aaaaabbCXYQQ \\ \Rightarrow\ & aaaaabbCQYQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word *aaaaabbbbbccccc*:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \\ &\Rightarrow aaaaabQYQQQ \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S &\Rightarrow aSQ \\ &\Rightarrow aaSQQ \\ &\Rightarrow aaaSQQQ \\ &\Rightarrow aaaaSQQQQ \\ &\Rightarrow aaaaabCQQQQ \\ &\Rightarrow aaaaabXQQQQ \\ &\Rightarrow aaaaabXYQQQ \\ &\Rightarrow aaaaabQYQQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \\ &\Rightarrow aaaaabbXQCQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\longrightarrow aSQ \\ S &\longrightarrow abC \\ bQC &\longrightarrow bbCC \\ C &\longrightarrow c \end{aligned} \qquad\qquad \begin{aligned} CQ &\longrightarrow XQ \\ XQ &\longrightarrow XY \\ XY &\longrightarrow QY \\ QY &\longrightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} S \Rightarrow\ & aSQ & \Rightarrow\ & aaaaabQCQQQ \\ \Rightarrow\ & aaSQQ & \Rightarrow\ & aaaaabbCCQQQ \\ \Rightarrow\ & aaaSQQQ & \Rightarrow\ & aaaaabbCXQQQ \\ \Rightarrow\ & aaaaSQQQQ & \Rightarrow\ & aaaaabbCXYQQ \\ \Rightarrow\ & aaaaabCQQQQ & \Rightarrow\ & aaaaabbCQYQQ \\ \Rightarrow\ & aaaaabXQQQQ & \Rightarrow\ & aaaaabbCQCQQ \\ \Rightarrow\ & aaaaabXYQQQ & \Rightarrow\ & aaaaabbXQCQQ \\ \Rightarrow\ & aaaaabQYQQQ & \Rightarrow\ & aaaaabbXYCQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\to aSQ \\ S &\to abC \\ bQC &\to bbCC \\ C &\to c \end{aligned} \qquad\qquad \begin{aligned} CQ &\to XQ \\ XQ &\to XY \\ XY &\to QY \\ QY &\to QC \end{aligned}$$

A derivation of word *aaaabbbbcccc*:

$$\Rightarrow aaaaabbQYCQQ$$

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \\ &\Rightarrow aaaaabbXQCQQ \\ &\Rightarrow aaaaabbXYCQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\to aSQ & CQ &\to XQ \\ S &\to abC & XQ &\to XY \\ bQC &\to bbCC & XY &\to QY \\ C &\to c & QY &\to QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$\Rightarrow\ aaaaabQCQQQ$

$\Rightarrow\ aaaaabbCCQQQ$

$\Rightarrow\ aaaaabbCXQQQ$

$\Rightarrow\ aaaaabbCXYQQ$

$\Rightarrow\ aaaaabbCQYQQ$

$\Rightarrow\ aaaaabbCQCQQ$

$\Rightarrow\ aaaaabbXQCQQ$

$\Rightarrow\ aaaaabbXYCQQ$

$\Rightarrow\ aaaaabbQYCQQ$

$\Rightarrow\ aaaaabbQCCQQ$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$\Rightarrow aaaaabQCQQQ$
$\Rightarrow aaaaabbCCQQQ$
$\Rightarrow aaaaabbCXQQQ$
$\Rightarrow aaaaabbCXYQQ$
$\Rightarrow aaaaabbCQYQQ$
$\Rightarrow aaaaabbCQCQQ$
$\Rightarrow aaaaabbXQCQQ$
$\Rightarrow aaaaabbXYCQQ$
$\Rightarrow aaaaabbQYCQQ$
$\Rightarrow aaaaabbQCCQQ$
$\Rightarrow aaaaabbbCCCQQ$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \\ &\Rightarrow aaaaabbXQCQQ \\ &\Rightarrow aaaaabbXYCQQ \\ &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & \qquad CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ & \qquad &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbQCCQQ & &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbbCCCQQ & &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbbCCXQQ & &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbbCCXYQ & &\Rightarrow aaaaabbCQYQQ \\ & & &\Rightarrow aaaaabbCQCQQ \\ & & &\Rightarrow aaaaabbXQCQQ \\ & & &\Rightarrow aaaaabbXYCQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word *aaaabbbbcccc*:

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \\ &\Rightarrow aaaaabbXQCQQ \\ &\Rightarrow aaaaabbXYCQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad\qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \\ &\Rightarrow aaaaabbbCCQCQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \\ &\Rightarrow aaaaabbXQCQQ \\ &\Rightarrow aaaaabbXYCQQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabQCQQQ \\ &\Rightarrow aaaaabbCCQQQ \\ &\Rightarrow aaaaabbCXQQQ \\ &\Rightarrow aaaaabbCXYQQ \\ &\Rightarrow aaaaabbCQYQQ \\ &\Rightarrow aaaaabbCQCQQ \\ &\Rightarrow aaaaabbXQCQQ \\ &\Rightarrow aaaaabbXYCQQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \\ &\Rightarrow aaaaabbbCCQCQ \\ &\Rightarrow aaaaabbbCXQCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \\ &\Rightarrow aaaaabbbCCQCQ \\ &\Rightarrow aaaaabbbCXQCQ \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabbbCXYCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \\ &\Rightarrow aaaaabbbCCQCQ \\ &\Rightarrow aaaaabbbCXQCQ \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$
\begin{aligned}
S &\to aSQ & \qquad CQ &\to XQ \\
S &\to abC & XQ &\to XY \\
bQC &\to bbCC & XY &\to QY \\
C &\to c & QY &\to QC
\end{aligned}
$$

A derivation of word $aaaabbbbcccc$:

$$
\begin{aligned}
&\Rightarrow aaaaabbQYCQQ & \qquad &\Rightarrow aaaaabbbCXYCQ \\
&\Rightarrow aaaaabbQCCQQ & &\Rightarrow aaaaabbbCQYCQ \\
&\Rightarrow aaaaabbbCCCQQ & &\Rightarrow aaaaabbbCQCCQ \\
&\Rightarrow aaaaabbbCCXQQ \\
&\Rightarrow aaaaabbbCCXYQ \\
&\Rightarrow aaaaabbbCCQYQ \\
&\Rightarrow aaaaabbbCCQCQ \\
&\Rightarrow aaaaabbbCXQCQ
\end{aligned}
$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaabbQYCQQ \\ &\Rightarrow aaaabbQCCQQ \\ &\Rightarrow aaaabbbCCCQQ \\ &\Rightarrow aaaabbbCCXQQ \\ &\Rightarrow aaaabbbCCXYQ \\ &\Rightarrow aaaabbbCCQYQ \\ &\Rightarrow aaaabbbCCQCQ \\ &\Rightarrow aaaabbbCXQCQ \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaabbbCXYCQ \\ &\Rightarrow aaaabbbCQYCQ \\ &\Rightarrow aaaabbbCQCCQ \\ &\Rightarrow aaaabbbXQCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \\ &\Rightarrow aaaaabbbCCQCQ \\ &\Rightarrow aaaaabbbCXQCQ \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ \\ &\Rightarrow aaaaabbQCCQQ \\ &\Rightarrow aaaaabbbCCCQQ \\ &\Rightarrow aaaaabbbCCXQQ \\ &\Rightarrow aaaaabbbCCXYQ \\ &\Rightarrow aaaaabbbCCQYQ \\ &\Rightarrow aaaaabbbCCQCQ \\ &\Rightarrow aaaaabbbCXQCQ \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\to aSQ & CQ &\to XQ \\ S &\to abC & XQ &\to XY \\ bQC &\to bbCC & XY &\to QY \\ C &\to c & QY &\to QC \end{aligned}$$

A derivation of word *aaaabbbbcccc*:

$\Rightarrow\ aaaaabbQYCQQ$
$\Rightarrow\ aaaaabbQCCQQ$
$\Rightarrow\ aaaaabbbCCCQQ$
$\Rightarrow\ aaaaabbbCCXQQ$
$\Rightarrow\ aaaaabbbCCXYQ$
$\Rightarrow\ aaaaabbbCCQYQ$
$\Rightarrow\ aaaaabbbCCQCQ$
$\Rightarrow\ aaaaabbbCXQCQ$
$\Rightarrow\ aaaaabbbCXYCQ$
$\Rightarrow\ aaaaabbbCQYCQ$
$\Rightarrow\ aaaaabbbCQCCQ$
$\Rightarrow\ aaaaabbbXQCCQ$
$\Rightarrow\ aaaaabbbXYCCQ$
$\Rightarrow\ aaaaabbbQYCCQ$
$\Rightarrow\ aaaaabbbQCCCQ$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbQYCQQ & &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbQCCQQ & &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCCCQQ & &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbCCXQQ & &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbCCXYQ & &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbCCQYQ & &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbCCQCQ & &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbCXQCQ & &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$\Rightarrow\ aaaaabbbbCCCXQ$

$$\begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaabbbbCCCXQ \\ &\Rightarrow aaaabbbbCCCXY \\ &\Rightarrow aaaabbbbCCCQY \\ &\Rightarrow aaaabbbbCCCQC \\ &\Rightarrow aaaabbbbCCXQC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaabbbCXYCQ \\ &\Rightarrow aaaabbbCQYCQ \\ &\Rightarrow aaaabbbCQCCQ \\ &\Rightarrow aaaabbbXQCCQ \\ &\Rightarrow aaaabbbXYCCQ \\ &\Rightarrow aaaabbbQYCCQ \\ &\Rightarrow aaaabbbQCCCQ \\ &\Rightarrow aaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaabbbbCCCXQ \\ &\Rightarrow aaaabbbbCCCXY \\ &\Rightarrow aaaabbbbCCCQY \\ &\Rightarrow aaaabbbbCCCQC \\ &\Rightarrow aaaabbbbCCXQC \\ &\Rightarrow aaaabbbbCCXYC \\ &\Rightarrow aaaabbbbCCQYC \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaabbbCXYCQ \\ &\Rightarrow aaaabbbCQYCQ \\ &\Rightarrow aaaabbbCQCCQ \\ &\Rightarrow aaaabbbXQCCQ \\ &\Rightarrow aaaabbbXYCCQ \\ &\Rightarrow aaaabbbQYCCQ \\ &\Rightarrow aaaabbbQCCCQ \\ &\Rightarrow aaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \\ &\Rightarrow aaaaabbbbCCQYC \\ &\Rightarrow aaaaabbbbCCQCC \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabbbCXYCQ \\ &\Rightarrow aaaaabbbCQYCQ \\ &\Rightarrow aaaaabbbCQCCQ \\ &\Rightarrow aaaaabbbXQCCQ \\ &\Rightarrow aaaaabbbXYCCQ \\ &\Rightarrow aaaaabbbQYCCQ \\ &\Rightarrow aaaaabbbQCCCQ \\ &\Rightarrow aaaaabbbbCCCCQ \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$
\begin{aligned}
S &\rightarrow aSQ & \qquad CQ &\rightarrow XQ \\
S &\rightarrow abC & XQ &\rightarrow XY \\
bQC &\rightarrow bbCC & XY &\rightarrow QY \\
C &\rightarrow c & QY &\rightarrow QC
\end{aligned}
$$

A derivation of word $aaaaabbbbbccccc$:

$\Rightarrow aaaaabbbbCCCXQ$
$\Rightarrow aaaaabbbbCCCXY$
$\Rightarrow aaaaabbbbCCCQY$
$\Rightarrow aaaaabbbbCCCQC$
$\Rightarrow aaaaabbbbCCXQC$
$\Rightarrow aaaaabbbbCCXYC$
$\Rightarrow aaaaabbbbCCQYC$
$\Rightarrow aaaaabbbbCCQCC$
$\Rightarrow aaaaabbbbCXQCC$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \\ &\Rightarrow aaaaabbbbCCQYC \\ &\Rightarrow aaaaabbbbCCQCC \\ &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \\ &\Rightarrow aaaaabbbbCCQYC \\ &\Rightarrow aaaaabbbbCCQCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \\ &\Rightarrow aaaaabbbbCCQYC \\ &\Rightarrow aaaaabbbbCCQCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaabbbbCCCXQ \\ &\Rightarrow aaaabbbbCCCXY \\ &\Rightarrow aaaabbbbCCCQY \\ &\Rightarrow aaaabbbbCCCQC \\ &\Rightarrow aaaabbbbCCXQC \\ &\Rightarrow aaaabbbbCCXYC \\ &\Rightarrow aaaabbbbCCQYC \\ &\Rightarrow aaaabbbbCCQCC \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaabbbbCXQCC \\ &\Rightarrow aaaabbbbCXYCC \\ &\Rightarrow aaaabbbbCQYCC \\ &\Rightarrow aaaabbbbCQCCC \\ &\Rightarrow aaaabbbbXQCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ & CQ &\rightarrow XQ \\ S &\rightarrow abC & XQ &\rightarrow XY \\ bQC &\rightarrow bbCC & XY &\rightarrow QY \\ C &\rightarrow c & QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaabbbbCCCXQ & &\Rightarrow aaaabbbbCXQCC \\ &\Rightarrow aaaabbbbCCCXY & &\Rightarrow aaaabbbbCXYCC \\ &\Rightarrow aaaabbbbCCCQY & &\Rightarrow aaaabbbbCQYCC \\ &\Rightarrow aaaabbbbCCCQC & &\Rightarrow aaaabbbbCQCCC \\ &\Rightarrow aaaabbbbCCXQC & &\Rightarrow aaaabbbbXQCCC \\ &\Rightarrow aaaabbbbCCXYC & &\Rightarrow aaaabbbbXYCCC \\ &\Rightarrow aaaabbbbCCQYC & & \\ &\Rightarrow aaaabbbbCCQCC & & \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\to aSQ \\ S &\to abC \\ bQC &\to bbCC \\ C &\to c \end{aligned} \qquad \begin{aligned} CQ &\to XQ \\ XQ &\to XY \\ XY &\to QY \\ QY &\to QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \\ &\Rightarrow aaaaabbbbCCQYC \\ &\Rightarrow aaaaabbbbCCQCC \end{aligned} \qquad \begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCCCXQ \\ &\Rightarrow aaaaabbbbCCCXY \\ &\Rightarrow aaaaabbbbCCCQY \\ &\Rightarrow aaaaabbbbCCCQC \\ &\Rightarrow aaaaabbbbCCXQC \\ &\Rightarrow aaaaabbbbCCXYC \\ &\Rightarrow aaaaabbbbCCQYC \\ &\Rightarrow aaaaabbbbCCQCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \\ &\Rightarrow aaaaabbbbQCCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaabbbbcccc$:

$$\Rightarrow \; aaaabbbbCCCC$$

$$\begin{aligned} &\Rightarrow \; aaaabbbbCXQCC \\ &\Rightarrow \; aaaabbbbCXYCC \\ &\Rightarrow \; aaaabbbbCQYCC \\ &\Rightarrow \; aaaabbbbCQCCC \\ &\Rightarrow \; aaaabbbbXQCCC \\ &\Rightarrow \; aaaabbbbXYCCC \\ &\Rightarrow \; aaaabbbbQYCCC \\ &\Rightarrow \; aaaabbbbQCCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbbCCCCC \\ &\Rightarrow aaaaabbbbbcCCCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \\ &\Rightarrow aaaaabbbbQCCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \\ &\Rightarrow aaaaabbbbQCCCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbbCCCCC \\ &\Rightarrow aaaaabbbbbcCCCC \\ &\Rightarrow aaaaabbbbbccCCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \\ &\Rightarrow aaaaabbbbQCCCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbbCCCCC \\ &\Rightarrow aaaaabbbbbcCCCC \\ &\Rightarrow aaaaabbbbbccCCC \\ &\Rightarrow aaaaabbbbbcccCC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \\ &\Rightarrow aaaaabbbbQCCCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbbCCCCC \\ &\Rightarrow aaaaabbbbbcCCCC \\ &\Rightarrow aaaaabbbbbccCCC \\ &\Rightarrow aaaaabbbbbcccCC \\ &\Rightarrow aaaaabbbbbccccC \end{aligned}$$

Context-sensitive Grammars

A context-sensitive grammar generating language $L = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{aligned} S &\rightarrow aSQ \\ S &\rightarrow abC \\ bQC &\rightarrow bbCC \\ C &\rightarrow c \end{aligned} \qquad \begin{aligned} CQ &\rightarrow XQ \\ XQ &\rightarrow XY \\ XY &\rightarrow QY \\ QY &\rightarrow QC \end{aligned}$$

A derivation of word $aaaaabbbbbccccc$:

$$\begin{aligned} &\Rightarrow aaaaabbbbCXQCC \\ &\Rightarrow aaaaabbbbCXYCC \\ &\Rightarrow aaaaabbbbCQYCC \\ &\Rightarrow aaaaabbbbCQCCC \\ &\Rightarrow aaaaabbbbXQCCC \\ &\Rightarrow aaaaabbbbXYCCC \\ &\Rightarrow aaaaabbbbQYCCC \\ &\Rightarrow aaaaabbbbQCCCC \end{aligned}$$

$$\begin{aligned} &\Rightarrow aaaaabbbbbCCCCC \\ &\Rightarrow aaaaabbbbbcCCCC \\ &\Rightarrow aaaaabbbbbccCCC \\ &\Rightarrow aaaaabbbbbcccCC \\ &\Rightarrow aaaaabbbbbccccC \\ &\Rightarrow aaaaabbbbbccccc \end{aligned}$$

Context-free Grammars

Another special type of generative grammars are **context-free grammars**. Context-free grammars are also called **type-2** grammars.

A grammar $\mathcal{G} = (\Pi, \Sigma, S, P)$ is **context-free** if all its rules are of the form

$$X \rightarrow \gamma$$

where $X \in \Pi$, $\gamma \in (\Pi \cup \Sigma)^*$.

A example of a rule:

$$C \rightarrow DaBBc$$

Context-free Grammars

Remark: Not every context-free grammar is context-sensitive since a context-free grammar can contain also some other ε-rules (i.e., rules of the form $X \longrightarrow \varepsilon$) in addition to $S \longrightarrow \varepsilon$.

Arbitrary context-free grammar without ε-rules (resp. with at most one ε-rule $S \longrightarrow \varepsilon$ where nonterminal S does not occur on the right-hand side of any rule) is a special case of a context-sensitive grammar.

For every context-free grammar $\mathcal{G}$, it is possible an equivalent context-free grammar without ε-rules.

So for every context-free grammar, there is an equivalent context-sensitive grammar.

Regular Grammars

Let us recall that a grammar is a **right** (resp. **left**) **regular** grammar if all its rules are of the following forms:

- $A \longrightarrow wB$ (resp. $A \longrightarrow Bw$)
- $A \longrightarrow w$

where $A, B \in \Pi$, $w \in \Sigma^*$.

A grammar is **regular** if it is a right or left regular grammar.

Regular grammar are denoted as **type 3** grammars.

It is obvious that regular grammars are a special case of context-free grammars.

Chomsky Hierarchy

So according to the types of rules that can be used in a grammar, the grammars can be divided into these four types:

- **Type-0** — General **generative grammars**

  no restrictions on the rules

- **Type-1** — **Context-sensitive grammars**

  rules of the form $\alpha X \beta \rightarrow \alpha \gamma \beta$, where $|\gamma| \geq 1$
  (An exception is possible rule $S \rightarrow \varepsilon$, but then S does not occur on the right-hand side of any rule.)

- **Type-2** — **context-free grammars**

  rules of the form $X \rightarrow \gamma$

- **Type-3** — **regular grammars**

  rules of the form $X \rightarrow wY$ (resp. $X \rightarrow Yw$) or $X \rightarrow w$

where $\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*$, $X \in \Pi$, and $w \in \Sigma^*$

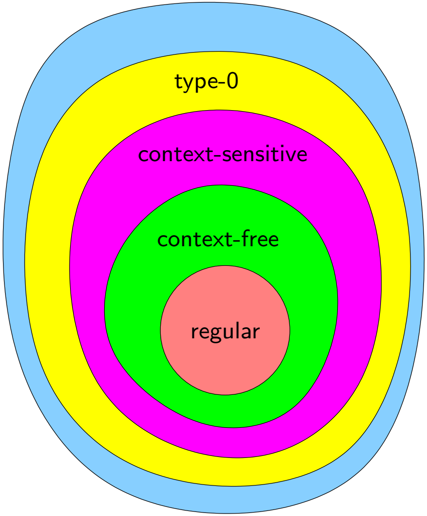
Chomsky Hierarchy

For all these types of grammars, there are corresponding classes of languages:

- **Type-0**: Language L is **recursively enumerable** (or of **type-0**) if there exists a generative grammar generating this language.
- **Type-1**: Language L is **context-sensitive** (or of **type-1**) if there exists a context-sensitive grammar generating this language.
- **Type-2**: Language L is **context-free** (or of **type-2**) if there exists a context-free grammar generating this language.
- **Type-3**: Language L is **regular** (or of **type-3**) if there exists a regular grammar generating this language.

Chomsky Hierarchy

Classes of languages:



Chomsky Hierarchy

- An example of a language that is context-free but is not regular:

$$\{a^n b^n \mid n \geq 1\}$$

- An example of a language that is context-sensitive but is not context-free:

$$\{a^n b^n c^n \mid n \geq 1\}$$

Chomsky Hierarchy

- Examples of languages that are type-0 but are not context-sensitive:
  - A language consisting of words that represent logically valid formulas of predicate logic.
  - Language consisting of words that represent codes of those Turing machines that will halt in a computation over an empty word after a finite number of steps.
- Examples of languages that are not of type-0:
  - A language consisting of those words that represent exactly those formulas of predicate logic, which are not logically valid.
  - A language constisting of words that represent codes of those Turing machines that never halt in a computation over an empty word.
  - A language consisting of words that represent codes of those Turing machines that will always halt after some finite number of steps in a computation over an arbitrary word.

Chomsky Hierarchy

- Other possible characterizations of **regular** languages:
  - languages accepted by finite automata (deterministic, nondeterministic, generalized nondeterministic)
  - languages that can be described by regular expressions
- Other possible characterization of **context-free** languages:
  - languages accepted by nondeterministic pushdown automata
- Other possible characterization of **context-sensitive** languages:
  - languages accepted by nondeterministic linear bounded automata
- Other possible characterization of **type-0** languages:
  - languages accepted by (deterministic or nondeterministic) Turing machines

Chomsky Hierarchy

Chomsky hierarchy — summary:

- **Type-0** — **recursively enumerable** languages:
    - unrestricted generative grammars
    - Turing machines (deterministic, nondeterministic)
- **Type-1** — **context-sensitive** languages:
    - context-sensitive grammars
    - nondeterministic linear bounded automata
- **Type-2** — **context-free** languages:
    - context-free grammars
    - nondeterministic pushdown automata
- **Type-3** — **regular** languages:
    - regular grammars
    - finite automata (deterministic, nondeterministic)
    - regular expressions

Models of Computation

Computation of an Algorithm

Algorithms are executed on machines — 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** and produces the corresponding **output** during its computation.

Models of Computation

Model of Computation — an idealized mathematical model of a computer

- abstracts from some unimportant implementation details
- we want to analyze those propeties of algorithms that are as much as possible independent of details of a machine that will execute the given algorithm

Examples of some models of computation:

- finite automata
- pushdown automata
- Turing machines
- random-access machines
- ...

Models of Computation

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
- what kind of operations the machine can do with this data

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

Models of Computation

One role, for which models of computations are used for, is to define precisely some notions that are important for specifying **computational complexity** of a given algorithm:

- **running time** of a given algorithm $\mathcal{A}$ for a given input w (remark: typically, it is a number of steps performed during the computation by the machine)
- **amount of memory** used by the machine during this computation

In general, it is also important for different models of computation

- whether a given type of machine is able to **simulate** computations of some other type of machine
- how the running time or the amount of used memory differs compared to the original machine

Simulation of a Computation

Explanation what it means that a machine $\mathcal{M}$ is **simulated** by a machine $\mathcal{M}'$:

- A computation of machine $\mathcal{M}$ for input w is a (finite or infinite) sequence of configurations of machine $\mathcal{M}$

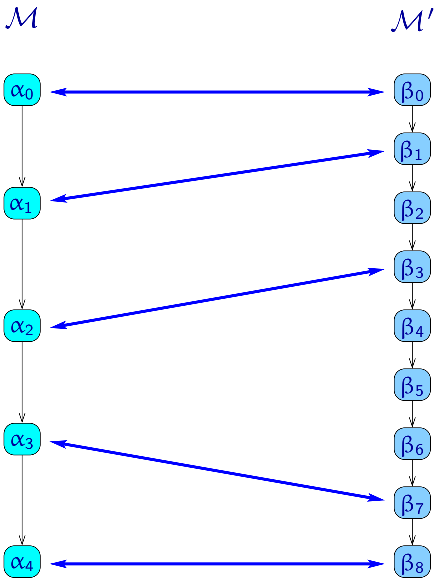
$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots$$

- For this computation, there is a corresponding computation of machine $\mathcal{M}'$ consisting of configurations

$$\beta_0 \longrightarrow \beta_1 \longrightarrow \beta_2 \longrightarrow \cdots$$

  where for every configuration α_i there is some corresponding configuration $\beta_{f(i)}$ where $f : \mathbb{N} \to \mathbb{N}$ is a function, for which $f(i) \leq f(j)$ for every i and j where $i < j$.
- There is a relation between configurations of machine $\mathcal{M}$ to configurations of machine $\mathcal{M}'$ that correspond to them.
- There are functions mapping an input w to corresponding initial configurations α_0 and β_0 and analogously functions mapping final configurations to a result of computation.

Simulation of a Computation


$$\begin{tikzcd}
\mathcal{M} & \mathcal{M}' \\
\alpha_0 \arrow[d] \arrow[r, leftrightarrow] & \beta_0 \arrow[d] \\
& \beta_1 \arrow[d] \arrow[dl, leftrightarrow] \\
\alpha_1 \arrow[d] & \beta_2 \arrow[d] \\
& \beta_3 \arrow[d] \arrow[dl, leftrightarrow] \\
\alpha_2 \arrow[d] & \beta_4 \arrow[d] \\
& \beta_5 \arrow[d] \\
\alpha_3 \arrow[d] \arrow[dr, leftrightarrow] & \beta_6 \arrow[d] \\
& \beta_7 \arrow[d] \\
\alpha_4 \arrow[r, leftrightarrow] & \beta_8
\end{tikzcd}$$

Models of Computation

Some models of computation are weaker (finite automata, pushdown automata, . . .) and they can not be used to implement an arbitrary algorithm.

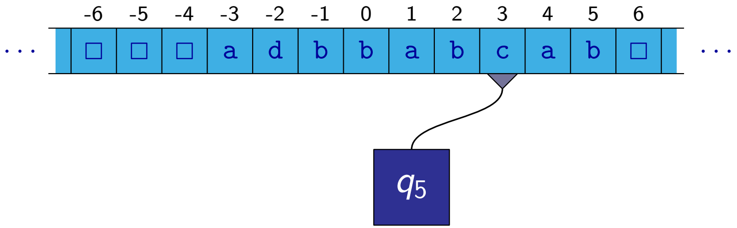
We will concentrate now on models of computation that are powerful enough to be able to execute arbitrary algorithm (for example such that can be represented as a program in some programming language).

Such models of computation are called **Turing-complete**:

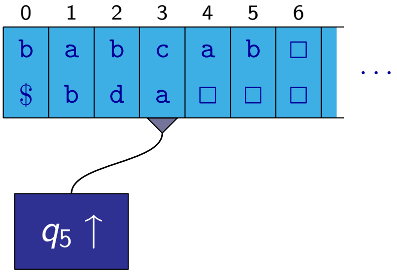
- they are able to simulate a behaviour of arbitrary Turing machine
- their bahaviour can be simulated by a Turing machine

Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

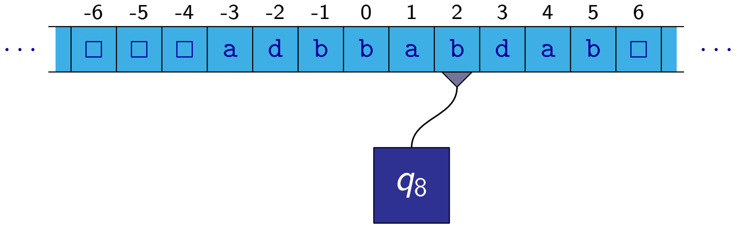


A tape infinite only on one side:

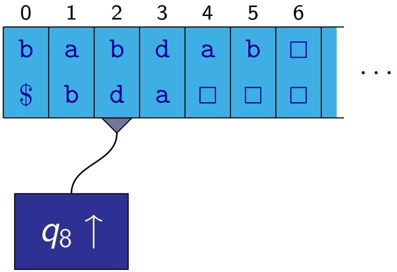


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

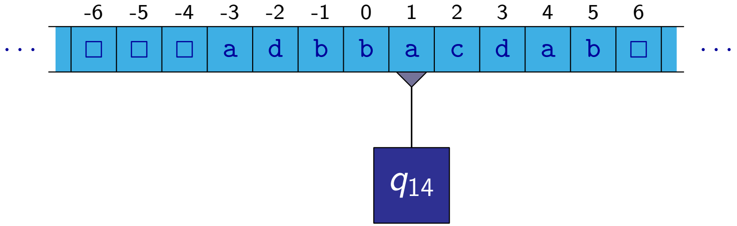


A tape infinite only on one side:

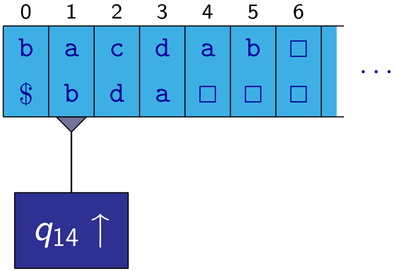


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

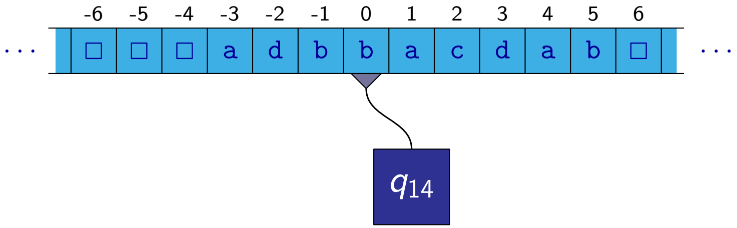


A tape infinite only on one side:

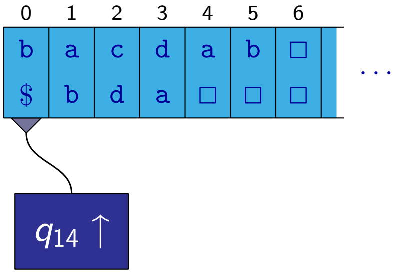


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

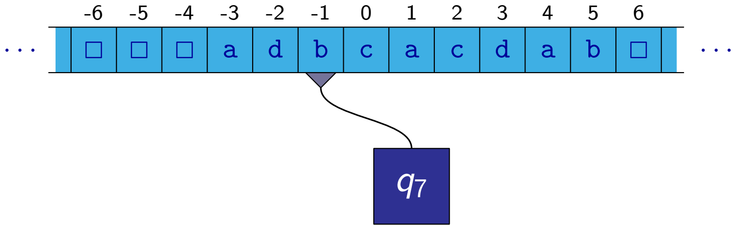


A tape infinite only on one side:

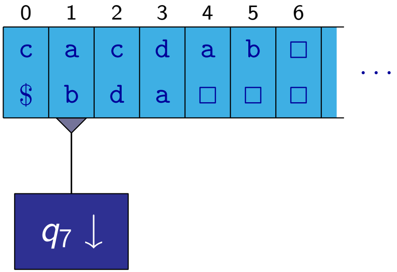


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

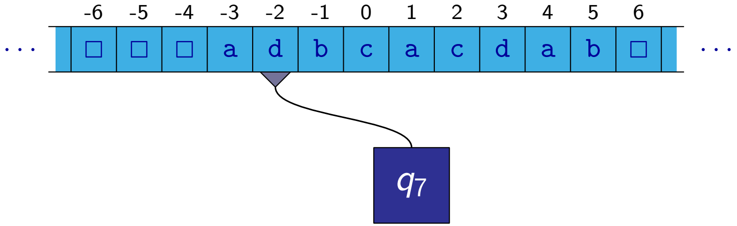


A tape infinite only on one side:

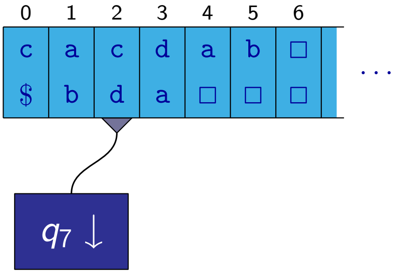


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

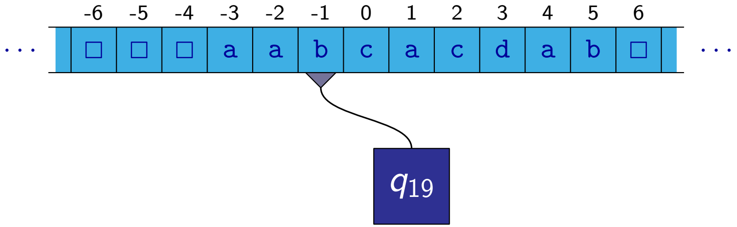


A tape infinite only on one side:

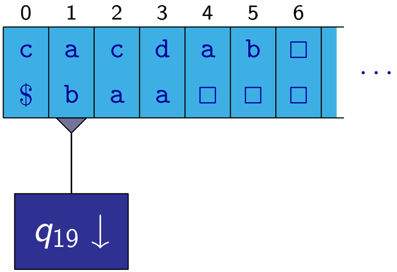


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

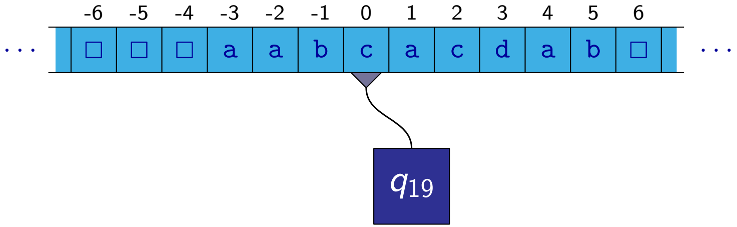


A tape infinite only on one side:

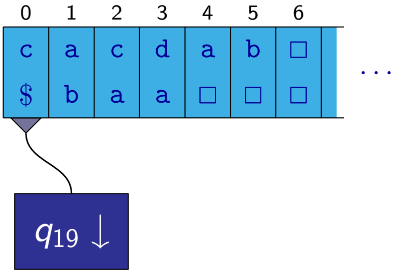


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

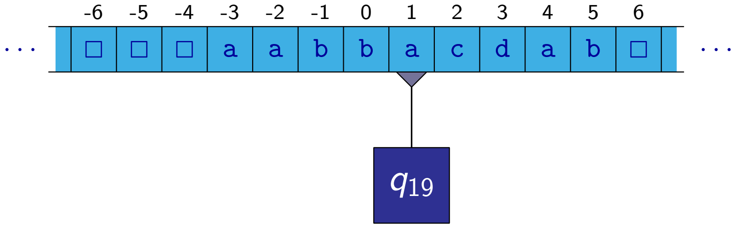


A tape infinite only on one side:

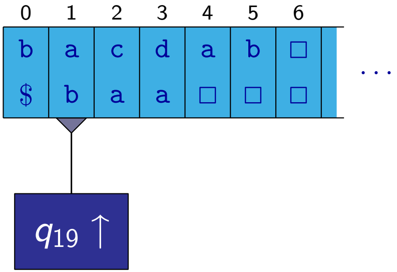


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:

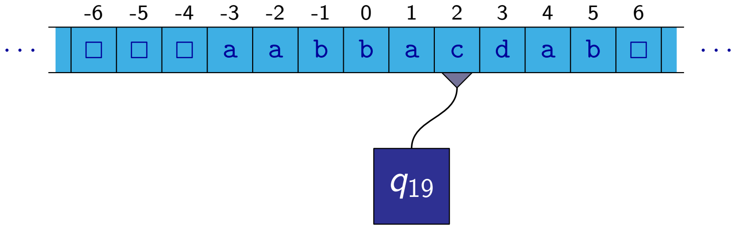


A tape infinite only on one side:

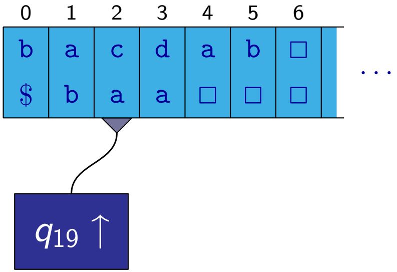


Two-Sided Infinite Tape by One-Sided Infinite Tape

A tape infinite on both sides:



A tape infinite only on one side:



Alphabet $\{0, 1\}$

A Turing machine with an arbitrary tape alphabet Γ can be simulated by a Turing machine with tape alphabet $\{0, 1\}$.

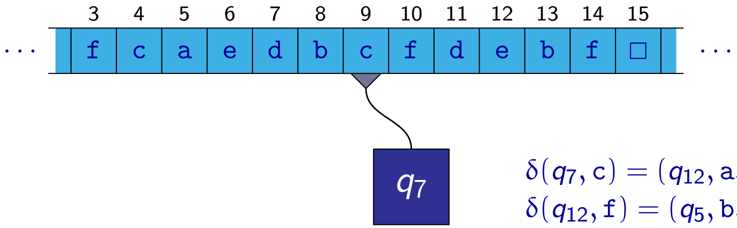
We can choose some appropriate encoding of symbols of alphabet Γ by k-bit sequences.

Example: Tape alphabet $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$\square$	$\leftrightarrow$	000
a	$\leftrightarrow$	001
b	$\leftrightarrow$	010
c	$\leftrightarrow$	011
d	$\leftrightarrow$	100
e	$\leftrightarrow$	101
f	$\leftrightarrow$	110
g	$\leftrightarrow$	111

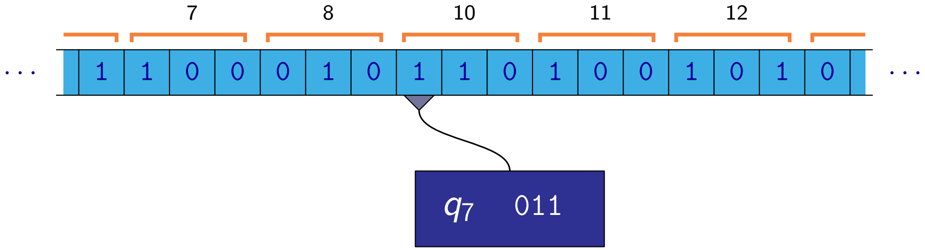
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



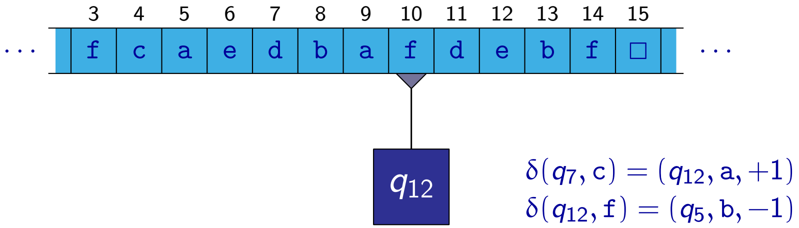
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

The corresponding machine with alphabet $\{0, 1\}$:

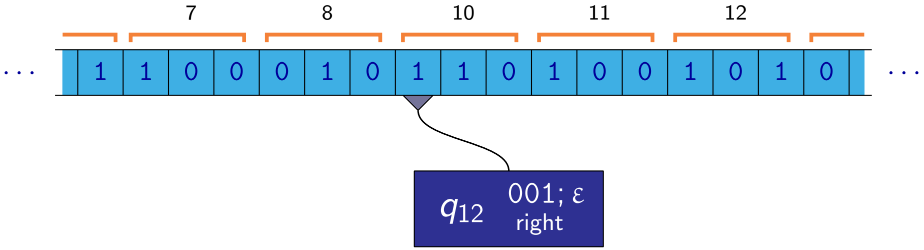


Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:

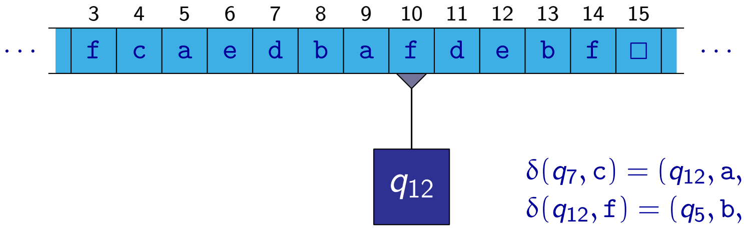

3
4
5
6
7
8
9
10
11
12
13
14
15
f c a e d b a f d e b f □
q_{12}
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



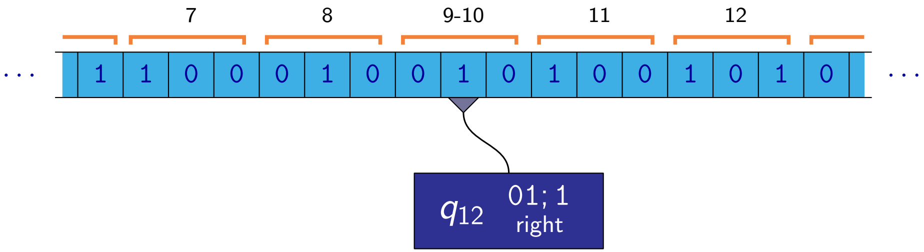
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



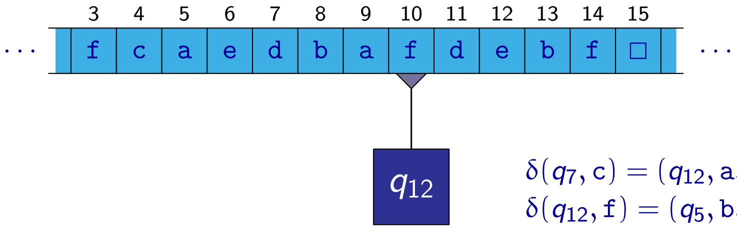
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



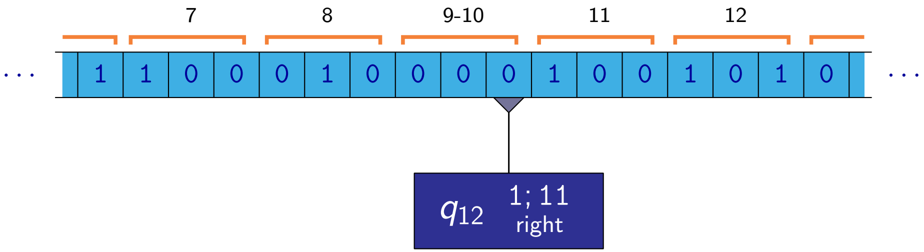
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



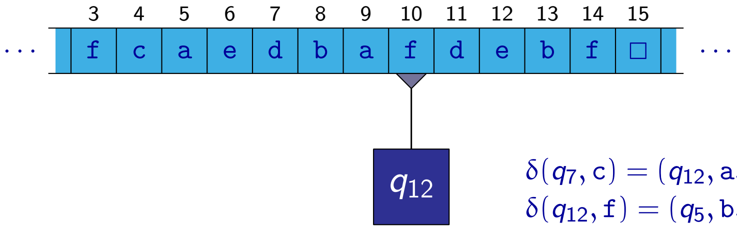
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



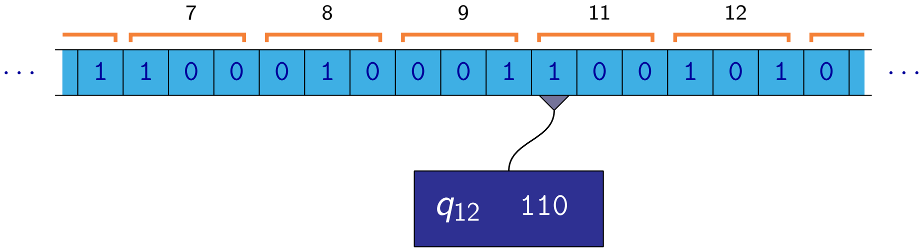
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



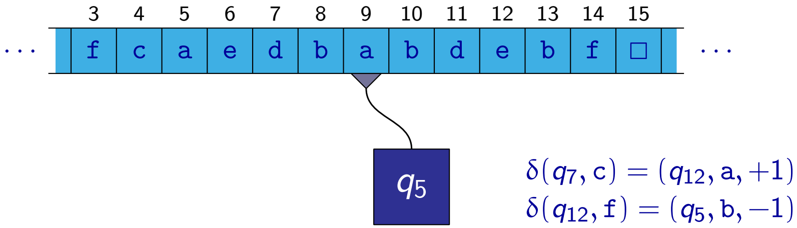
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:

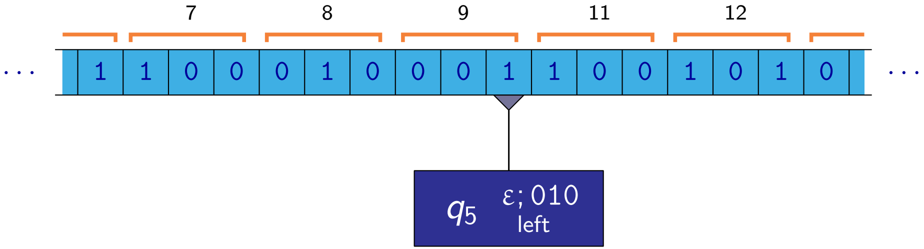


Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:

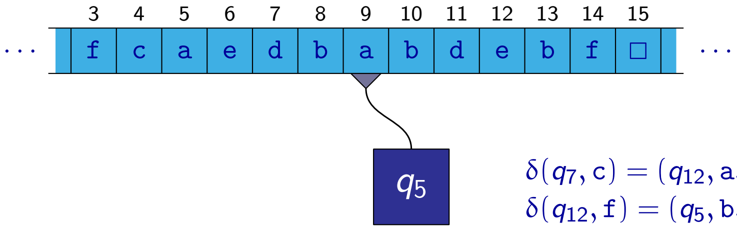

3 4 5 6 7 8 9 10 11 12 13 14 15
f c a e d b a b d e b f □
q_5
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



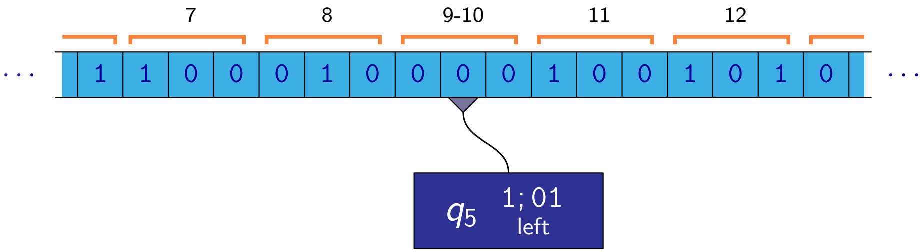
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



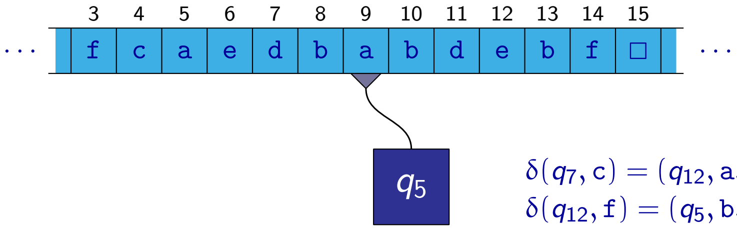
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



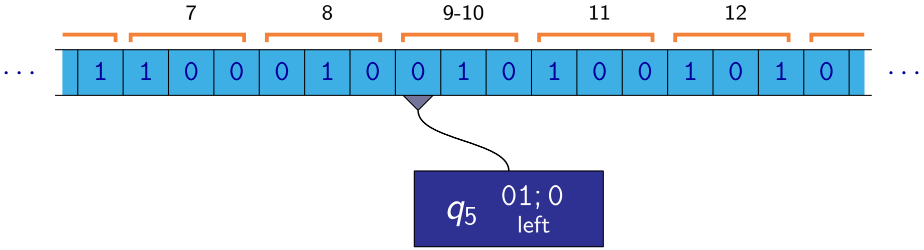
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



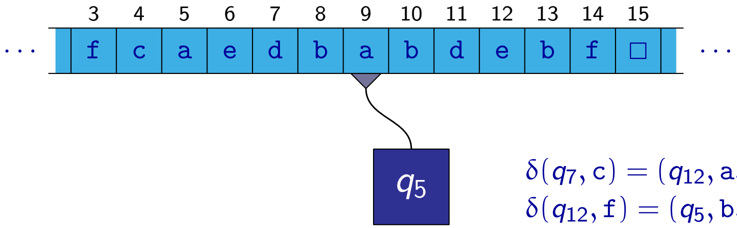
$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



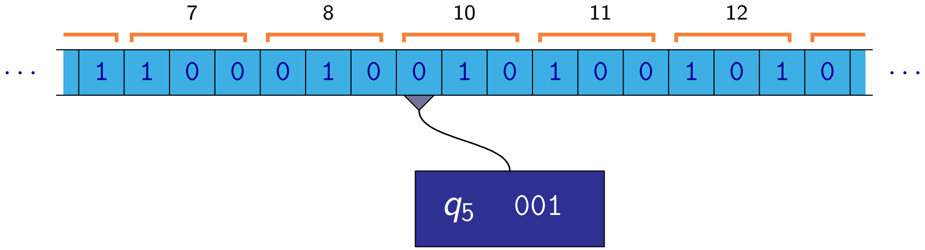
Alphabet $\{0, 1\}$

A machine with tape alphabet Γ:



$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

The corresponding machine with alphabet $\{0, 1\}$:



Alphabet $\{0, 1\}$

In this simulation, each step of the original machine is simulated by $k + 1$ steps where k is the number of bits used for encoding of one symbol of alphabet Γ.

So if the original machine performs t steps in a computation, the simulating machine performs $O(t)$ steps.

Decreasing the number of states of the control unit

Remark: Similarly, as is possible to decrease the tape alphabet to only two symbols by increasing the number of states of the control unit, it is also possible to decrease the number of states of the control unit:

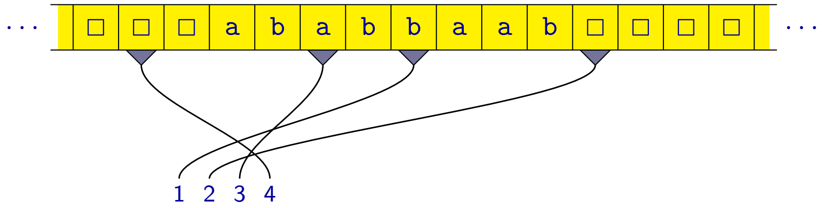
- An arbitrary Turing machine can be simulated by a Turing machine with only two non-final states of its control unit (and possibly with some final states). However, this simulation requires increase in the size of the tape alphabet.

Similarly as in the previous case, one step of the original machine is simulated by s steps where s is a constant depending only on the number of the states of the control unit of the original machines (i.e., the size of set Q).

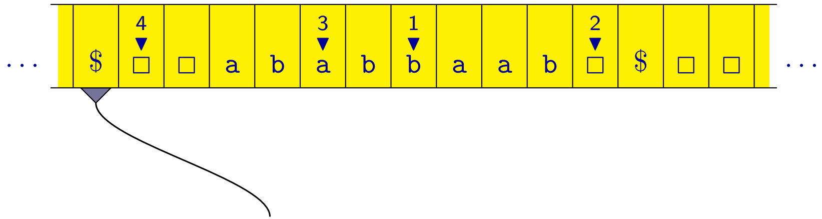
So as before, if the original machine performs t steps, the simulating machine performs $O(t)$ steps.

Simulation of several heads on a tape with one head

Several heads on a tape:

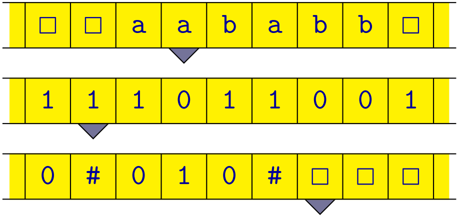


A tape with one head:

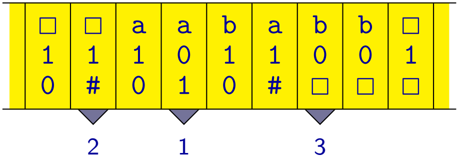


Simulation of several tapes with one tape

Several tapes:

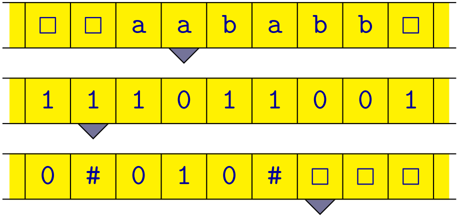


One tape with several heads:

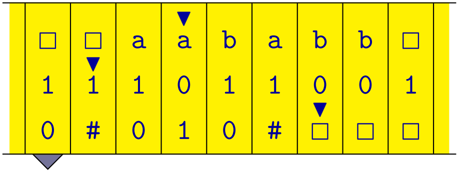


Simulation of several tapes with one tape

Several tapes:

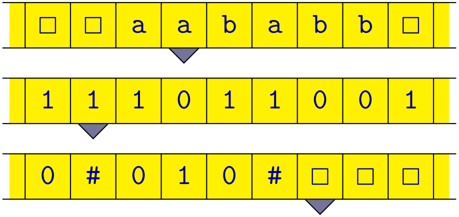


One tape with one head: the variant where where marks on the tape are moved

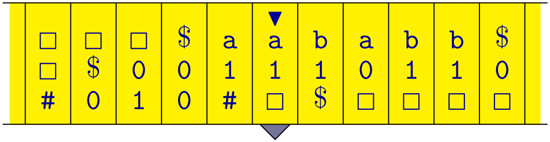


Simulation of several tapes with one tape

Several tapes:



One tape with one head: the variant where the content of tapes is moved



Tapes, stacks, and counters

We consider different types of machines that have a finite control unit equipped with some sort of memory of unbounded size.

Such memory can constist of one of more structures such as:

- **Tape** — reading and writing a symbol on a current position, movement of the head to the left and to the right

  Remark: The tape can be infinite of one side or on both sides.

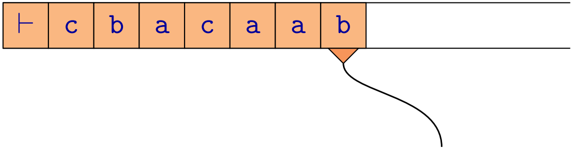
- **Stack** — push, pop, a test of emptiness of the stack

- **Counter** — a value is a natural number, operations of incrementing and decrementing by one, a test whether the value is equal to zero

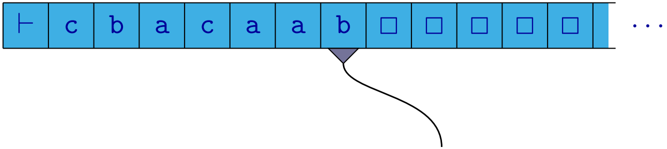
Stack

A stack can be viewed as a special case of a tape, which is infinite on one side.

Stack:



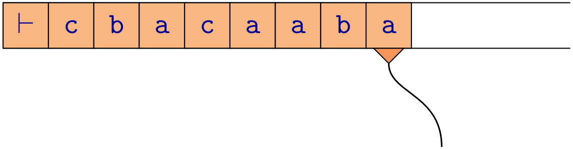
Tape:



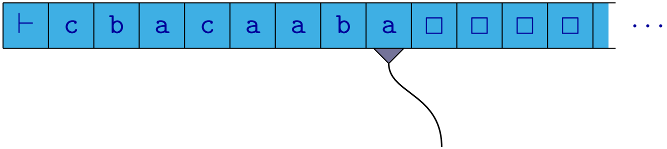
Stack

A stack can be viewed as a special case of a tape, which is infinite on one side.

Stack:



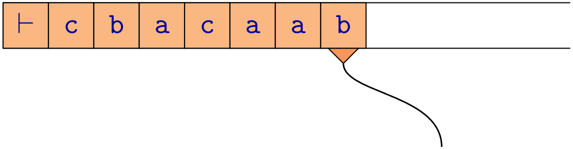
Tape:



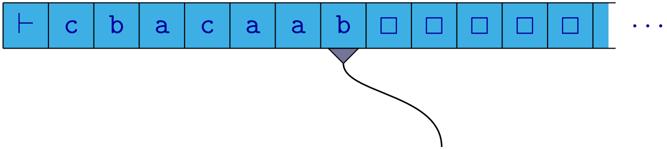
Stack

A stack can be viewed as a special case of a tape, which is infinite on one side.

Stack:



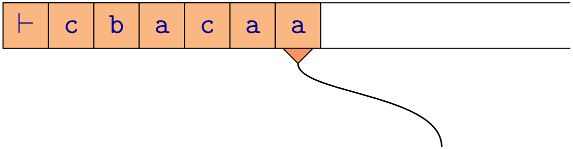
Tape:



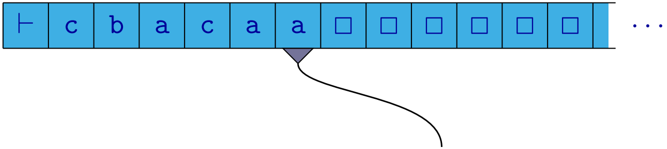
Stack

A stack can be viewed as a special case of a tape, which is infinite on one side.

Stack:



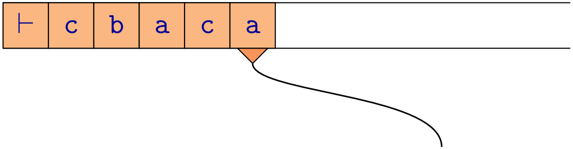
Tape:



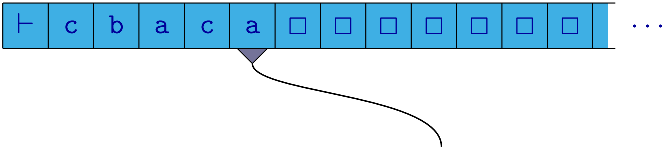
Stack

A stack can be viewed as a special case of a tape, which is infinite on one side.

Stack:

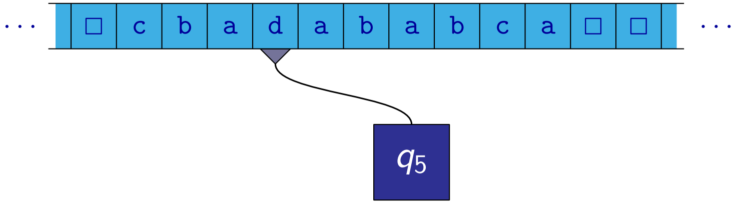


Tape:

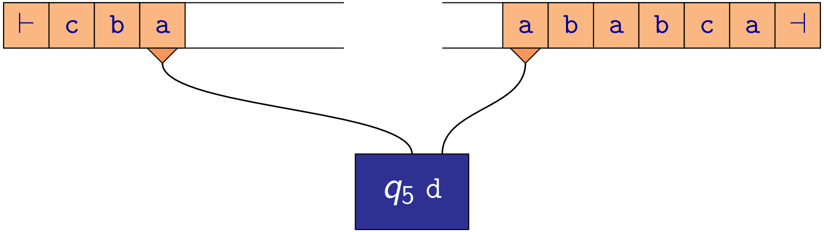


Stack

A tape, infinite on both sides, can be simulated by two stacks:

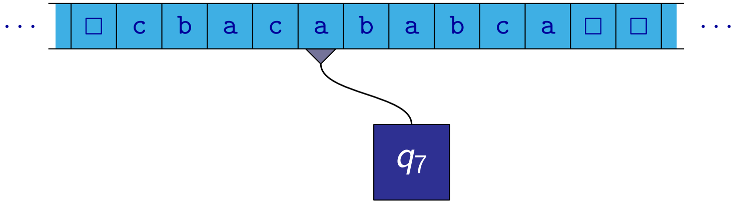


A machine with two stacks:

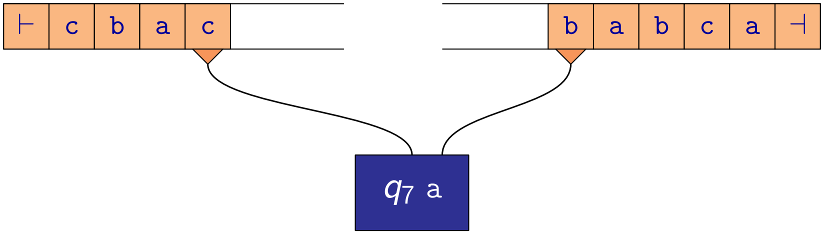


Stack

A tape, infinite on both sides, can be simulated by two stacks:



A machine with two stacks:



Counter

Counter — a value of a counter can be an arbitrarily big natural number, i.e., an element of the set $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

Basic operations:

- incrementing the value by one:

$$x := x + 1$$

- decrementing the value by one:

$$x := x - 1$$

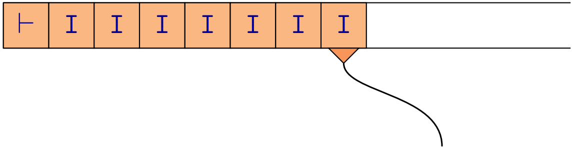
- test whether the value of the counter is zero:

$$\textbf{if } (x = 0) \textbf{ goto } \ell$$

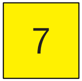
Counter

A counter can be viewed as a special case of a stack or of a tape.

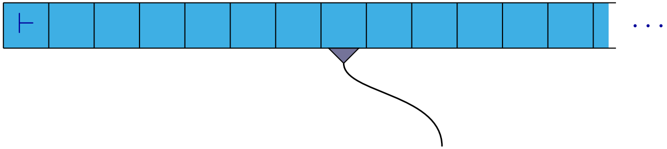
Stack:



Counter:

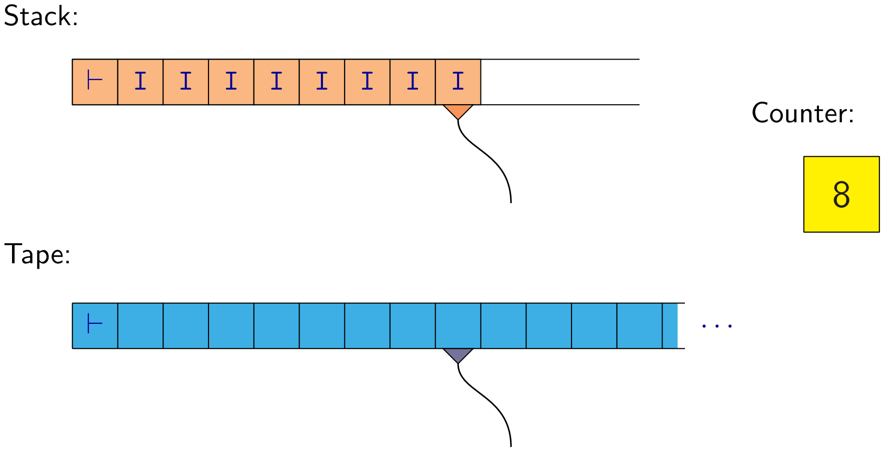


Tape:



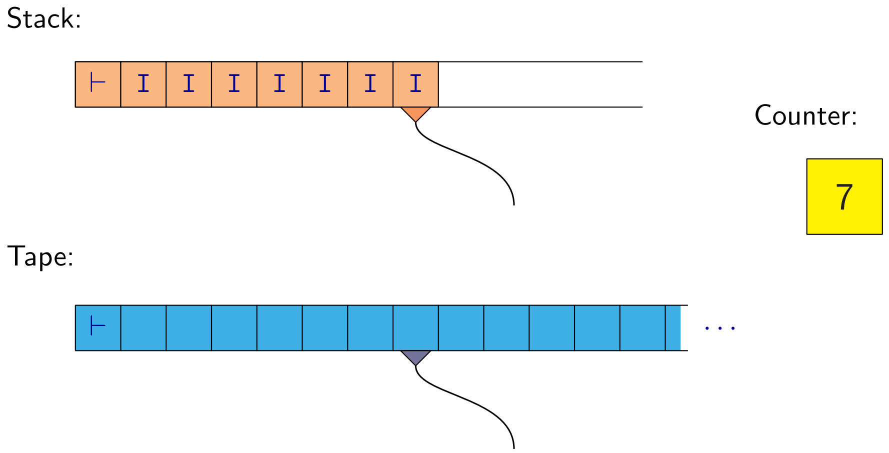
Counter

A counter can be viewed as a special case of a stack or of a tape.



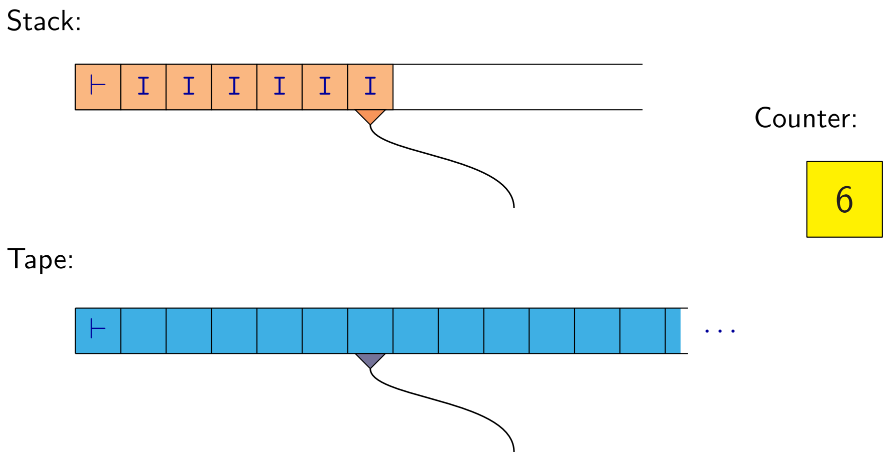
Counter

A counter can be viewed as a special case of a stack or of a tape.



Counter

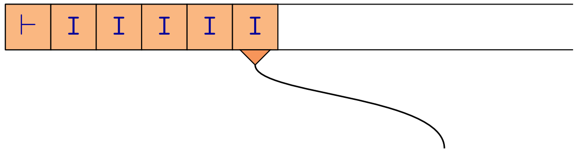
A counter can be viewed as a special case of a stack or of a tape.



Counter

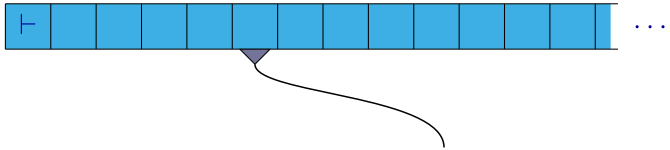
A counter can be viewed as a special case of a stack or of a tape.

Stack:



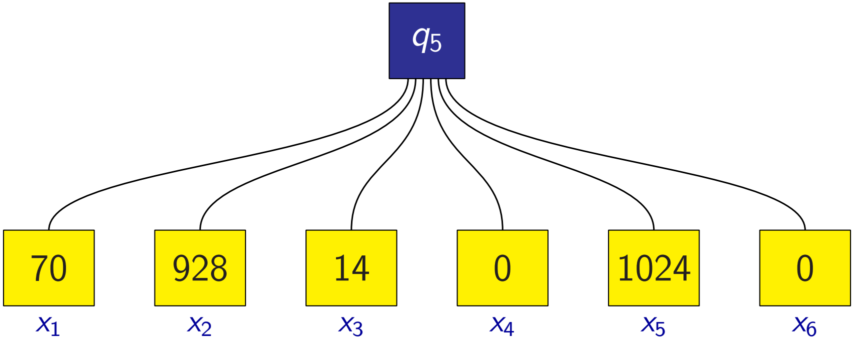
Counter:

Tape:



Minsky machine

Minsky machine — a machine with a finite control unit and a finite set of counters $x_1, x_2, \ldots, x_k$:



Remark: In addition to symbols $x_1, x_2, \ldots$, we can also use symbols such as $x, y, z, \ldots$ to denote counters.

Minsky machine

A Minsky machine can be viewed as a program consisting of a sequence of instructions, with the following five types of instructions:

- incrementing the value of a given counter by one:

$$x_i := x_i + 1$$

- decrementing the value of a given counter by one:

$$x_i := x_i - 1$$

- test whether the value of a given counter is zero:

$$\textbf{if } (x_i = 0) \textbf{ goto } \ell$$

- unconditional jump:

$$\textbf{goto } \ell$$

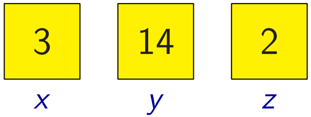
- halting of the computation of the program:

halt

Minsky machine

Setting the counter x to zero:

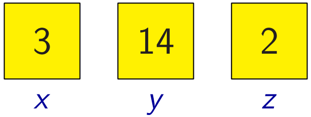
```
➡ $L_1$ : if $(x = 0)$ goto $L_2$
          $x := x - 1$
          goto $L_1$
   $L_2$ :   ...
```



Minsky machine

Setting the counter x to zero:

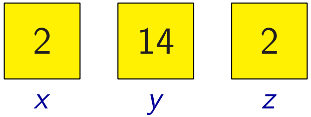
```
$L_1$ : if $(x = 0)$ goto $L_2$
        $x := x - 1$
        goto $L_1$
$L_2$ :   ...
```



Minsky machine

Setting the counter x to zero:

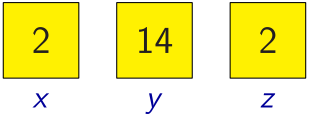
```
$L_1$ : if $(x = 0)$ goto $L_2$
        $x := x - 1$
→       goto $L_1$
$L_2$ : …
```



Minsky machine

Setting the counter x to zero:

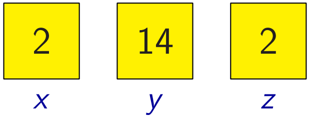
```
→ $L_1$ : if $(x = 0)$ goto $L_2$
          $x := x - 1$
          goto $L_1$
  $L_2$ :   $\dots$
```



Minsky machine

Setting the counter x to zero:

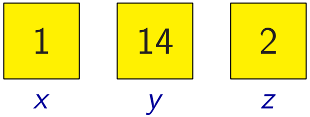
```
$L_1$ : if $(x = 0)$ goto $L_2$
→       $x := x - 1$
        goto $L_1$
$L_2$ :   ...
```



Minsky machine

Setting the counter x to zero:

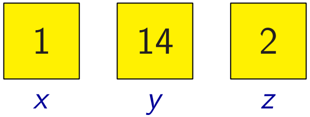
```
$L_1$ : if $(x = 0)$ goto $L_2$
        $x := x - 1$
→       goto $L_1$
$L_2$ :   ...
```



Minsky machine

Setting the counter x to zero:

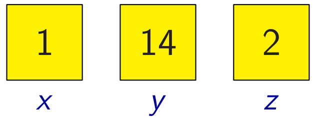
```
→ $L_1$ : if ($x = 0$) goto $L_2$
          $x := x - 1$
          goto $L_1$
  $L_2$ :   ...
```



Minsky machine

Setting the counter x to zero:

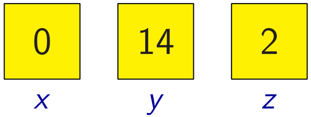
```
$L_1$ : if ($x = 0$) goto $L_2$
      $x := x - 1$
      goto $L_1$
$L_2$ :   ...
```



Minsky machine

Setting the counter x to zero:

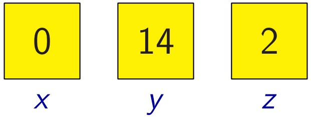
```
$L_1$ : if ($x = 0$) goto $L_2$
        $x := x - 1$
        goto $L_1$
$L_2$ :   …
```



Minsky machine

Setting the counter x to zero:

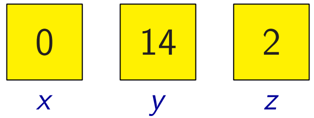
```
→   L_1 : if (x = 0) goto L_2
          x := x − 1
          goto L_1
    L_2 :  ...
```



Minsky machine

Setting the counter x to zero:

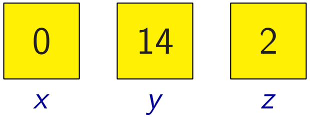
```
$L_1$ : if ($x = 0$) goto $L_2$
        $x := x - 1$
        goto $L_1$
$L_2$ :  ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

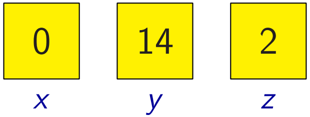
```
→   $L_2$ : if ($z = 0$) goto $L_3$
            $z := z - 1$
            $y := y + 1$
            goto $L_1$
    $L_3$ :   ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

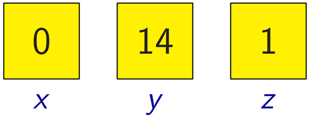
```
$L_2$ : if ($z = 0$) goto $L_3$
→       $z := z - 1$
        $y := y + 1$
        goto $L_1$
$L_3$ :   ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

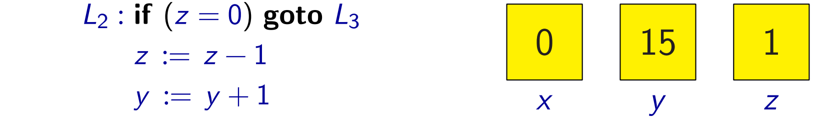
```
    $L_2$ : if ($z = 0$) goto $L_3$
            $z := z - 1$
→           $y := y + 1$
            goto $L_1$
    $L_3$ :  ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

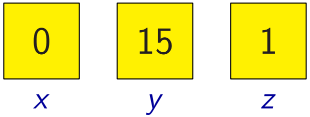
```
$L_2$ : if $(z = 0)$ goto $L_3$
        $z := z - 1$
        $y := y + 1$
        goto $L_1$
$L_3$ :   ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

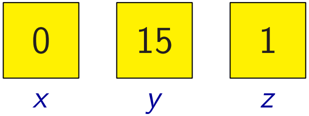
```
→  $L_2$ : if $(z = 0)$ goto $L_3$
          $z := z - 1$
          $y := y + 1$
          goto $L_1$
   $L_3$ :  …
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

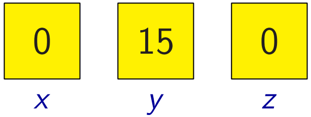
```
$L_2$ : if ($z = 0$) goto $L_3$
        $z := z - 1$
        $y := y + 1$
        goto $L_1$
$L_3$ :  ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

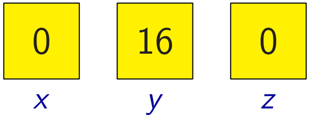
```
    $L_2$ : **if** $(z = 0)$ **goto** $L_3$
            $z := z - 1$
→           $y := y + 1$
            **goto** $L_1$
    $L_3$ :   $\ldots$
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

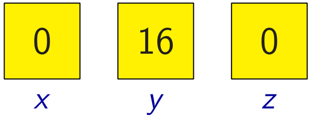
```
  $L_2$ : if $(z = 0)$ goto $L_3$
          $z := z - 1$
          $y := y + 1$
→         goto $L_1$
  $L_3$ :  $\dots$
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

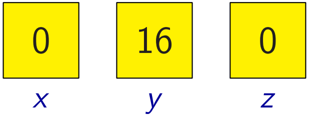
```
→  $L_2$ : if ($z = 0$) goto $L_3$
          $z := z - 1$
          $y := y + 1$
          goto $L_1$
   $L_3$ :  ...
```



Minsky machine

Adding the value of the counter z to the counter y (together with setting the counter z to zero):

```
    $L_2$ : if $(z = 0)$ goto $L_3$
          $z := z - 1$
          $y := y + 1$
          goto $L_1$
→   $L_3$ :  …
```



Minsky machine

Multiplying the value of counter x with constant 5:

```
$L_1$ : **if** $(x = 0)$ **goto** $L_2$
        $x := x - 1$
        $y := y + 1$
        $y := y + 1$
        $y := y + 1$
        $y := y + 1$
        $y := y + 1$
        **goto** $L_1$
$L_2$ : **if** $(y = 0)$ **goto** $L_3$
        $y := y - 1$
        $x := x + 1$
        **goto** $L_2$
$L_3$ :   ...
```

Minsky machine

Division of the value of the counter x with constant 5 and finding out the remainder after this division:

```
$L_1$ : if ($x = 0$) goto $M_0$
     $x := x - 1$
     if ($x = 0$) goto $M_1$
     $x := x - 1$
     if ($x = 0$) goto $M_2$
     $x := x - 1$
     if ($x = 0$) goto $M_3$
     $x := x - 1$
     if ($x = 0$) goto $M_4$
     $x := x - 1$
     $y := y + 1$
     goto $L_1$
```

Minsky machine

A stack can be simulated using a pair of counters — a value of the first counter represents the content of the stack as a number of base $k = |\Gamma| + 1$ (where Γ is a stack alphabet).

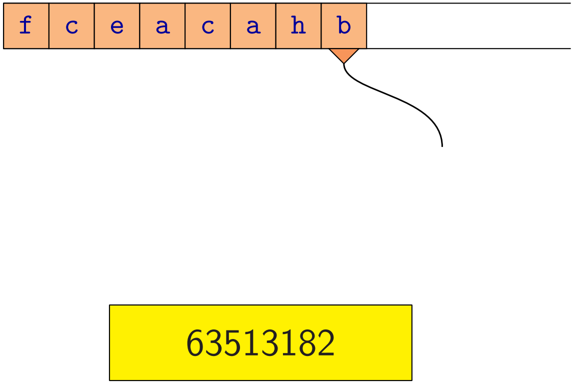
- A stack on the top of the stack — the remainder of division by k
- Pop — to divide by k
- Push — to multiply by k and to add the code of the given symbol

The second counter is used as an auxiliary counter for performing the above given operations.

Minsky machine

Example:

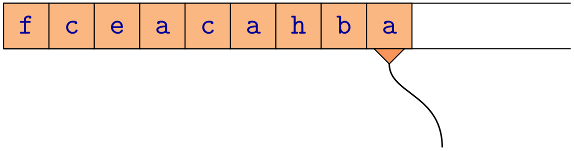
a	1
b	2
c	3
d	4
e	5
f	6
g	7
h	8
i	9



Minsky machine

Example:

a	$\leftrightarrow$	1
b	$\leftrightarrow$	2
c	$\leftrightarrow$	3
d	$\leftrightarrow$	4
e	$\leftrightarrow$	5
f	$\leftrightarrow$	6
g	$\leftrightarrow$	7
h	$\leftrightarrow$	8
i	$\leftrightarrow$	9

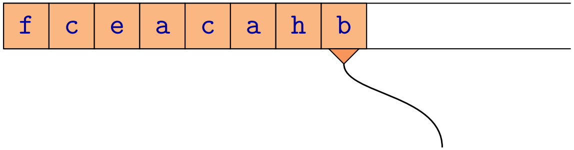


635131821

Minsky machine

Example:

a	1
b	2
c	3
d	4
e	5
f	6
g	7
h	8
i	9

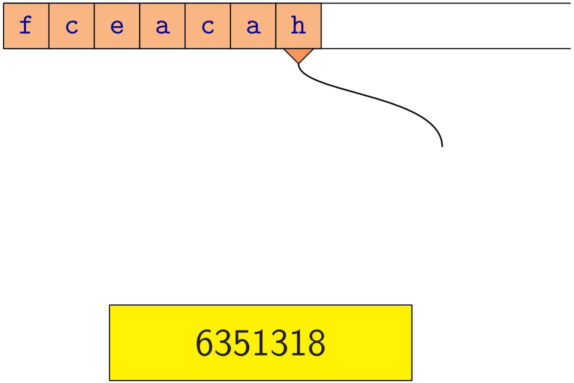


63513182

Minsky machine

Example:

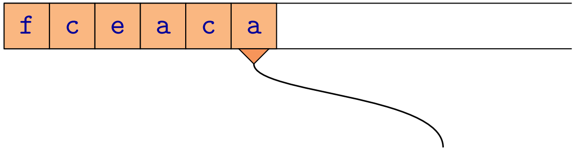
a	$\leftrightarrow$ 1
b	$\leftrightarrow$ 2
c	$\leftrightarrow$ 3
d	$\leftrightarrow$ 4
e	$\leftrightarrow$ 5
f	$\leftrightarrow$ 6
g	$\leftrightarrow$ 7
h	$\leftrightarrow$ 8
i	$\leftrightarrow$ 9



Minsky machine

Example:

a	$\leftrightarrow$ 1
b	$\leftrightarrow$ 2
c	$\leftrightarrow$ 3
d	$\leftrightarrow$ 4
e	$\leftrightarrow$ 5
f	$\leftrightarrow$ 6
g	$\leftrightarrow$ 7
h	$\leftrightarrow$ 8
i	$\leftrightarrow$ 9



635131

Minsky machine

Recall that a tape infinite of both sides can be simulated by a pair of stacks.

In a Minsky machine, the content of each of these stacks can be represented by a corresponding counter.

We also need one additional counter for the implementation of multiplication and division by a constant on these counters representing contents of the stacks.

We can see that a Turing machine with k tapes can be simulated by a Minsky machine with $2k + 1$ counters.

Minsky machine

Any finite number of counters can be simulated by two counters.

- One counter (let it be denoted as C) represents values of all counters — e.g., values of three counters x, y, z can be represented in the counter C by the number $2^x 3^y 5^z$.
- The second counter is used as an auxiliary counter to perform operations of multiplication and division on counter C.
- Incrementing counter x by one is simulated as multiplying by 2, incrementing counter y by one is simulated as multiplying by 3, etc.
- In a similar way, decrementing counter x by one is simulated by division of counter C by number 2, decrementing counter y by one by division by number 3, etc.
- The test if $x = 0$ corrensponds to test if the value of counter C is divisible by 2, etc.

Minsky machine

We can see that computation of an arbitrary Turing machine can be simulated by a Minsky machine with two counters.

This simulation is extremely inefficient:

- Already simulation of a tape of a Turing machine by three counters requires number of steps that is exponentially bigger than the number of steps performed by this Turing machine.
- Simulation of these three counters using only two counters farther exponentially increases the performed number of steps.

Random Access Machines

Random Access Machine

A **Random Access Machine** (**RAM**) is an idealized model of a computer.

It consists of the following parts:

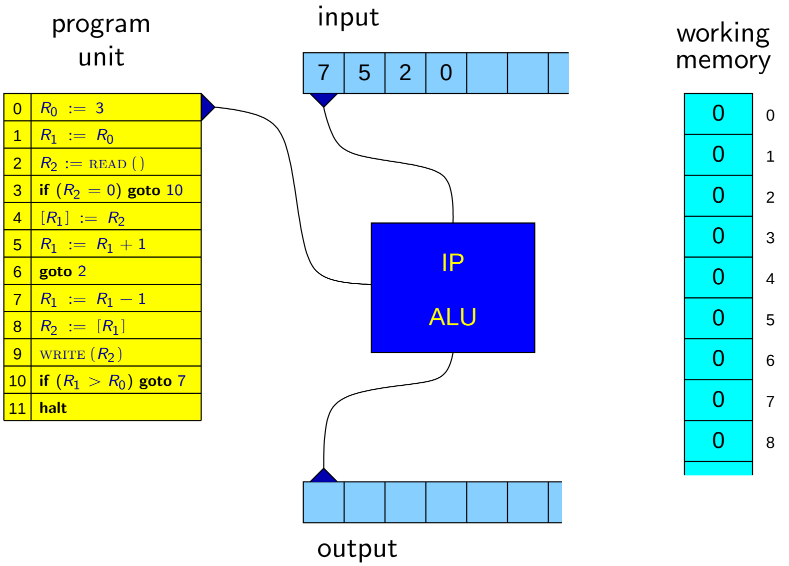
- **Program unit** – contains a program for the RAM and a pointer to the currently executed instruction
- **Working memory** consists of cells numbered $0, 1, 2, \ldots$

  These cells will be denoted $R_0, R_1, R_2, \ldots$

  The content of the cells can be read and written to.
- **Input tape** – read-only
- **Output tape** – write-only

The cells of memory, as well as the cells of input and output tapes contain integers (i.e., elements of set $\mathbb{Z}$) as their values.

Random Access Machine



0	$R_0 := 3$
1	$R_1 := R_0$
2	$R_2 := \text{READ}()$
3	**if** $(R_2 = 0)$ **goto** 10
4	$[R_1] := R_2$
5	$R_1 := R_1 + 1$
6	**goto** 2
7	$R_1 := R_1 - 1$
8	$R_2 := [R_1]$
9	$\text{WRITE}(R_2)$
10	**if** $(R_1 > R_0)$ **goto** 7
11	**halt**

Random Access Machine

Overview of instructions:

$R_i := c$	– assignment of a constant
$R_i := R_j$	– assignment
$R_i := [R_j]$	– load (reading from memory)
$[R_i] := R_j$	– store (writing to memory)
$R_i := R_j \ op\ R_k$ or $R_i := R_j \ op\ c$	– arithmetic instructions, $op \in \{+, -, *, /\}$
if $(R_i \ rel\ R_j)$ **goto** ℓ or **if** $(R_i \ rel\ c)$ **goto** ℓ	– conditional jump, $rel \in \{=, \neq, \leq, \geq, <, >\}$
goto ℓ	– unconditional jump
$R_i := \textsc{read}()$	– reading from input
$\textsc{write}(R_i)$	– writing to output
halt	– program termination

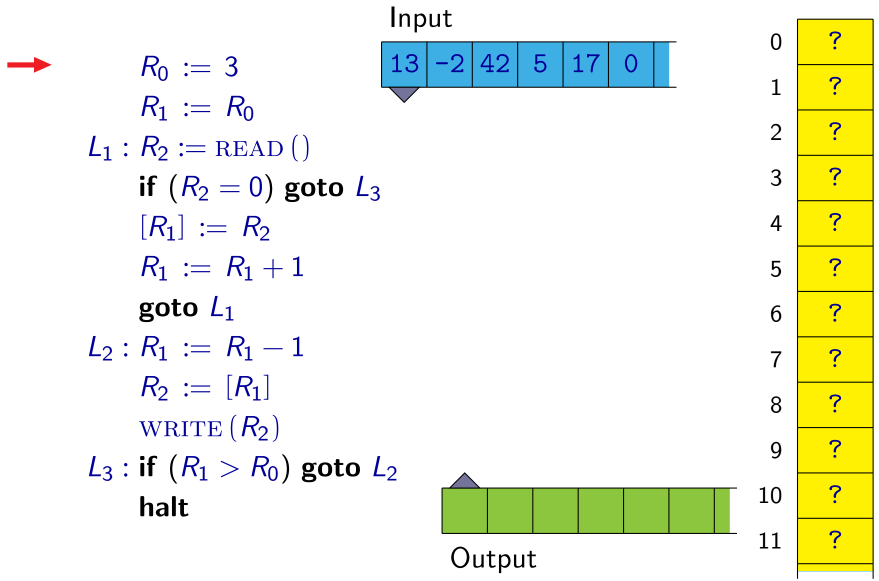
Random Access Machine

Examples of instructions:

$R_5 := 42$	– assignment of a constant
$R_{12} := R_3$	– assignment
$R_8 := [R_2]$	– load (reading from memory)
$[R_{15}] := R_9$	– store (writing to memory)
$R_7 := R_3 + R_6$	– arithmetic instruction
$R_{18} := R_{18} - 1$	– arithmetic instruction
if $(R_4 \geq R_1)$ **goto** 2801	– conditional jump
if $(R_2 \neq 0)$ **goto** 3581	– conditional jump
goto 537	– unconditional jump
$R_{23} := \textsc{read}\,()$	– reading from input
$\textsc{write}\,(R_{17})$	– writing to output
halt	– program termination

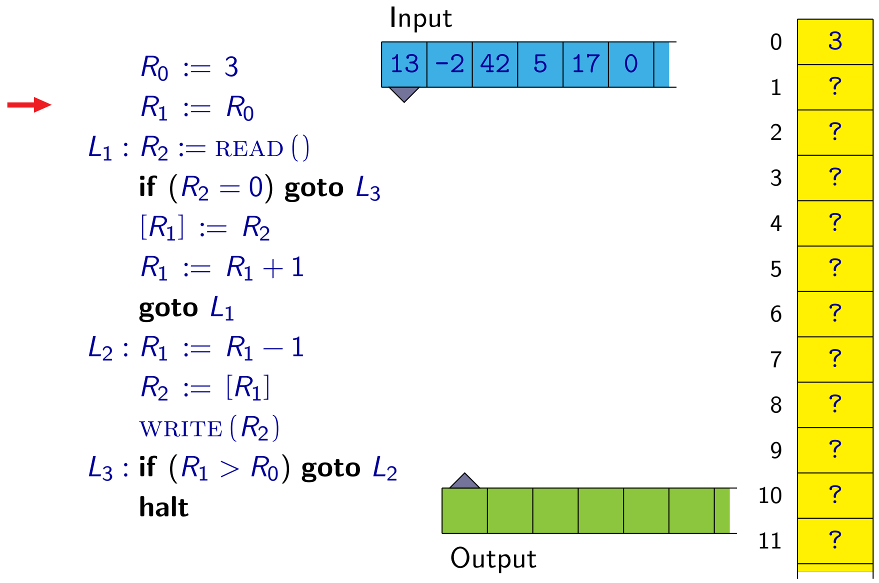
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
   $L_1$: $R_2 :=$ READ ( )
         if $(R_2 = 0)$ goto $L_3$
         $[R_1] := R_2$
         $R_1 := R_1 + 1$
         goto $L_1$
   $L_2$: $R_1 := R_1 - 1$
         $R_2 := [R_1]$
         WRITE $(R_2)$
   $L_3$: if $(R_1 > R_0)$ goto $L_2$
         halt
```



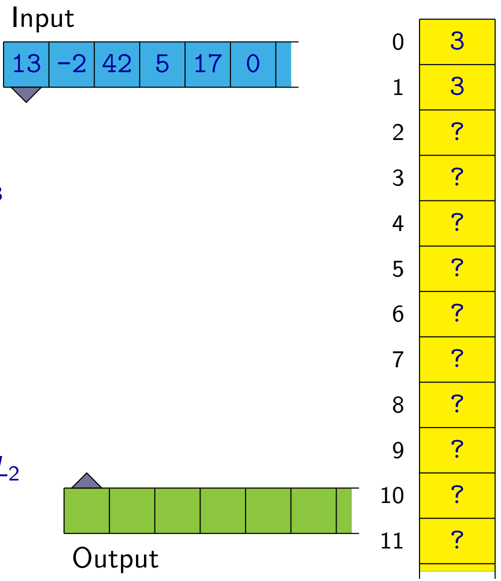
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



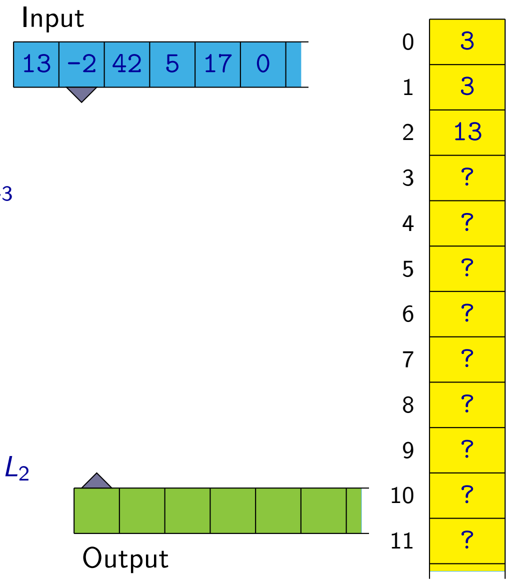
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
→ $L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
  $L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



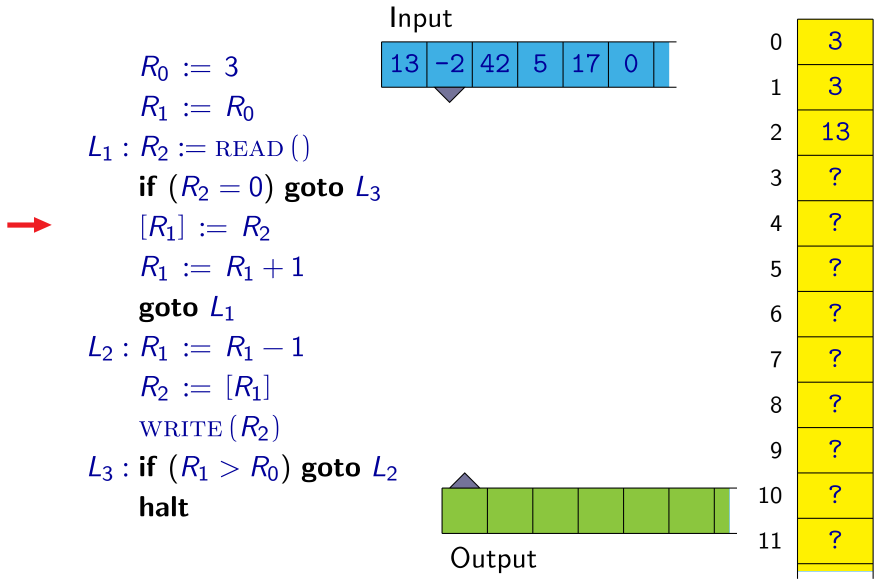
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       if ($R_2 = 0$) goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE ($R_2$)
$L_3$: if ($R_1 > R_0$) goto $L_2$
       halt
```



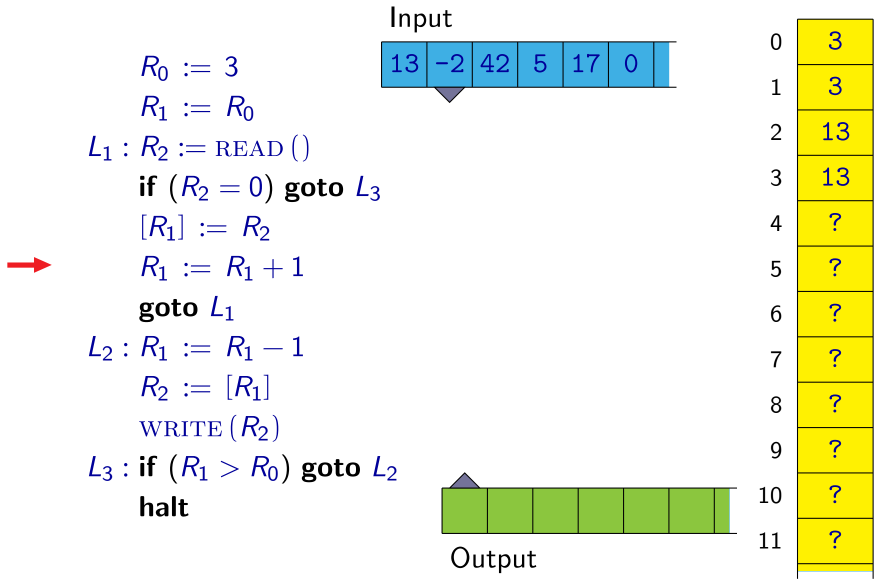
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



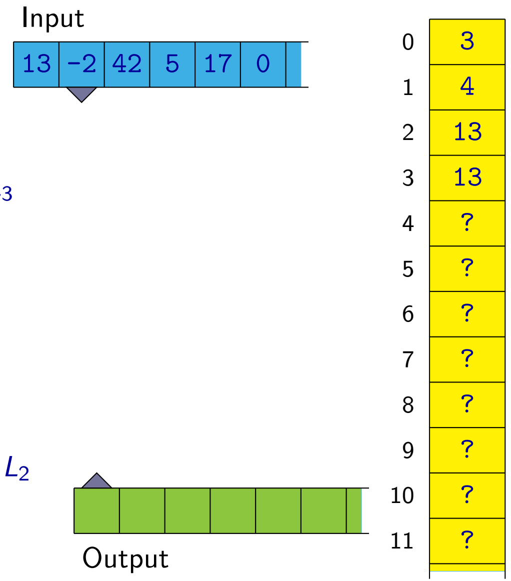
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
$L_1$ :   $R_2 :=$ READ ( )
          **if** $(R_2 = 0)$ **goto** $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          **goto** $L_1$
$L_2$ :   $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE ($R_2$)
$L_3$ :   **if** $(R_1 > R_0)$ **goto** $L_2$
          **halt**
```



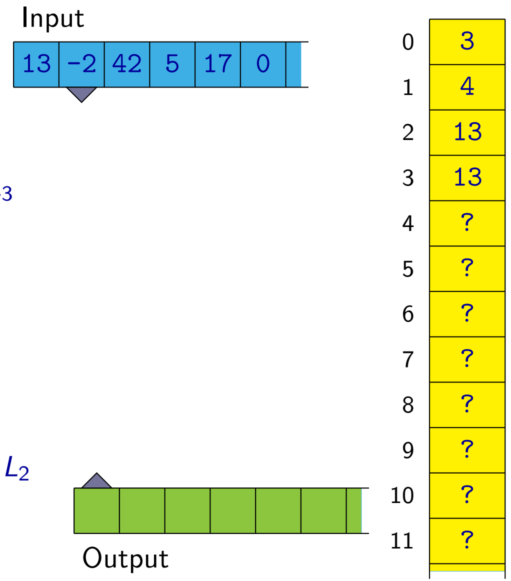
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ( )
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
→     **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



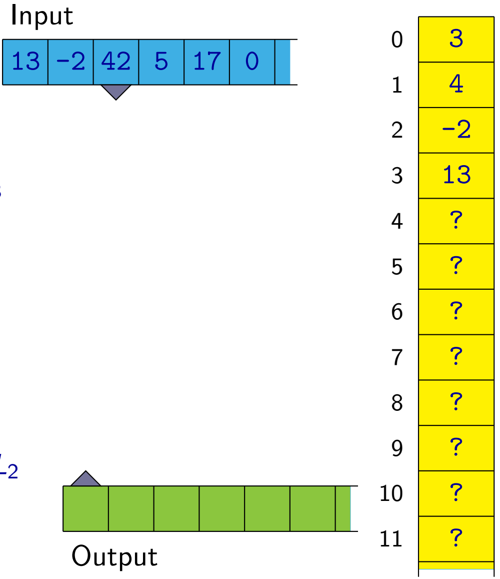
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
→ $L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
  $L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



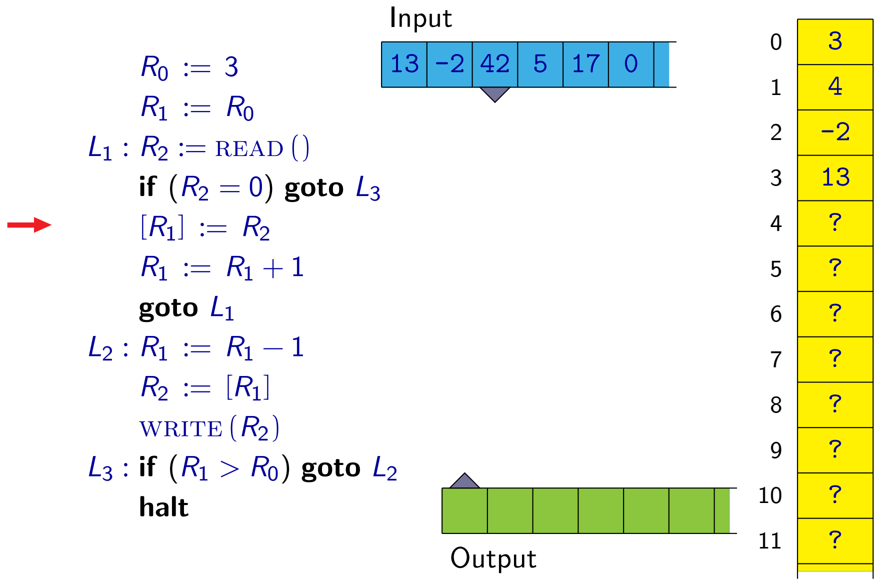
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
→       **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



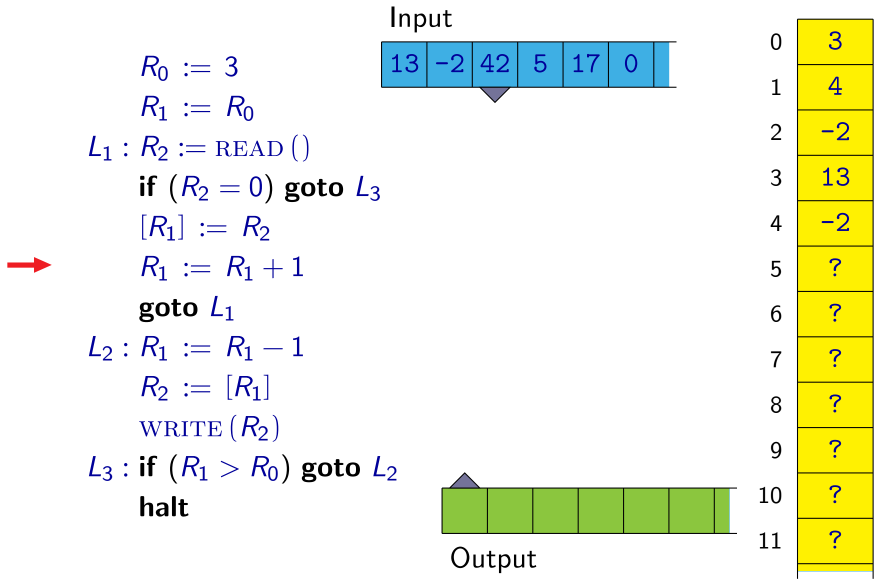
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



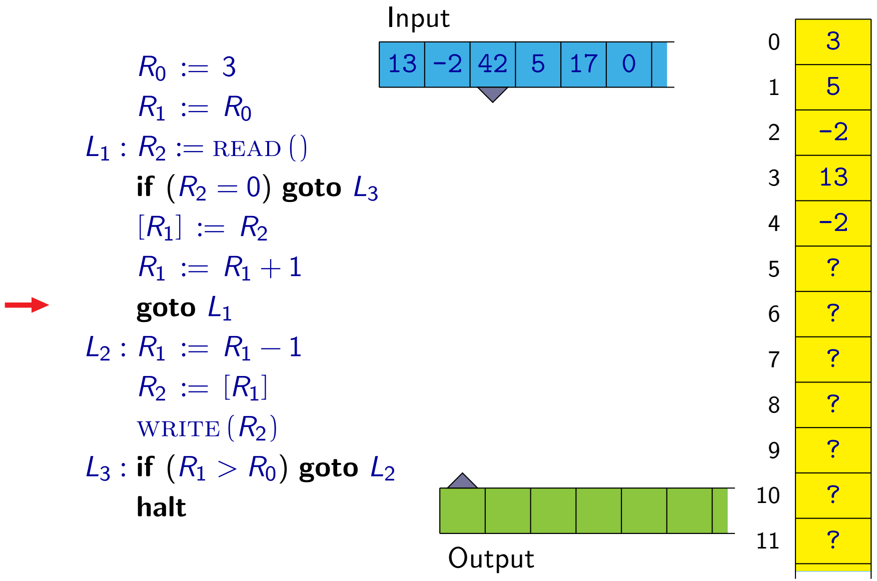
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ( )
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



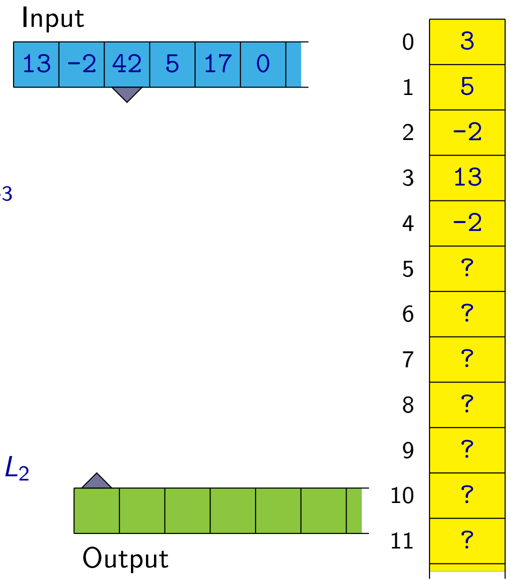
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       if ($R_2 = 0$) goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE ($R_2$)
$L_3$: if ($R_1 > R_0$) goto $L_2$
       halt
```



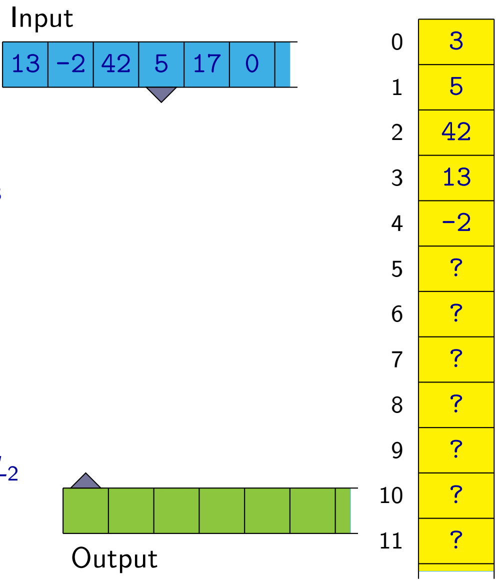
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
→  $L_1$ : $R_2 :=$ READ ()
         if $(R_2 = 0)$ goto $L_3$
         $[R_1] := R_2$
         $R_1 := R_1 + 1$
         goto $L_1$
   $L_2$ : $R_1 := R_1 - 1$
         $R_2 := [R_1]$
         WRITE $(R_2)$
   $L_3$ : if $(R_1 > R_0)$ goto $L_2$
         halt
```



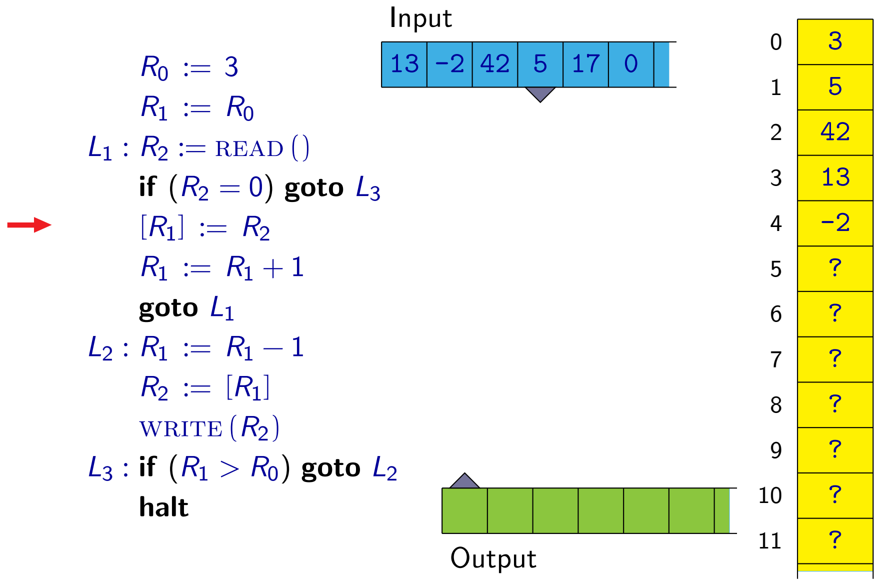
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



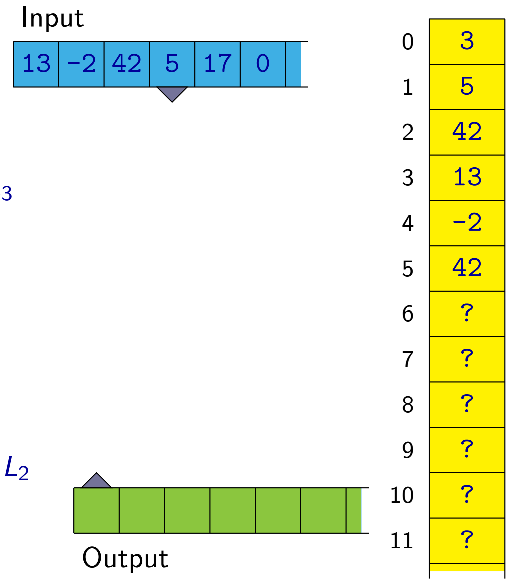
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



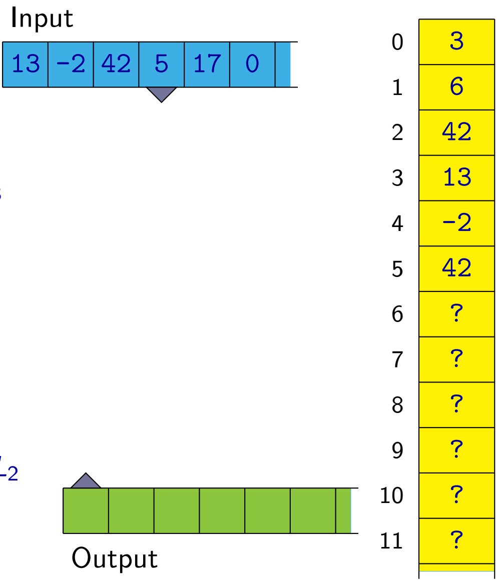
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
→       $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



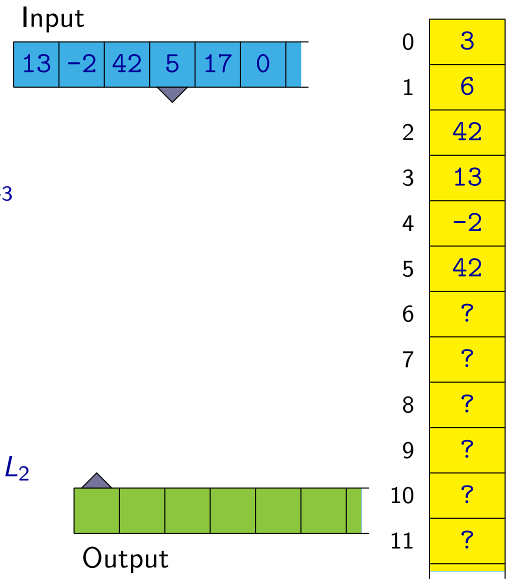
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
→       goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



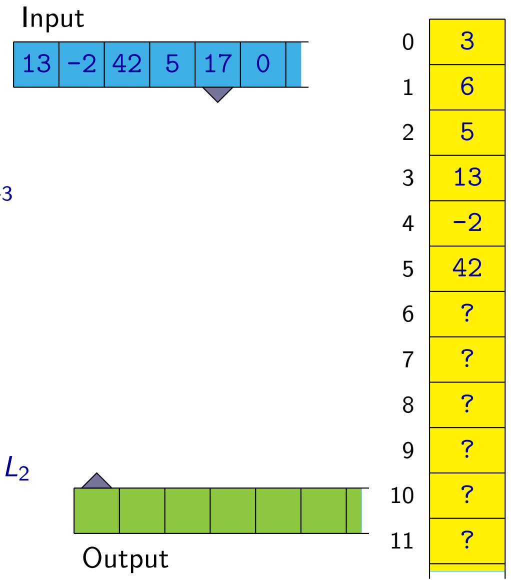
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      if ($R_2 = 0$) goto $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
      halt
```



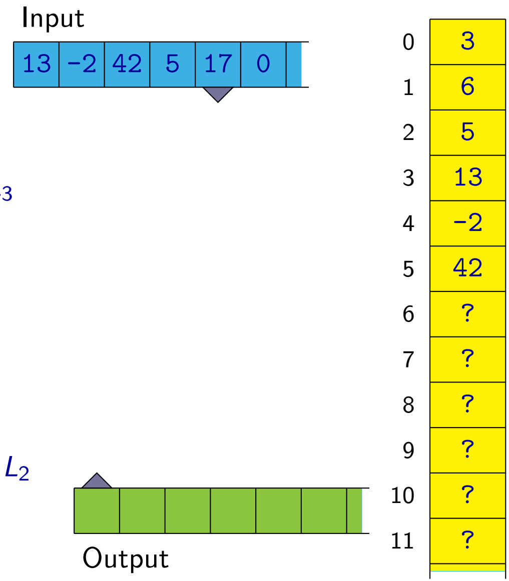
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



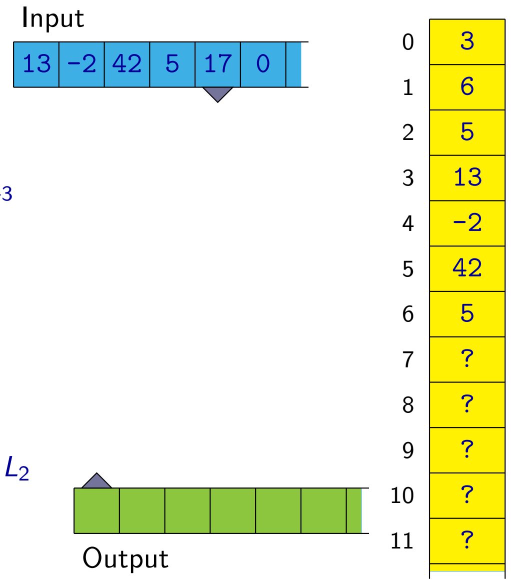
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ( )
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



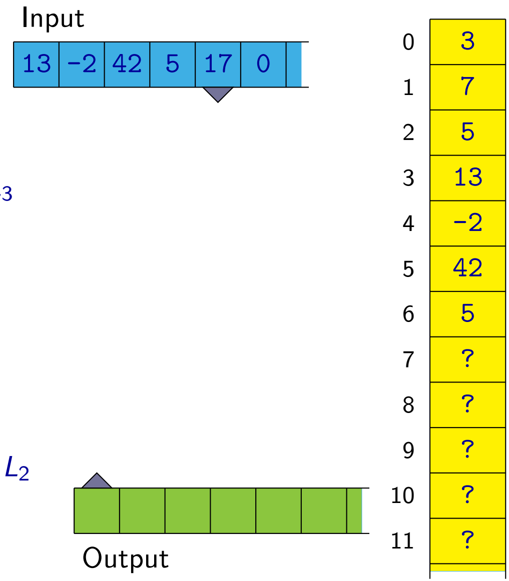
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
→       $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



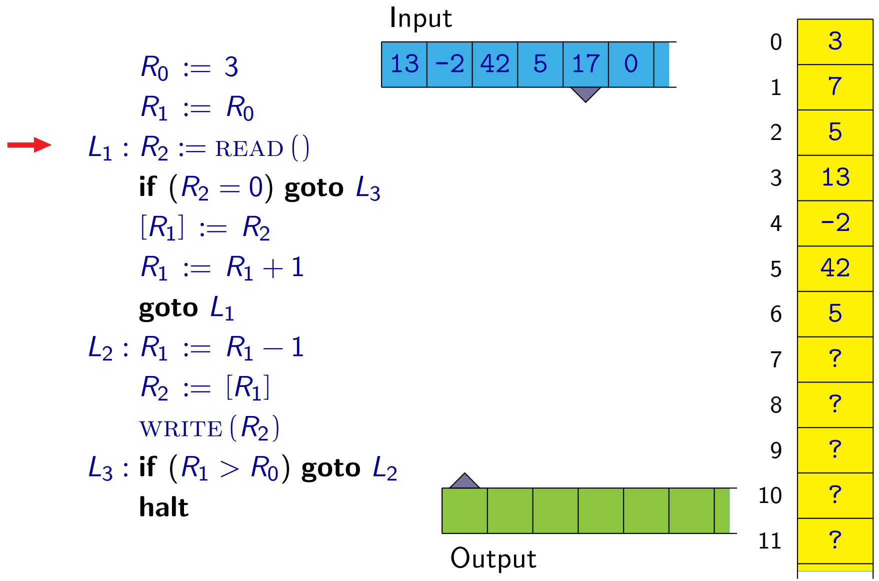
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ ()
          if $(R_2 = 0)$ goto $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
→         goto $L_1$
    $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE $(R_2)$
    $L_3$ : if $(R_1 > R_0)$ goto $L_2$
          halt
```



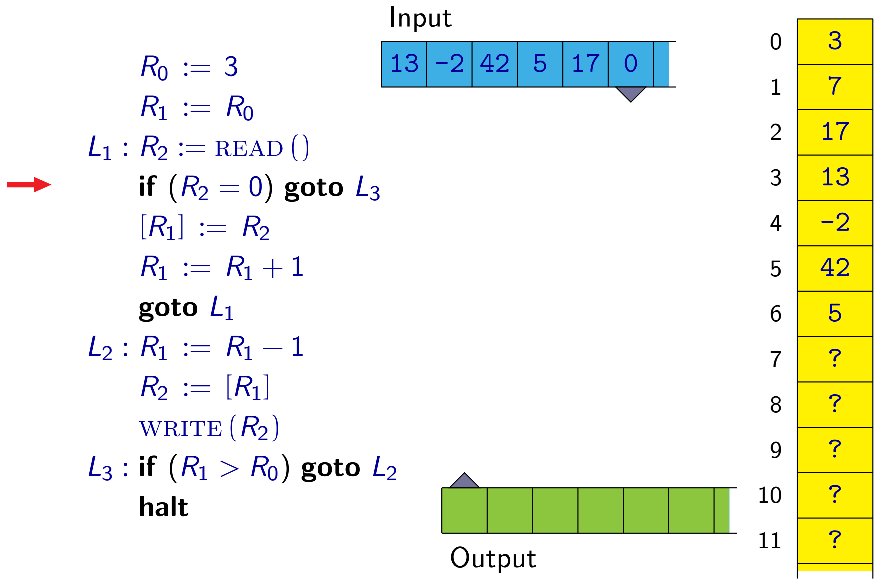
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



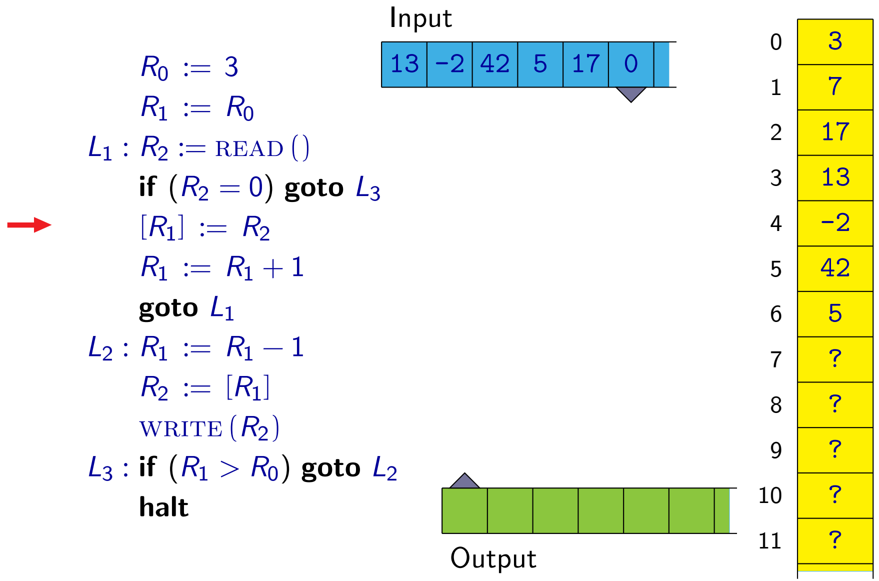
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      if ($R_2 = 0$) goto $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
      halt
```



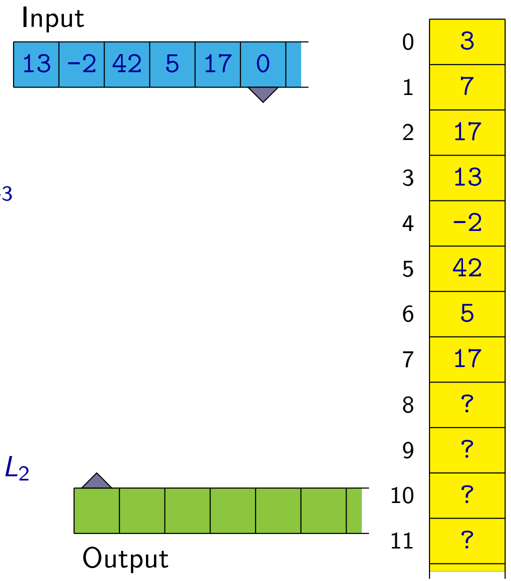
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
$L_1$ :   $R_2 :=$ READ ( )
          **if** $(R_2 = 0)$ **goto** $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          **goto** $L_1$
$L_2$ :   $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE $(R_2)$
$L_3$ :   **if** $(R_1 > R_0)$ **goto** $L_2$
          **halt**
```



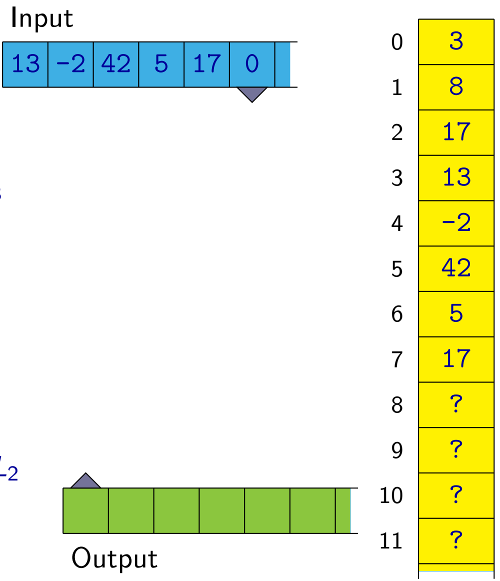
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
→       $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



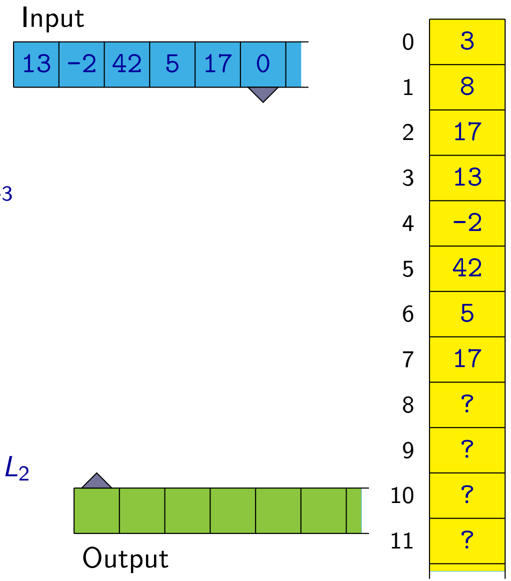
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
→       goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



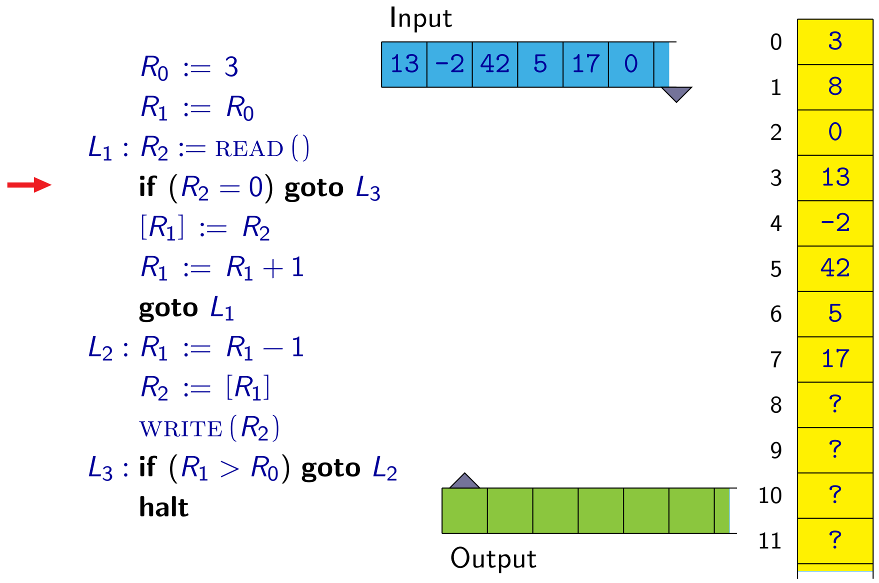
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
→  $L_1$: $R_2 :=$ READ ()
         if ($R_2 = 0$) goto $L_3$
         $[R_1] := R_2$
         $R_1 := R_1 + 1$
         goto $L_1$
   $L_2$: $R_1 := R_1 - 1$
         $R_2 := [R_1]$
         WRITE ($R_2$)
   $L_3$: if ($R_1 > R_0$) goto $L_2$
         halt
```



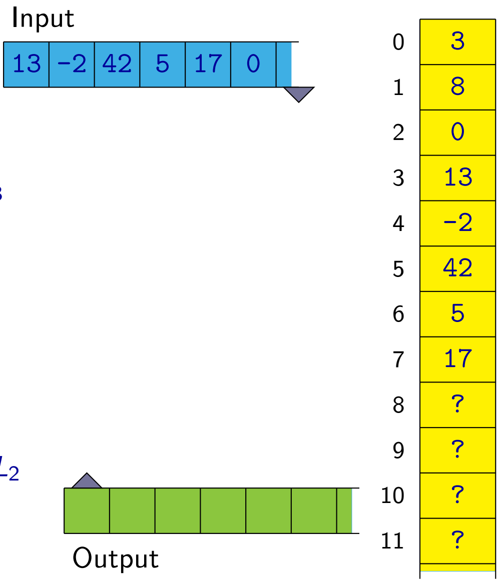
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



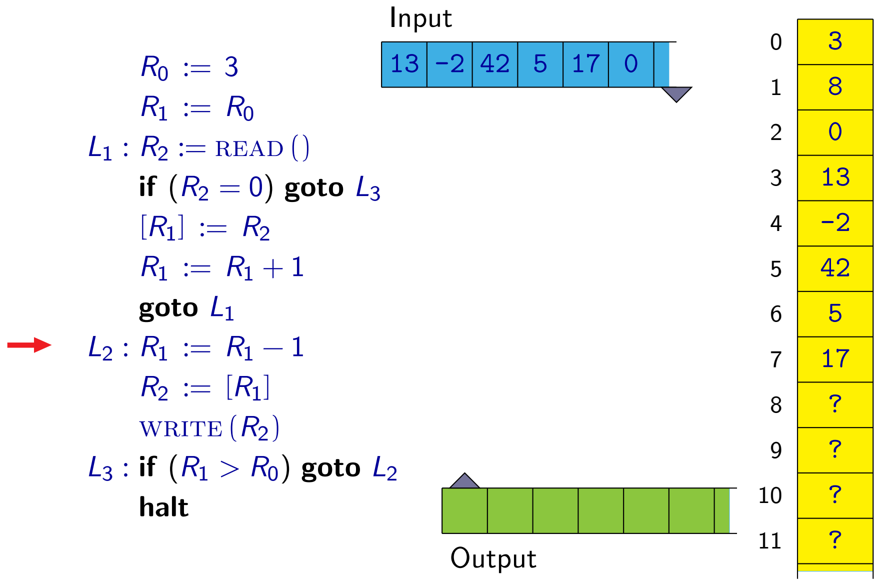
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
  $L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
→ $L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



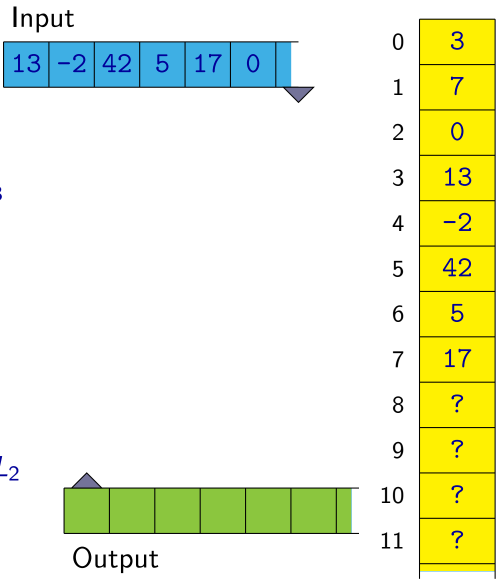
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
  $L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
  $L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



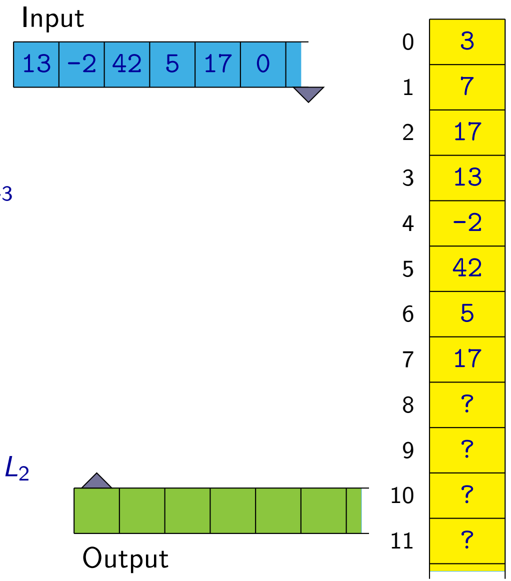
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      if $(R_2 = 0)$ goto $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
      halt
```



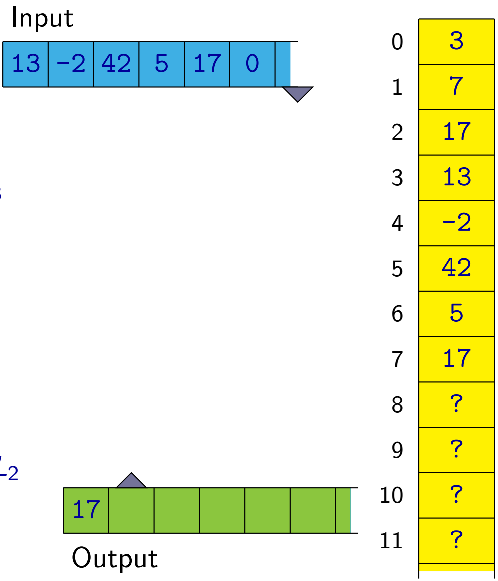
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
→       WRITE ($R_2$)
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



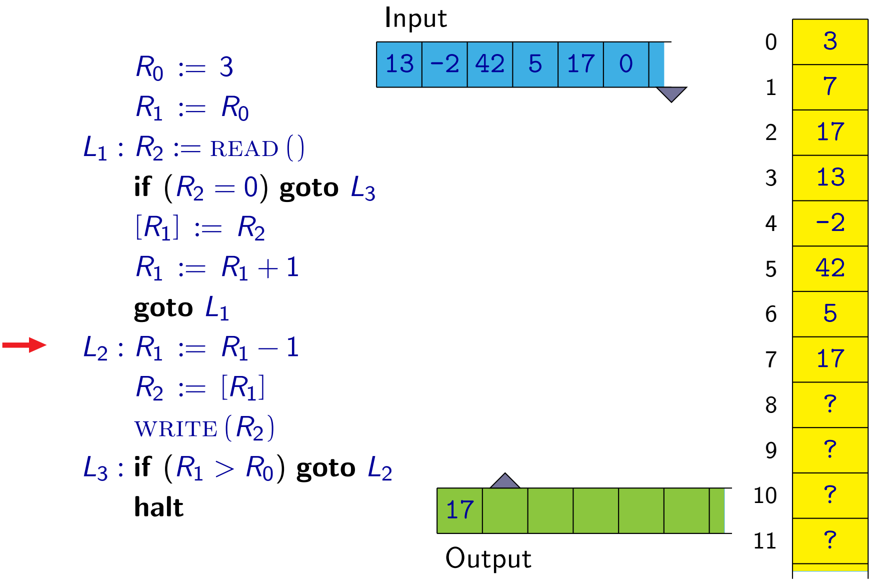
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ ()
          if $(R_2 = 0)$ goto $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          goto $L_1$
    $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE $(R_2)$
→   $L_3$ : if $(R_1 > R_0)$ goto $L_2$
          halt
```



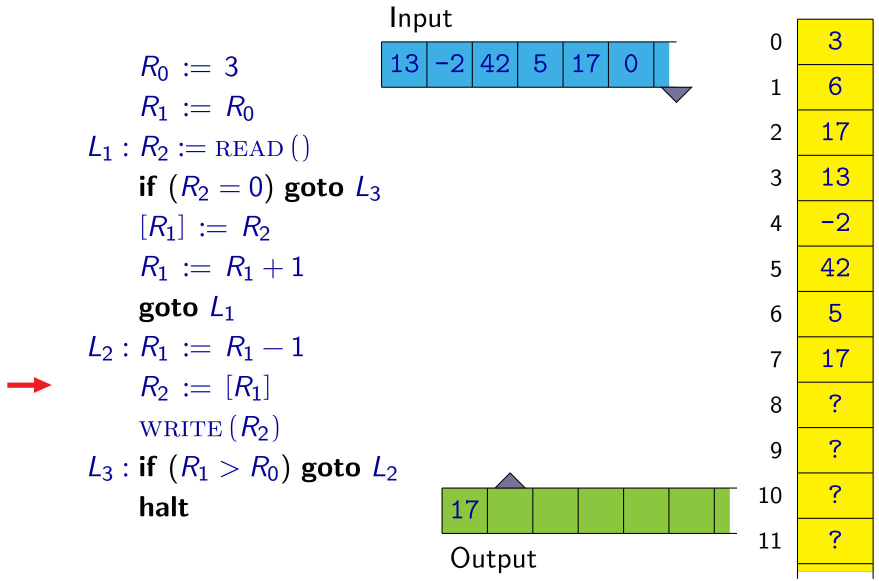
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ( )
       if ($R_2 = 0$) goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE ($R_2$)
$L_3$: if ($R_1 > R_0$) goto $L_2$
       halt
```



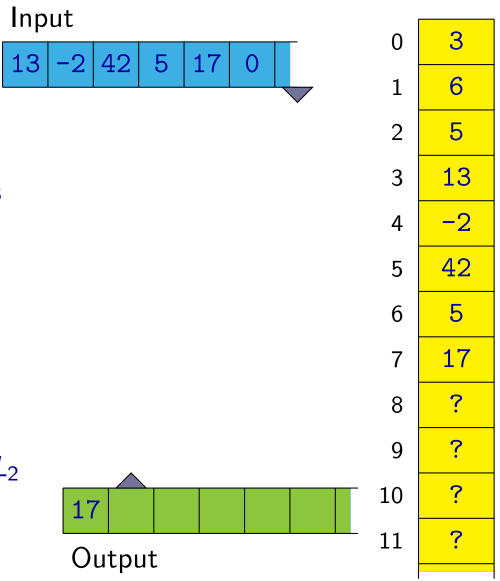
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



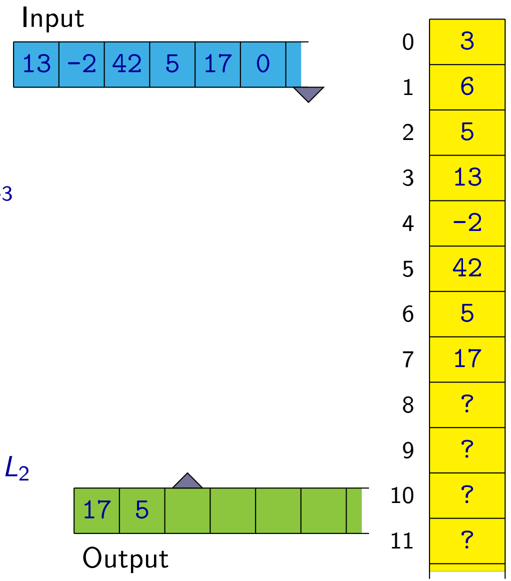
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ( )
      if ($R_2 = 0$) goto $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
→     WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
      halt
```



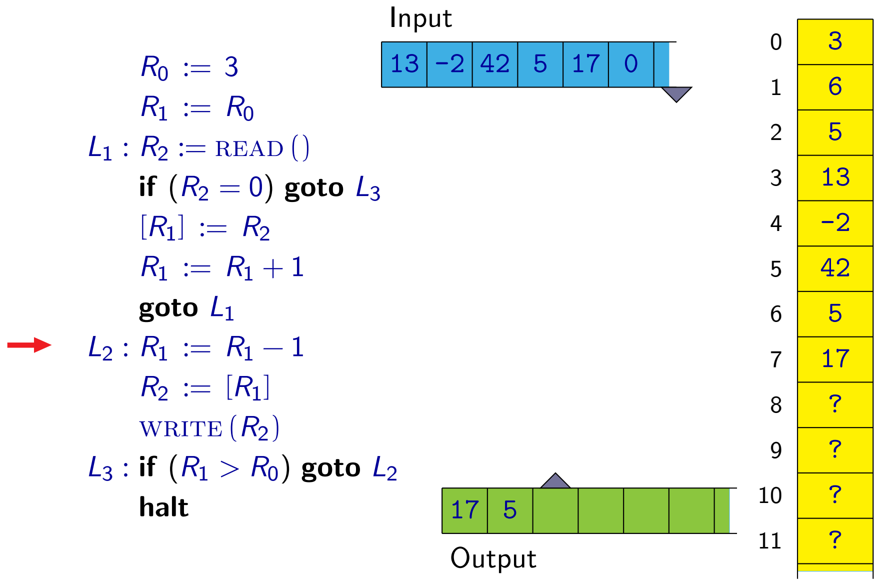
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ ()
          if ($R_2 = 0$) goto $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          goto $L_1$
    $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE ($R_2$)
→   $L_3$ : if ($R_1 > R_0$) goto $L_2$
          halt
```



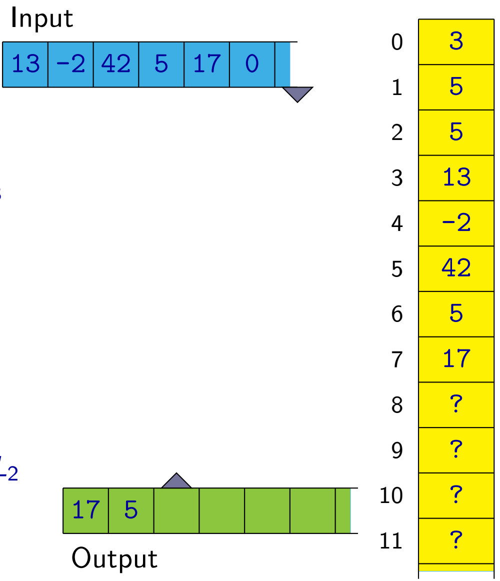
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



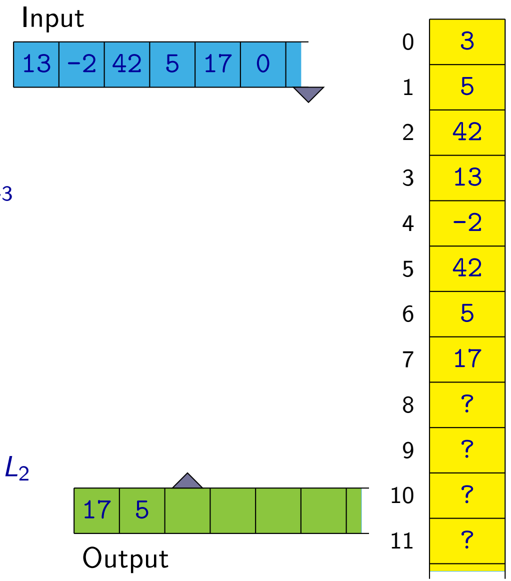
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



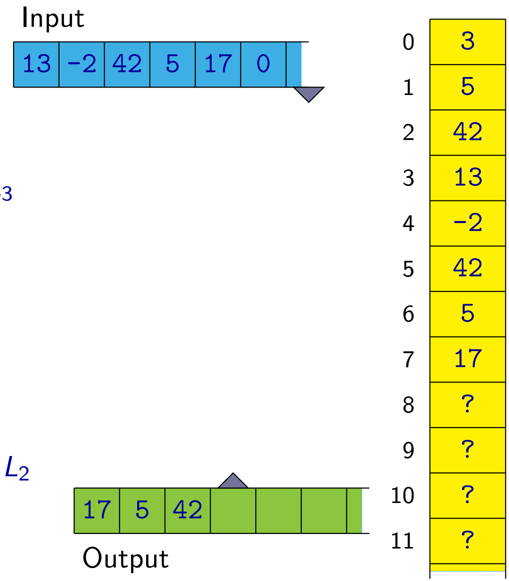
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ ()
          **if** ($R_2 = 0$) **goto** $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          **goto** $L_1$
    $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
→         WRITE ($R_2$)
    $L_3$ : **if** ($R_1 > R_0$) **goto** $L_2$
          **halt**
```



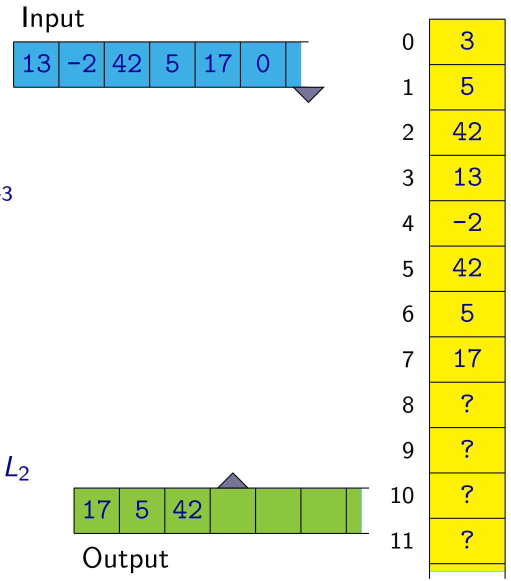
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ ()
          if $(R_2 = 0)$ goto $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          goto $L_1$
    $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE $(R_2)$
→   $L_3$ : if $(R_1 > R_0)$ goto $L_2$
          halt
```



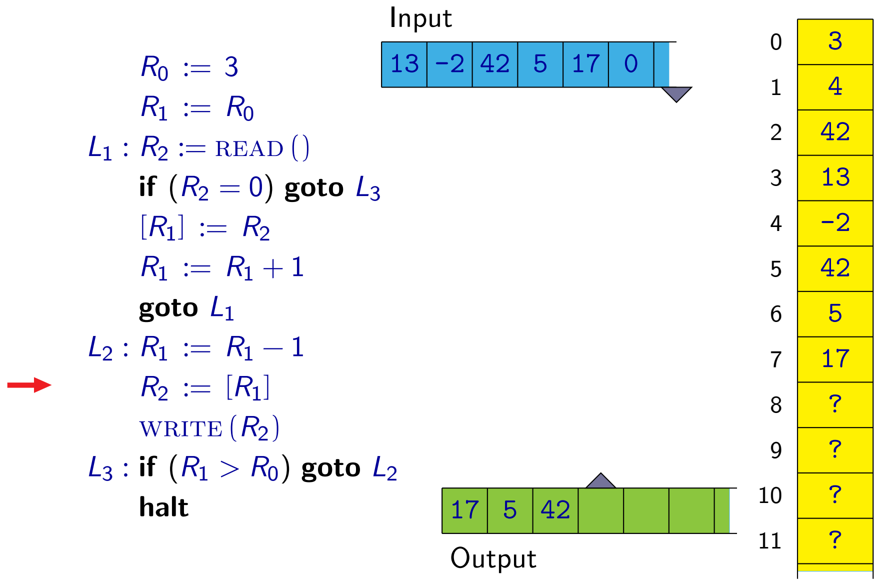
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       if $(R_2 = 0)$ goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE $(R_2)$
$L_3$: if $(R_1 > R_0)$ goto $L_2$
       halt
```



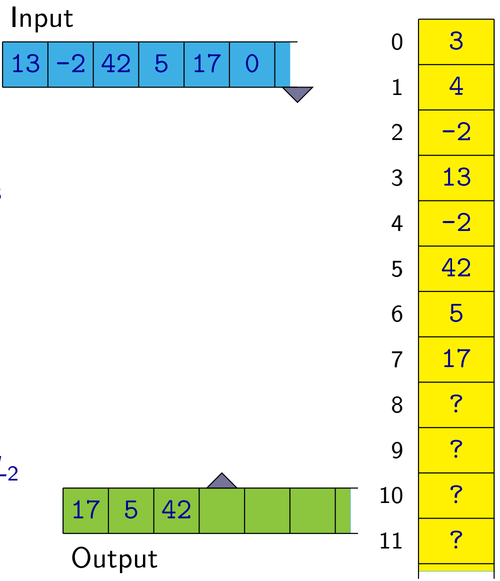
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ( )
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



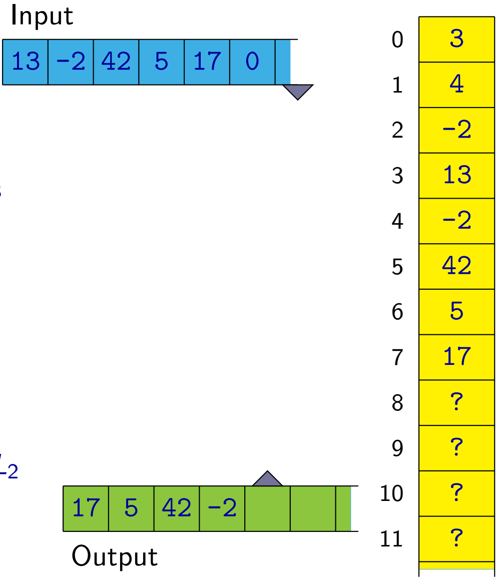
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 := \text{READ}()$
       if $(R_2 = 0)$ goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
→      $\text{WRITE}(R_2)$
$L_3$: if $(R_1 > R_0)$ goto $L_2$
       halt
```



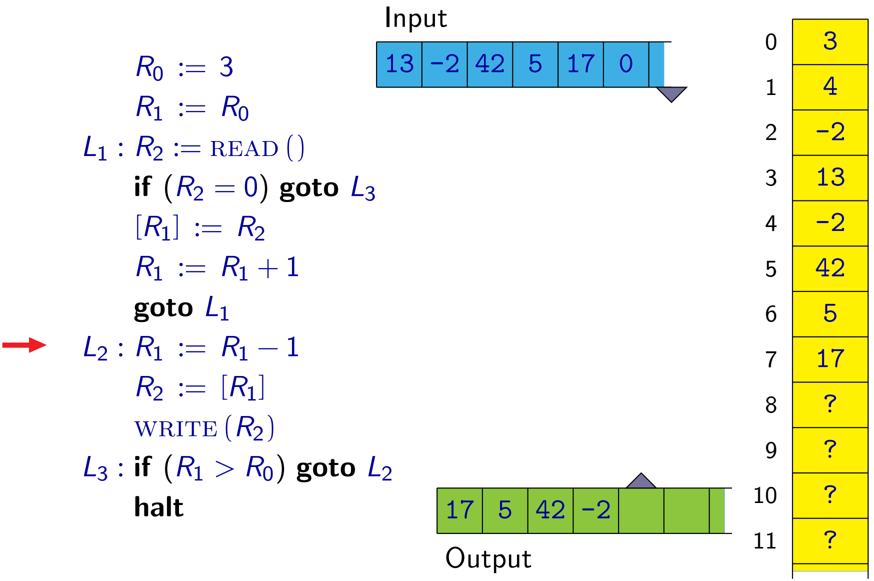
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
  $L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
→ $L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



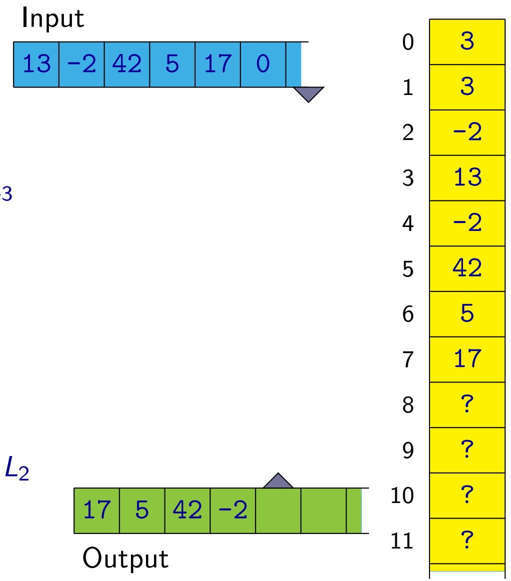
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       if ($R_2 = 0$) goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE ($R_2$)
$L_3$: if ($R_1 > R_0$) goto $L_2$
       halt
```



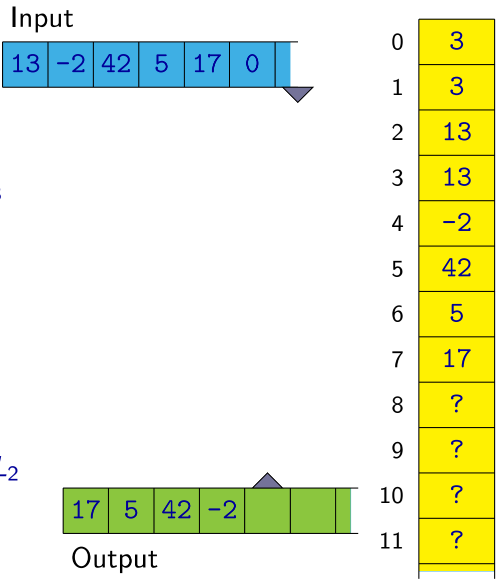
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ ()
         if ($R_2 = 0$) goto $L_3$
         $[R_1] := R_2$
         $R_1 := R_1 + 1$
         goto $L_1$
    $L_2$ : $R_1 := R_1 - 1$
→        $R_2 := [R_1]$
         WRITE ($R_2$)
    $L_3$ : if ($R_1 > R_0$) goto $L_2$
         halt
```



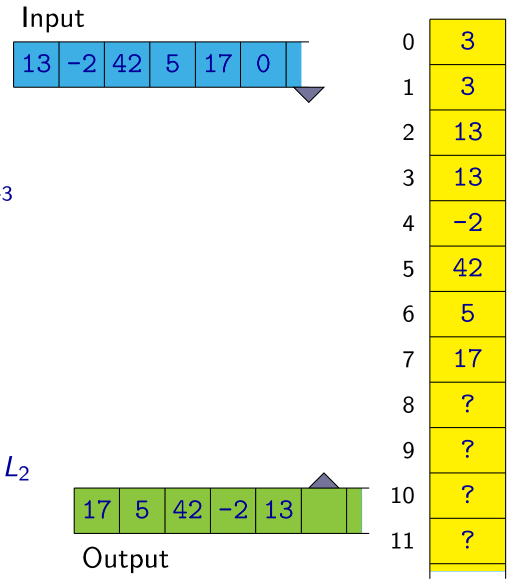
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
→       WRITE $(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



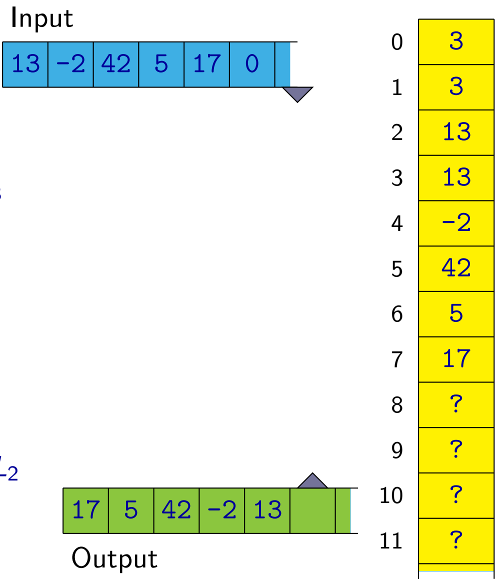
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 := \text{READ}()$
          **if** $(R_2 = 0)$ **goto** $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          **goto** $L_1$
    $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          $\text{WRITE}(R_2)$
→   $L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
          **halt**
```



Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
→       **halt**
```



Random Access Machine

Main differences with respect to real computers:

- The size of memory is not limited (an address can be an arbitrary natural number).
- The size of a content of individual memory cells is not limited (a cell can contain an arbitrary integer).
- It reads data sequantially from an input that consists of a sequence of integers. The input is read-only.
- It writes data sequantially on the output that consists of a sequence of integers. The output is write-only.

Random Access Machine

- Operations like an access to a memory cell with an address less than zero or division by zero result in an error — the computation is stuck.
- For an initial content of memory there are basically two possibilities how to define it:
  - All cells are initialized with value 0.
  - Reading a cell, to which nothing has been written, results in an error.

    Cells at the beginning contain a special value (denoted here by symbol '$?$') that represents that the given cell has not been initialized yet.
- We could consider also variants of RAMs where memory cells (and cells of input and output) do not contain integers (i.e., the elements of set $\mathbb{Z}$) but they can contain only natural numbers (i.e., elements of set $\mathbb{N}$).

  For example, operation of subtraction ($R_i := R_j - R_k$) then behaves in such a way that whenever the result should be a negative number, then value 0 is assigned as the result.

Random Access Machine

- Different variants of RAMs can differ in what particular operations can be used in arithmetic instructions.

  For example:

  - a support of bitwise operations (and, or, not, xor, . . .), bit shifts, . . .
  - a variant of RAM that does not have operations for multiplication and division

- We could also consider a variant of RAM where instead of instructions of the form

  $$\textbf{if } (R_i \ \mathit{rel}\ R_j) \textbf{ goto } \ell \qquad \text{nebo} \qquad \textbf{if } (R_i \ \mathit{rel}\ c) \textbf{ goto } \ell$$

  all conditional jumps are of the form

  $$\textbf{if } (R_i \ \mathit{rel}\ 0) \textbf{ goto } \ell$$

  Instead of all relations $\{=, \neq, \leq, \geq, <, >\}$, only a subset of them can be supported, e.g., $\{=, >\}$.

Random Access Machine

- In some variants of RAM, the input and output are not in a form of sequence of numbers.

  Instead, such machine could work with input and output tapes containg sequences of symbols from some alphabet, e.g., $\{0, 1\}$.

  This machine then could have for example some instructions that allow the branch the computation according to a symbol read from the input.

  However, the internal memory even in this variant works with numbers.

- When a machine should produce an answer of the form Yes/No (i.e., to accept or reject the given input), it does not need to have an output tape.

  Instruction **halt** is then replaced with instructions **accept** and **reject**.

Random Access Machine

- In the standard definition of RAM, jump instructions jumping to an adress stored in some memory cell are usualy not considered:

  $$\textbf{goto } R_i$$

  RAM could be extended with these instructions.

- For RAMs, a code of a program is usually stored in a separate read-only memory, not in a working memory.

  So the code can not be modified during a computation.

Random Access Machine

- A type of a machine, similar to RAM, but where its program is stored in its working memory (instructions are encoded by numbers) and so it can be modified during a computations, is called **RASP** (**random-access stored program**).

  RASP can simulate behaviour of self-modifying programs.

Turing Machine Simulating RAM

It is not difficult to come with a general way how a computation of an arbitrary Turing machine can be simulated by RAM.

To simulate behaviour of an arbitrary RAM by a Turing machine is more complicated.

In the description of how a Turing machine can simulate a RAM, it is simpler to proceed by smaller steps:

- We will show how to simulate a varint of RAM described before by a variant of RAM with somewhat simpler instructions.
- We will show how to simulate the behaviour of this simpler variant of RAM by a multitape Turing machine.
- We have already seen before how a multitape Turing machine can be simulated by one-tape Turing machine.

A simpler variant of RAM

This simpler variant of RAM has, in addition to its working memory, also three **registers**:

- **register A** — almost all instructions work with this register, results of all operations are stored into this register

  Remark: This kind of register is often called an **accumulator**.

- **register B** — this register is used to store the second operand of arithmetic instructions (the first operand is always in the accumulator)

- **register C** — this register is used to store an address of a memory cell, to which a value is written by a store operation

A simpler variant of RAM

Overview of instructions:

$A := c$	– assinment of a constant
$B := A$	– assinment to register B
$C := A$	– assinment to register C
$A := [A]$	– load (reading from memory)
$[C] := A$	– store (writing to memory)
$A := A\ op\ B$	– arithmetic instructions, $op \in \{+, -, *, /\}$
if $(A\ rel\ 0)$ **goto** ℓ	– conditional jump, $rel \in \{=, \neq, \leq, \geq, <, >\}$
goto ℓ	– unconditional jump
$A := \textsc{read}()$	– reading from input
$\textsc{write}(A)$	– writing to output
halt	– program termination

A simpler variant of RAM

For example, instruction

$$R_7 := R_3 + R_6$$

can be replaced with a sequence of instructions:

$$\begin{aligned}
A &:= 7\\
C &:= A\\
A &:= 6\\
A &:= [A]\\
B &:= A\\
A &:= 3\\
A &:= [A]\\
A &:= A + B\\
[C] &:= A
\end{aligned}$$

A simpler variant of RAM

For example, instruction

$$[R_{15}] := R_9$$

can be replaced with a sequence of instructions:

$$\begin{aligned}
A &:= 15\\
A &:= [A]\\
C &:= A\\
A &:= 9\\
A &:= [A]\\
[C] &:= A
\end{aligned}$$

A simpler variant of RAM

For example, instruction

$$\textbf{if } (R_4 \geq R_{11}) \textbf{ goto } \ell$$

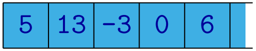
can be replaced with a sequence of instructions:

```
$A := 11$
$A := [A]$
$B := A$
$A := 4$
$A := [A]$
$A := A - B$
if $(A \geq 0)$ goto $\ell$
```

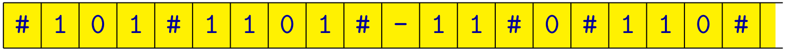
Turing Machine Simulating RAM

A Turing machine works with words over some alphabet, while a RAM works with numbers. But numbers can be written as sequences of symbols and conversely symbols of an alphabet can be written as numbers.

For example the following input of a RAM



can be represented for a Turing machine as



Turing Machine Simulating RAM

A Turing machine simulating a computation of a RAM has several tapes:

- A tape containing a content of the working memory of the RAM.
- Three tapes containing values of registers A, B, and C.

  (Values of registers A, B, and C will be written on these tapes in binary and delimited from the left and from the right by symbols #.)
- A tape representing the input tape of the RAM.
- A tape representing the output tape of the RAM.
- One auxiliary tape used for an implementation of the simulation of some instructions.

Turing Machine Simulating RAM

The Turing machine stores the information about the instruction of the RAM that is currently executed in its control unit.

Execution of most of instructions is not difficult:

- $A := c$

  it writes bits of the constant c to the tape of register A

- $B := A$ or $C := A$

  it will copy a content of the tape of register A to the tape of register B or C

- **goto** ℓ

  just changes the state of the control unit of the Turing machine

- **if** $(A\ rel\ 0)$ **goto** ℓ, kde $rel \in \{=, \neq, \leq, \geq, <, >\}$

  the content of the working register is tested and the state of the control unit is changed accordingly

Turing Machine Simulating RAM

- $A := \textsc{read}()$

  copy the value (marked at the ends by symbols "#") from the input tape to the tape of register A

- $\textsc{write}(A)$

  copy the value of register A to the output tape.

- **halt**

  the computation halts

Turing Machine Simulating RAM

Also arithmetic instructions are rather easy to implement, although the a little bit more complicated than the previous instructions:

- $A := A \ op \ B$, where $op \in \{+, -, *, /\}$

  The Turing machine performs the given operation (such as addition or subtraction) bit by bit, the result is stored to register A.

Remark: Multiplication and division can be done as a sequence of additions and bit shifts.

In the implementation of addition and division, it may be necessary to use an auxiliary tape to store intermediate results.

Turing Machine Simulating RAM

Probably the most complex is the implementation of the RAM memory.

One possibility is to store only values of those cells that were actually used so far in the computation of the RAM (we know that all other cells contain value 0).

Example: The RAM worked so far only with cells 2, 3 and 6:

- Cell 2 contains value 11.
- Cell 3 contains value -1.
- Cell 6 contains value 2.

The content of the tape of the Turing machine representing the content of the memory of the RAM will be as follows:

$	#	1	0	:	1	0	1	1	#	1	1	:	-	1	#	1	1	0	:	1	0	#	$	

Turing Machine Simulating RAM

Load instruction, i.e., $A := [A]$:

- The Turing machine will search the given address, stored in register A, on the tape containg the content of the memory of the RAM. (If it does not find it, it will appened it at the end with value 0.)
- The given value in the cell is copied to the tape of register A.

Turing Machine Simulating RAM

Store instruction, i.e., $[C] := A$:

- Similarly as before, the Turing machine will find the position of the tape representing a content of the memory, where the value in the given address, stored in register C, occurs.
- The rest of the memory tape is copied to an auxiliary tape.
- The content of the tape of register A is copied to the corresponding place.
- The rest of the tape, copied on the auxiliary tape, is copied back to the memory tape (after the newly written value).

Algorithms

Pseudocode

Usually, we will not represent algorithms as programs for a RAM but rather as programs in some high-level programming language.

We will not use any particular programming language.

Rather, we will write programs in a form of **pseudocode** whose syntax could be adjusted in arbitrary ways according to our needs (e.g., we will use things like arbitrary mathematical notation, descriptions in a natural language, and so on, freely).

Example:

Algorithm: 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

Remark:

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)

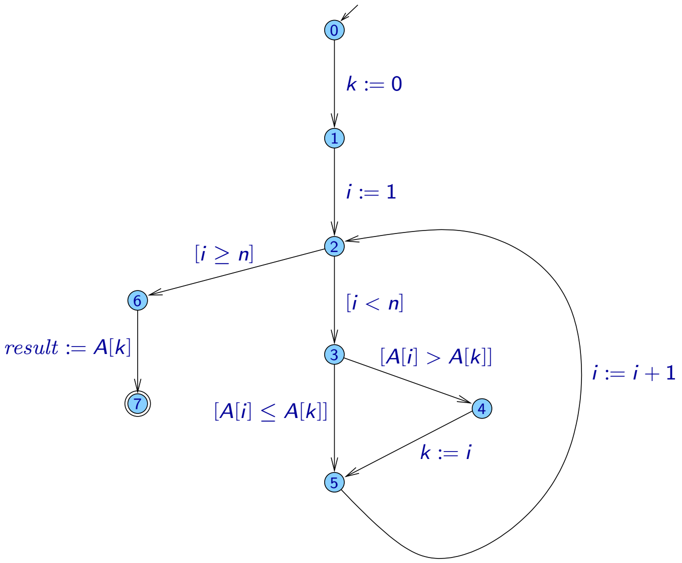
So in a pseudocode, input data will be often given as arguments of a function and an output will be represented as a return value of this function.

Control Flow

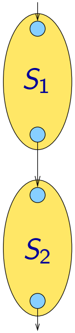
Instructions can be roughly devided into two groups:

- 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, ...)
  - ...

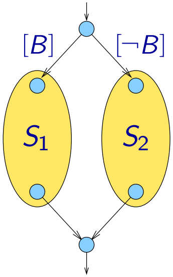
Control Flow Graph



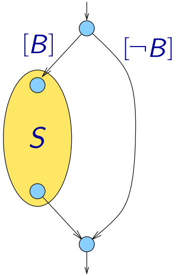
Some Basic Constructions of Structured Programming



$S_1; S_2$

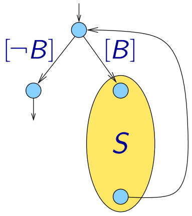


if B **then** S_1 **else** S_2

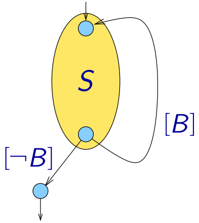


if B **then** S

Some Basic Constructions of Structured Programming

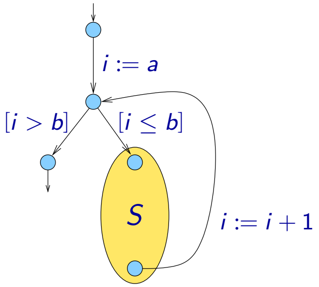


while B **do** S



do S **while** B

Some Basic Constructions of Structured Programming



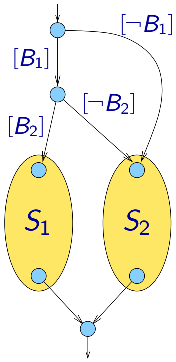
```
$i := a$
while $i \leq b$ do
    $S$
    $i := i + 1$
```

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

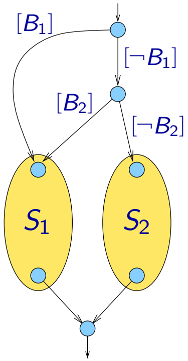
Some Basic Constructions of Structured Programming

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 $A[i] \leq A[k]$ 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

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

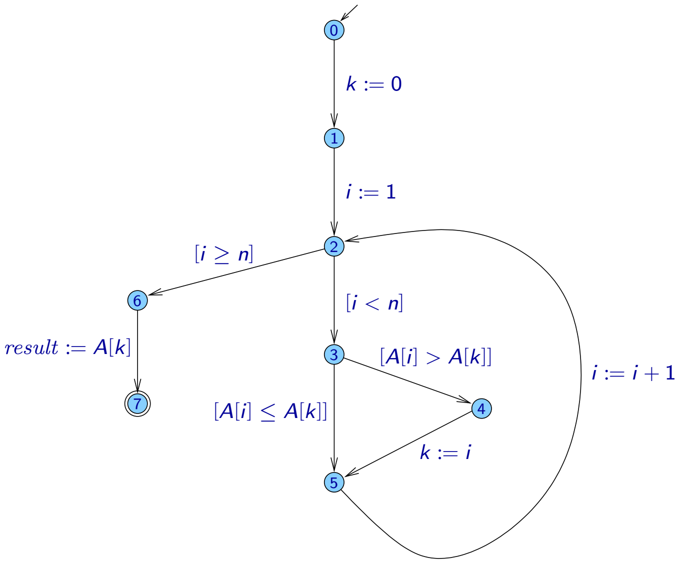
An example of a configuration:

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

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

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

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

Computation of an Algorithm

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

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

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

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 $n = 5$.

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

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 $\textsc{Find-Max}$ processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

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

Computation of an Algorithm

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

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

Computation of an Algorithm

Example: A computation, where algorithm $\textsc{Find-Max}$ processes an input where $A = [3,8,1,3,6]$ and $n = 5$.

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

Computation of an Algorithm

Example: A computation, where algorithm $\textsc{Find-Max}$ 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) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{14}: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{15}: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ 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) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{14}: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{15}: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{16}: & (6,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ \mathit{result}\colon ?\rangle)
\end{array}
$$

Computation of an Algorithm

Example: A computation, where algorithm $\textsc{Find-Max}$ 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) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{14}: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{15}: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{16}: & (6,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{17}: & (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**:

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

  where α_t is either a final configuration or a configuration where an error occurred and it is not possible to continue in the computation

- **Infinite**:

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

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 $I_0, I_1, I_2, \ldots$

Church-Turing Thesis

It should be clear from the previous discussion that:

- A program written in an arbitrary programming language could be translated to a program for a RAM.
- Behaviour of a RAM could be simulated by a Turing machine.

So the behaviour of a program written in an arbitrary programming language could be simulated by a Turing machine.

Church-Turing thesis

Every algorithm can be implemented as a Turing machine.

It is not a theorem that can be proved in a mathematical sense – it is not formally defined what an algorithm is.

The thesis was formulated in 1930s independently by Alan Turing and Alonzo Church.

Church-Turing Thesis

Examples of mathematical formalisms modelling the notion of an algorithm:

- Turing machines
- Random Access Machines
- Lambda calculus
- Recursive functions
- . . .

We can also mention:

- An arbitrary (general purpose) programming language (for example C, Java, Python, Lisp, Haskell, Prolog, etc.).

All these models are equivalent with respect to algorithms that can be implemented by them.

Proving Correctness of Algorithms

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

Algorithm is a correct solution of a given problem if it halts for all inputs and for all inputs it produces a correct output.

Example:

Problem: The problem of sorting
Algorithm: Quicksort

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.,

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

The output for this instance is value 11.

Correctness of Algorithms

Algorithm: 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 $A[k]$
```

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

Typically, the set of possible instances of a given problem is infinite (or at least very big), so it is not possible to test the behaviour of the algorithm for all instances.

As a justification and a verificatoin of the fact that an algorithm is a correct solution of a given problem, we need to have a **proof** that takes into account all possible computations on all possible inputs.

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.

Invariants

Consider an arbitrary system consisting of:

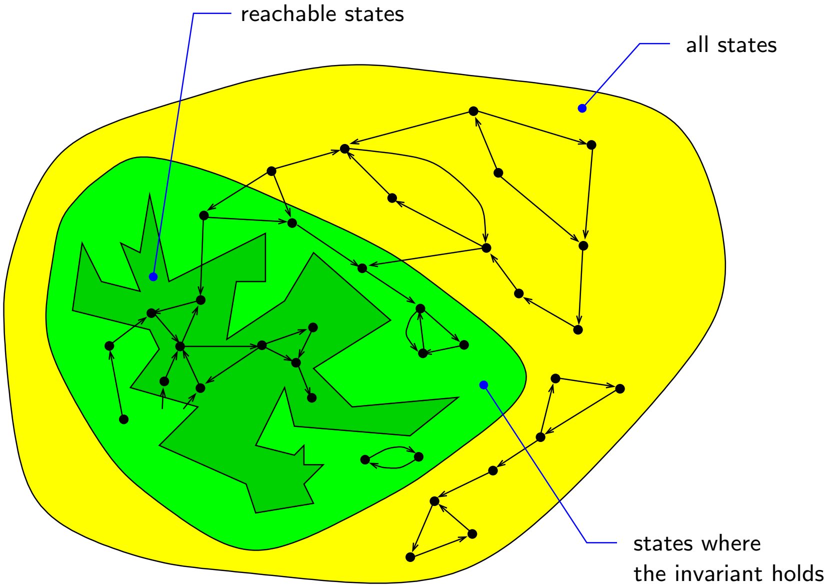
- a set of states (or configurations) — it can be infinite
- transitions between these states
- some states are specified as initial

A state is **reachable** if it is possible to reach it from some initial state using a sequence of transitions.

An **invariant** is a condition determining a subset of states such that all reachable states satisfy these condition:

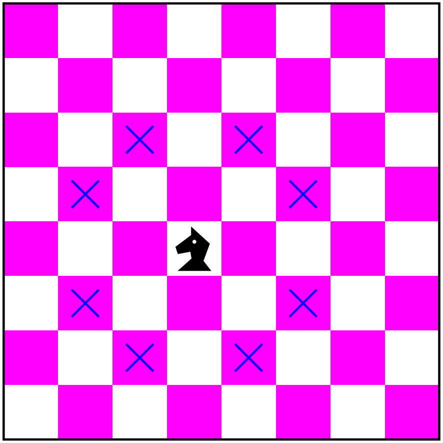
- it is satisfied in all initial states
- if it is satisfied in a state and there is a transition from this state, by which the system goes to another state in one step, then this condition will be satisfied also in this other state

Invariants


reachable states
all states
states where
the invariant holds

Invariants

Example: We will move with a knight (a chess piece) on a chessboard and at the same time we will count the number of the moves performed; the knight starts on some white square in the leftmost column:



Invariants

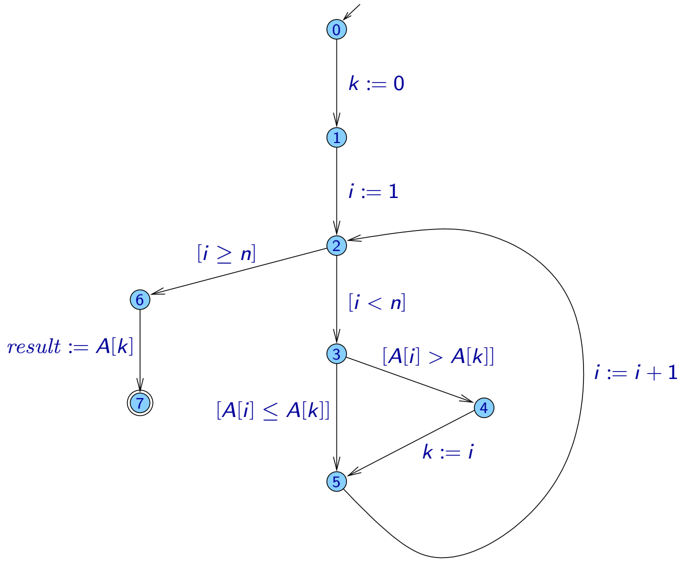
- **States** — pairs consisting of a current position of the knight on the chessboard and a value of the counter giving the number of moves performed so far
- **Transitions** — making one move with the knight (according to the rules of chess) and incrementing the counter by one
- **Initial states** — the knight is on a white square in the leftmost column and the value of the counter is 0

For example, the following invarint holds:

- if the value of the counter is even, the knight is on a white square
- if the value of the counter is odd, the knight is on a black square

Invariants

Example: Algorithm Find-Max represented as a control-flow graph



Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

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

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

$$
\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) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{14}: & (5,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 4,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle) \\
\alpha_{15}: & (2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 5,\ \ k\colon 1,\ \ \mathit{result}\colon ?\rangle)
\end{array}
$$

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

$$\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) \\
\alpha_{13}: & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{14}: & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{15}: & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ \mathit{result}\colon ?\rangle) \\
\alpha_{16}: & (6,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 5,\ k\colon 1,\ \mathit{result}\colon ?\rangle)
\end{array}$$

Invariants

A computation for input $A = [3, 8, 1, 3, 6]$ and $n = 5$ as a sequence of configurations:

$$
\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}
$$

Invariants

- **States** — configurations consisting of a state of the control unit and a content of the memory represented by values of all variables.
- **Transitions** — they are determined by instructions on the edges of the control-flow graph, they change both the control state and the content of the memory by assigning values to variables
- **Initial states** — all possible initial configurations for all possible input instances that are allowed according to a specification of the problem

Invariants will be propositions referring to configurations, i.e., they talk about states of the control unit and values of the variables

- If the control state is 2, then, in the given configuration, it holds that $1 \leq i \leq n$, $0 \leq k < i$, and $A[k]$ is the greatest of the elements $A[0], A[1], \ldots, A[i-1]$.

Invariants

For those systems, where configurations contain a control state, it can be convenient to state invariants in the form:

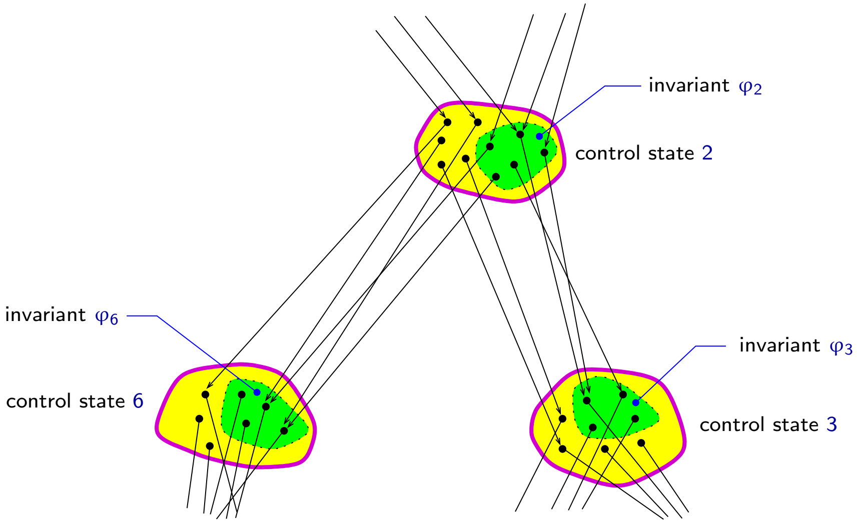
- if the control state is 0, then φ_0 holds
- if the control state is 1, then φ_1 holds

  $\vdots$
- if the control state is r, then φ_r holds

where the propositions $\varphi_0, \varphi_1, \ldots, \varphi_r$ refer only to the content of the memory, not to the control state.

Configurations can divided into (finitely many) groups according to the states of the control unit.

Invariants


invariant φ_2
control state 2
invariant φ_6
control state 6
invariant φ_3
control state 3

Invariants

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.

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

Established invariants can be useful for many different purposes:

- They can help in better understanding the behaviour of the algorithm.
- They can be used to verify that ceirtain types of errors do not occur — e.g., an out of bounds array access, division by zero, . . .

  We can verify that in those places in the code where such errors could potentially occur the invariants hold that ensure that the variables will always have values, for which the given error can not occur.

  Example: When element $A[i]$ will be accessed, it will always hold that $0 \leq i < n$, where n is the length of the array.
- An invariant that will hold in the final configurations will ensure that the output of the algorithm is correct with respect to the specification of the problem.
- In an analysis of the computational complexity, they could be useful in the examination how many times some instructions will be performed or how much memory is needed during the computation.

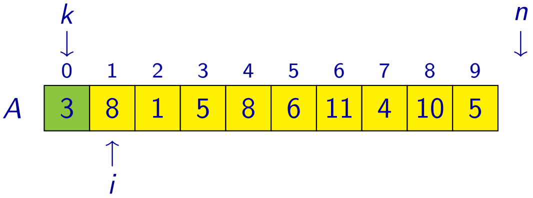
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



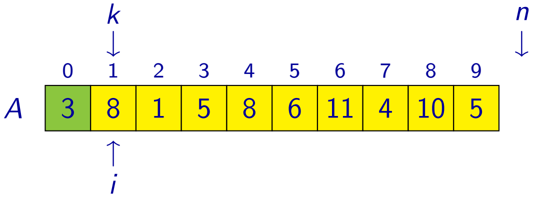
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



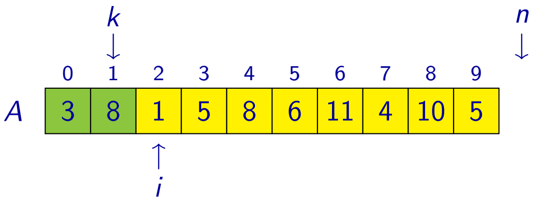
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm FIND-MAX for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



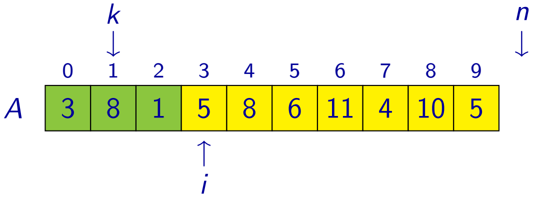
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



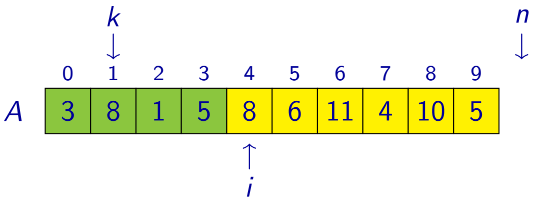
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm FIND-MAX for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



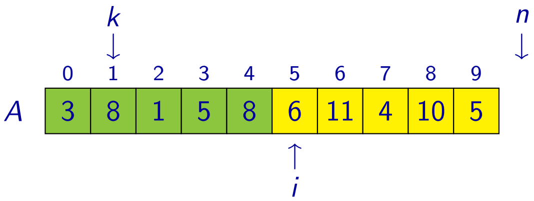
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm FIND-MAX for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



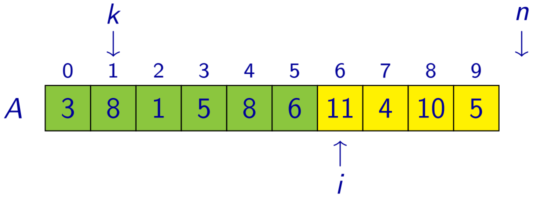
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm FIND-MAX for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



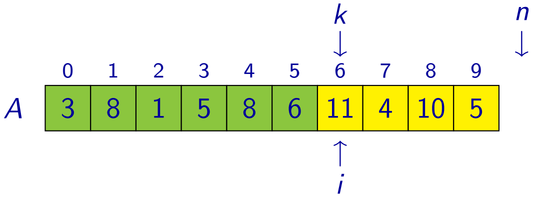
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm FIND-MAX for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



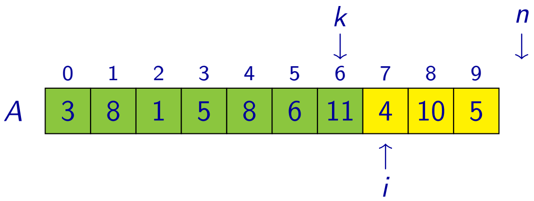
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



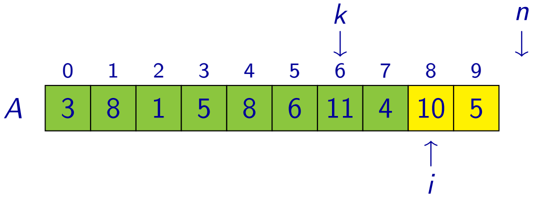
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



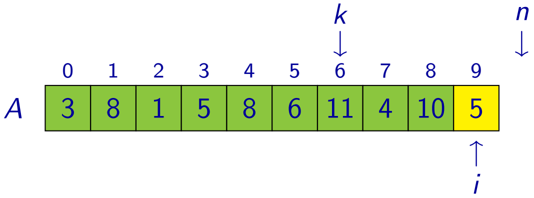
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



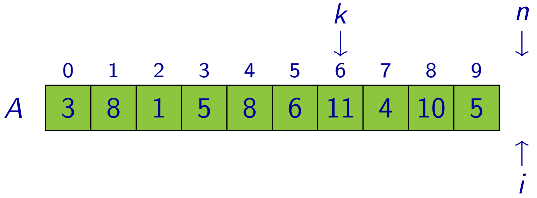
Invariants

Determining invariants is not a completely mechanical process. It requires a certain understanding of the behaviour of the algorithm.

Before formulating hypothesis what invariants hold in different control states, it can be useful to look at the behaviour of the algorithm on some particular concrete inputs.

Example: A computation of algorithm Find-Max for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



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

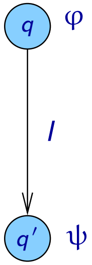
Checking that the given invariants really hold:

- We must check that the given invariants hold in the initial configurations — this is usually simple.
- 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 ψ

Invariants



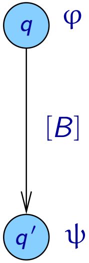
- for this edge we must check that for every configurations $\alpha = (q, mem)$ and $\alpha' = (q', mem')$ such that $\alpha \xrightarrow{I} \alpha'$, it holds that if
    - φ holds is configuration α,

  then
    - ψ holds in configuration α'

Invariants

Checking instructions, which are conditional tests:

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

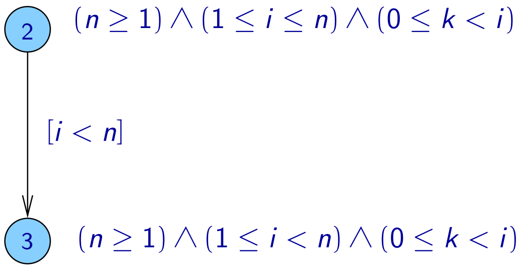

$$\begin{tikzcd} q \arrow[d, "{[B]}"] & \varphi \\ q' & \psi \end{tikzcd}$$

A content of memory is not modified, so it is sufficient to check that the following implication holds

$$(\varphi \land B) \Rightarrow \psi$$

Invariants

Example:



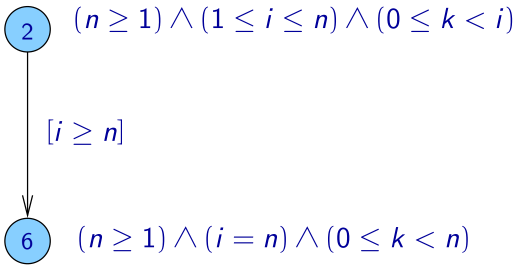
$$\begin{tikzcd}
\boxed{2} \quad (n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i) \arrow[d, "{[i < n]}"] \\
\boxed{3} \quad (n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)
\end{tikzcd}$$

It is sufficient to check that the followng implication holds:

- If $(n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i) \wedge (i < n)$,
  then $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$.

Invariants

Example:


2
$(n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i)$
$[i \geq n]$
6
$(n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$

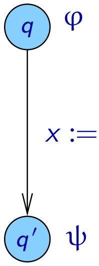
It is sufficient to check that the followng implication holds:

- If $(n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i) \wedge (i \geq n)$, then $(n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$.

Invariants

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

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



We must distinguish between the values of variable x before this assignment and after this assignment.

Invariants

We will need the following operation of **substitution** on formulas:

$$\varphi[E/x]$$

denotes a formula obtained from variable φ when we substitute an expressiion E for all free occurrences of variable x in formula φ.

Example: Let us say that φ is formula $(1 \le i) \land (i \le n)$.

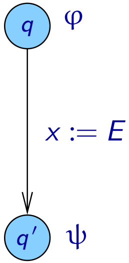
Notation $\varphi[i'/i]$ then denotes formula

$$(1 \le i') \land (i' \le n)$$

and notation $\varphi[(i+1)/i]$ denotes formula

$$(1 \le i+1) \land (i+1 \le n)$$

Invariants



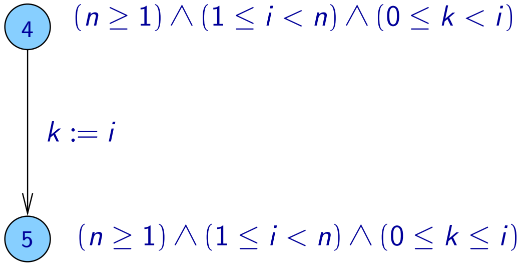
We will introduce a new variable x' representing the value of variable x after executing this assignment.

We need to check that the following implication holds:

$$(\varphi \land (x' = E)) \Rightarrow \psi[x'/x]$$

Invariants

Example:

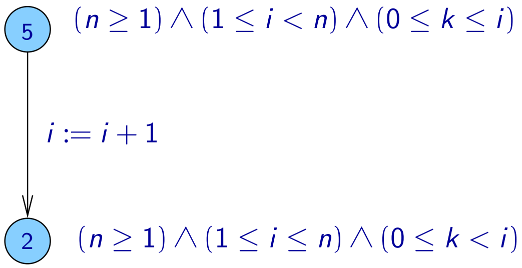

4
$(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
$k := i$
5
$(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k \leq i)$

It is sufficient to check that the following implication holds:

- If $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i) \wedge (k' = i)$, then $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k' \leq i)$.

Invariants

Example:


5
$(n \geq 1) \land (1 \leq i < n) \land (0 \leq k \leq i)$
$i := i + 1$
2
$(n \geq 1) \land (1 \leq i \leq n) \land (0 \leq k < i)$

It is sufficient to check that the following implication holds:

- If $(n \geq 1) \land (1 \leq i < n) \land (0 \leq k \leq i) \land (i' = i + 1)$, then $(n \geq 1) \land (1 \leq i' \leq n) \land (0 \leq k < i')$.

Invariants

Finishing the checking that the algorithm FIND-MAX 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

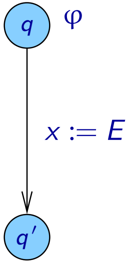
Example: In algorithm FIND-MAX, state 2 is such “important” state.

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 $A[j] \leq A[k]$.

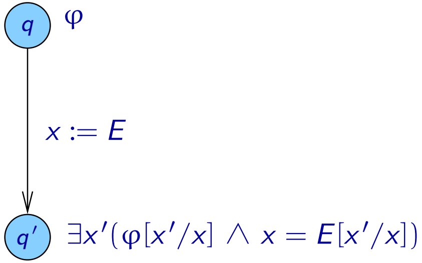
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



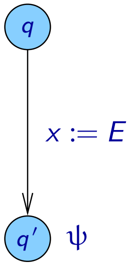
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



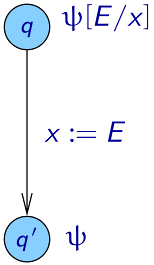
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



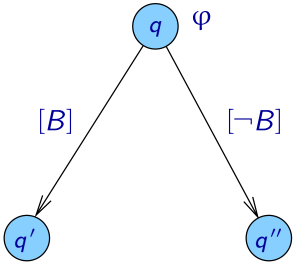
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



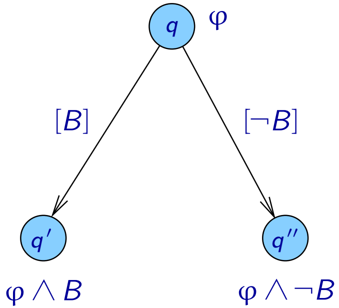
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



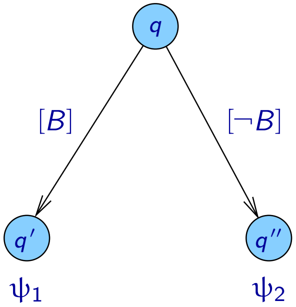
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



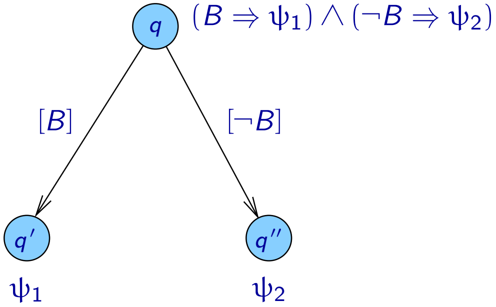
Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:


$$\begin{tikzcd} & q \arrow[dl, "{[B]}"'] \arrow[dr, "{[\neg B]}"] & \\ q' & & q'' \end{tikzcd}$$

Invariants

Examples that show how invariants for some other states could be determined, if we already have determinants for some states:



Invariants

Example:

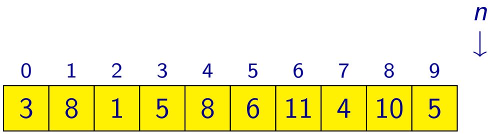
Algorithm: Insertion sort

```
INSERTION-SORT $(A, n)$:
    for $j := 1$ to $n - 1$ do
        $x := A[j]$
        $i := j - 1$
        while $i \geq 0$ and $A[i] > x$ do
            $A[i + 1] := A[i]$
            $i := i - 1$
        $A[i + 1] := x$
```

Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

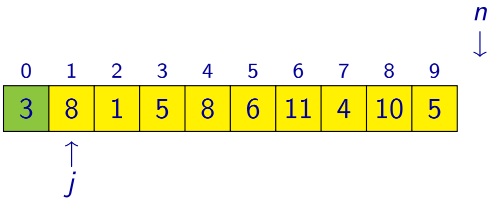


$x = ?$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

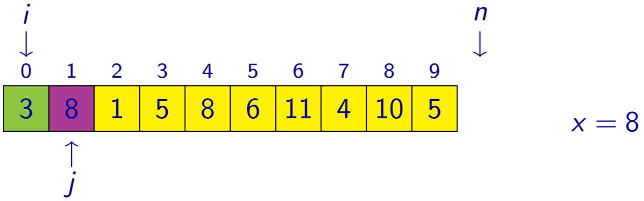


$x = ?$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

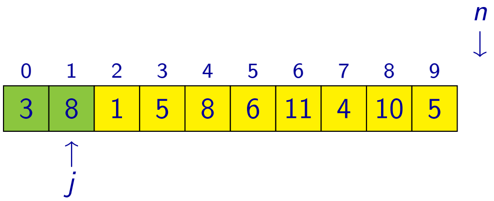
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$

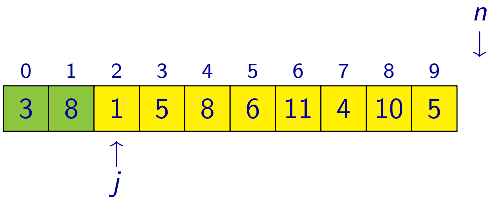


$x = 8$

Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

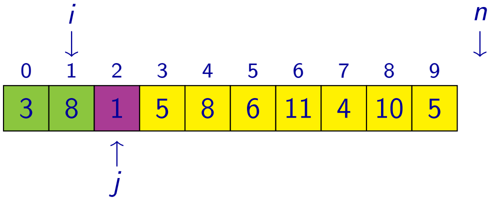


$x = 8$

Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

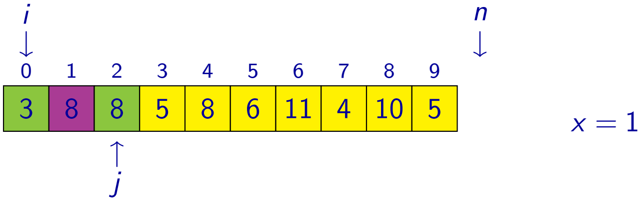


$x = 1$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

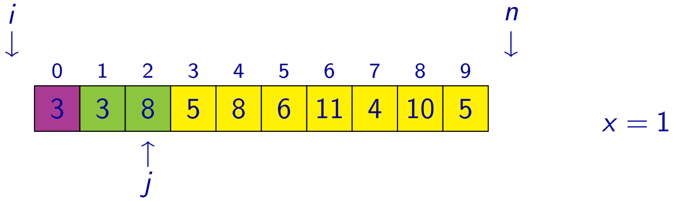
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

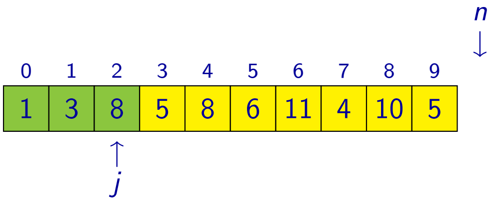
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5],\ n = 10.$$

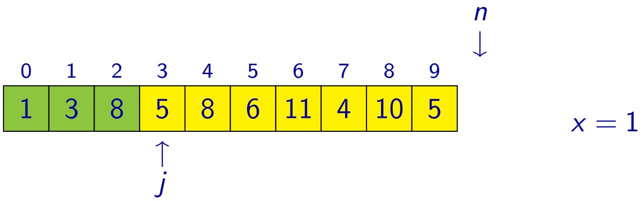


$x = 1$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

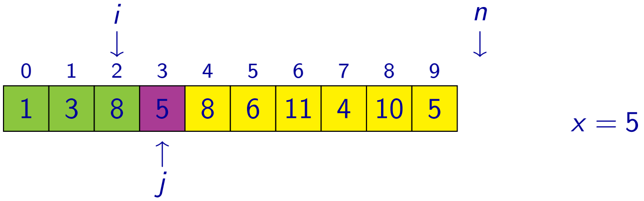
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

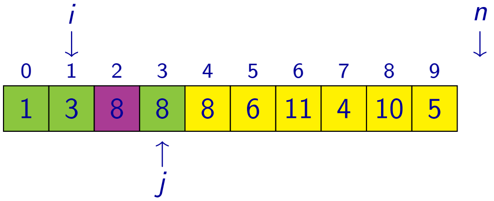
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$

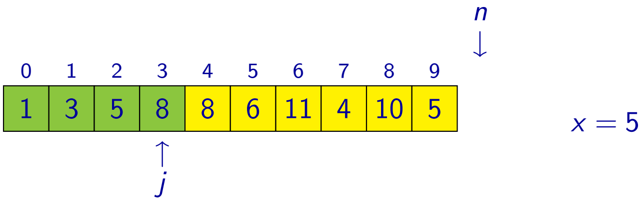


$x = 5$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

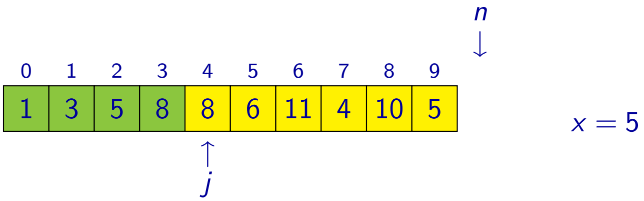
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

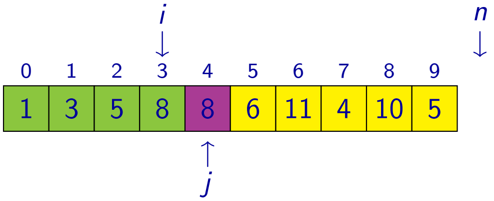
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

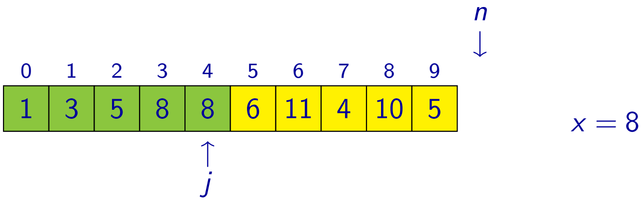


$x = 8$

Invariants

Example: A computation of algorithm Insertion-Sort for input

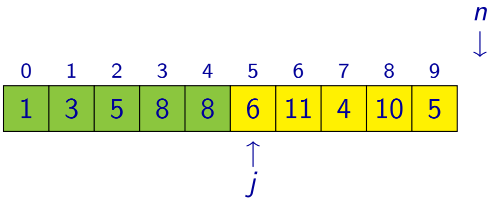
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

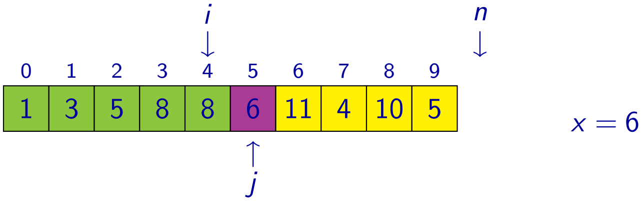


$x = 8$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

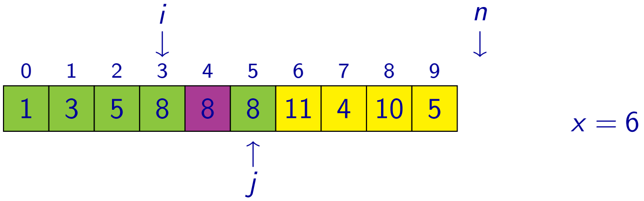
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

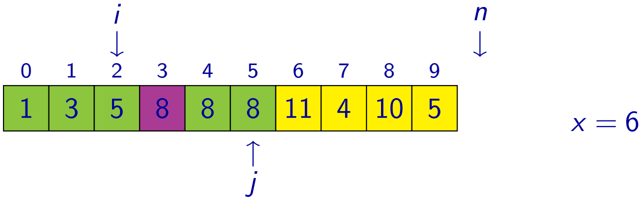
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

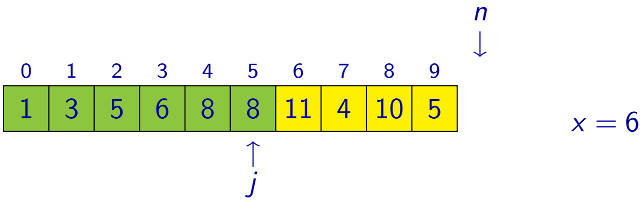
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

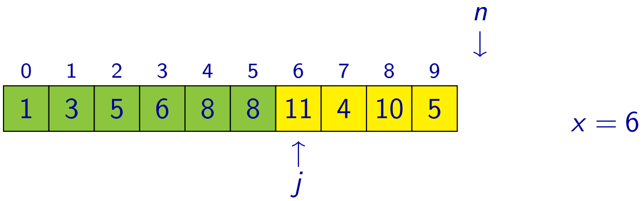
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

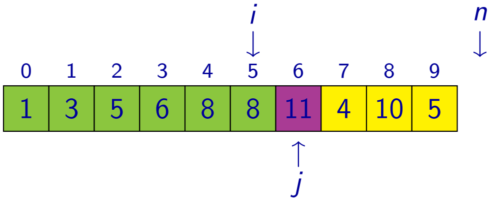
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$

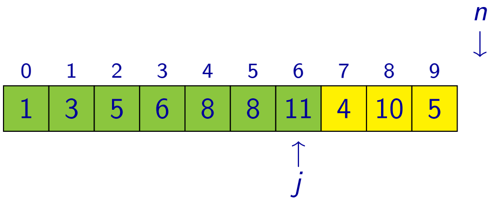


$x = 11$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

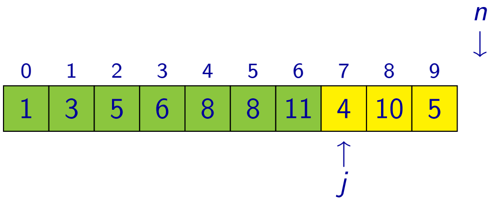


$x = 11$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

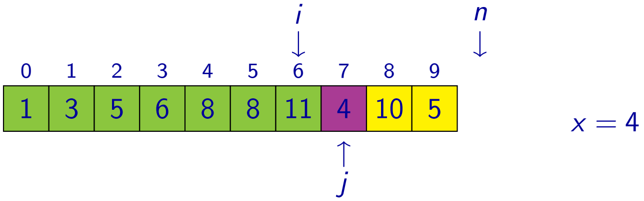


$x = 11$

Invariants

Example: A computation of algorithm Insertion-Sort for input

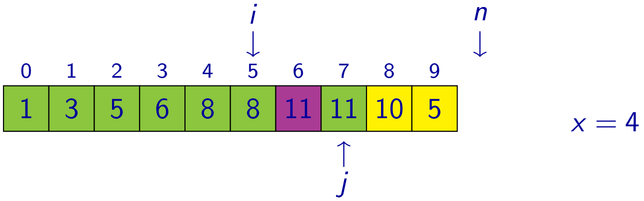
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

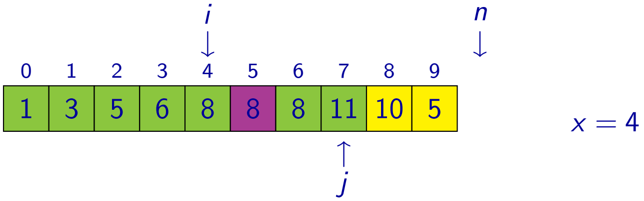
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

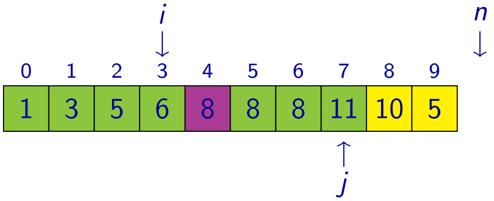
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

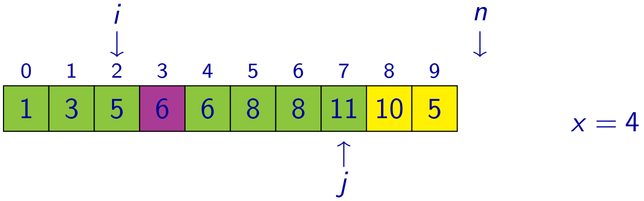


$x = 4$

Invariants

Example: A computation of algorithm Insertion-Sort for input

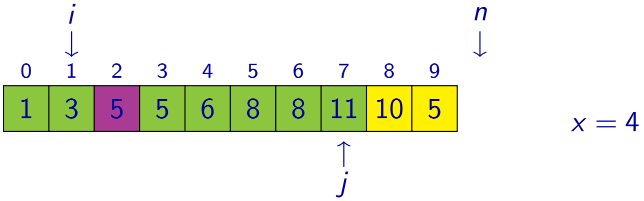
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

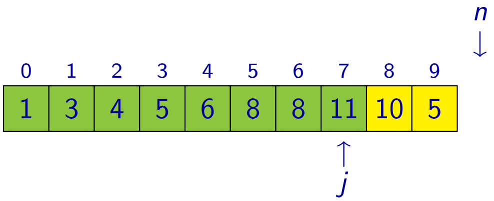
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

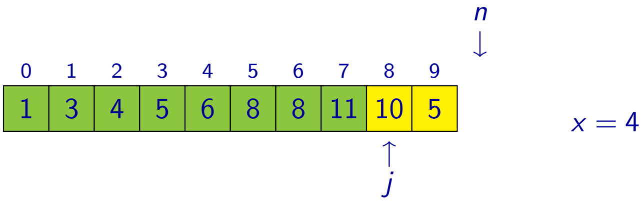


$x = 4$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

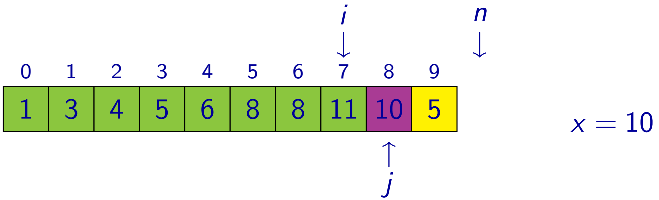
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

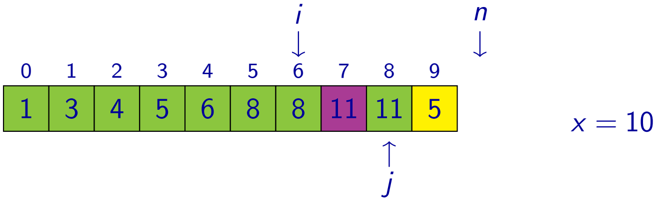
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

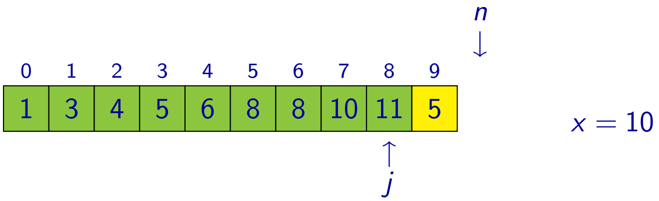
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

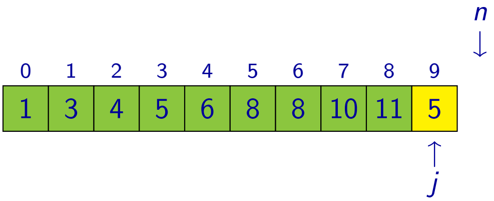
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

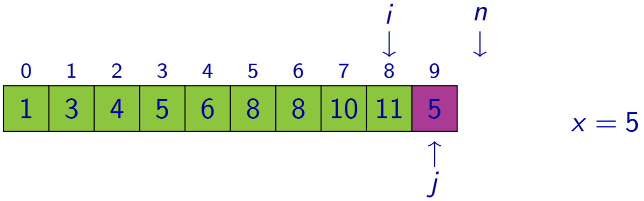


$x = 10$

Invariants

Example: A computation of algorithm INSERTION-SORT for input

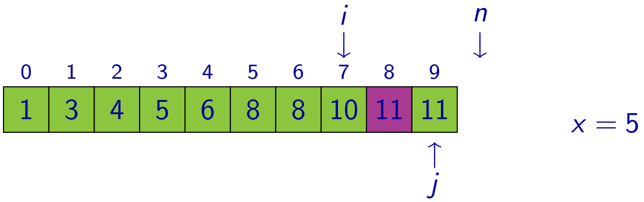
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

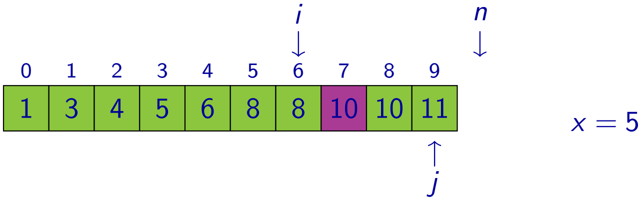
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

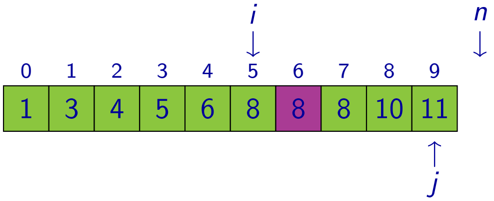
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$

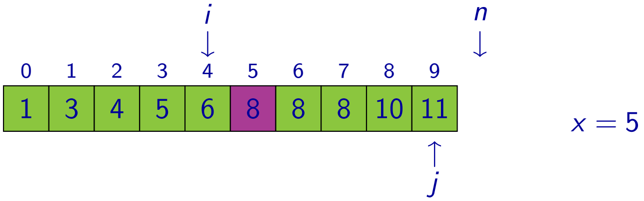


$x = 5$

Invariants

Example: A computation of algorithm Insertion-Sort for input

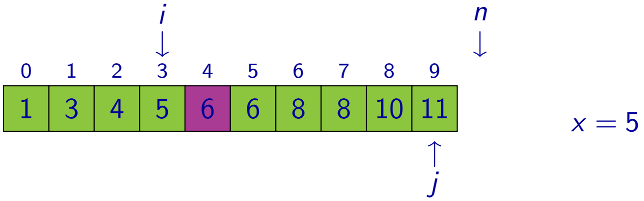
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm Insertion-Sort for input

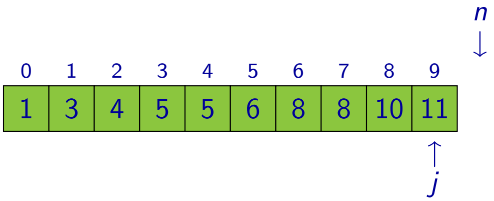
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Example: A computation of algorithm INSERTION-SORT for input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5],\ n = 10.$$

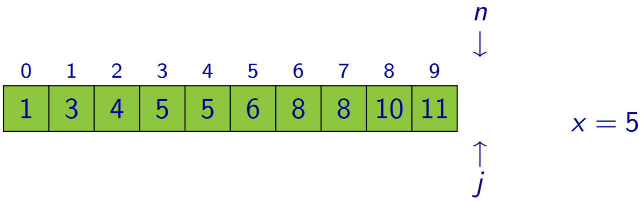


$x = 5$

Invariants

Example: A computation of algorithm Insertion-Sort for input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Invariants

Let us assume that the input is an array $A = [a_0, a_1, \ldots, a_{n-1}]$ and number n (where $n \geq 1$) specifying the length of this array, i.e., at the beginning, it holds for each i, where $0 \leq i < n$, that $A[i] = a_i$.

- At the beginning of the **for** cykle (i.e., always before executing test $j < n$, resp. $j \leq n - 1$), the following invariants hold:
  - $1 \leq j \leq n$
  - Elements of the array $A[0], A[1], \ldots, A[j-1]$ contain values $a_0, a_1, \ldots, a_{j-1}$ sorted from the smallest to the biggest, i.e.,
    $$A[0] \leq A[1] \leq \cdots \leq A[j-1]$$
  - Elements of the array $A[j], A[j+1], \ldots, A[n-1]$ contain values $a_j, a_{j+1}, \ldots, a_{n-1}$, i.e.,
    $$A[j] = a_j,\ A[j+1] = a_{j+1},\ \ldots,\ A[n-1] = a_{n-1}$$

Invariants

- At the beginning **while** cycle (i.e., always before executing test $i \geq 0$), the following invariants hold:
  - $1 \leq j < n$
  - $-1 \leq i < j$
  - Variable x contains value a_j, i.e., $x = a_j$.
  - Elements of the array $A[0], A[1], \ldots, A[i]$ and $A[i+2], A[i+3], \ldots, A[j]$ contain values $a_0, a_1, \ldots, a_{j-1}$ ordered from the smallest to the biggest, i.e.,
    $$A[0] \leq A[1] \leq \cdots \leq A[i] \leq A[i+2] \leq A[i+3] \leq \cdots \leq A[j]$$
  - All elements $A[i+2], A[i+3], \ldots, A[j]$ are strictly greater than x.
  - Elements of the array $A[j+1], A[j+2], \ldots, A[n-1]$ contain values $a_{j+1}, a_{j+2}, \ldots, a_{n-1}$, i.e.,
    $$A[j+1] = a_{j+1},\ A[j+2] = a_{j+2},\ \ldots,\ A[n-1] = a_{n-1}$$

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

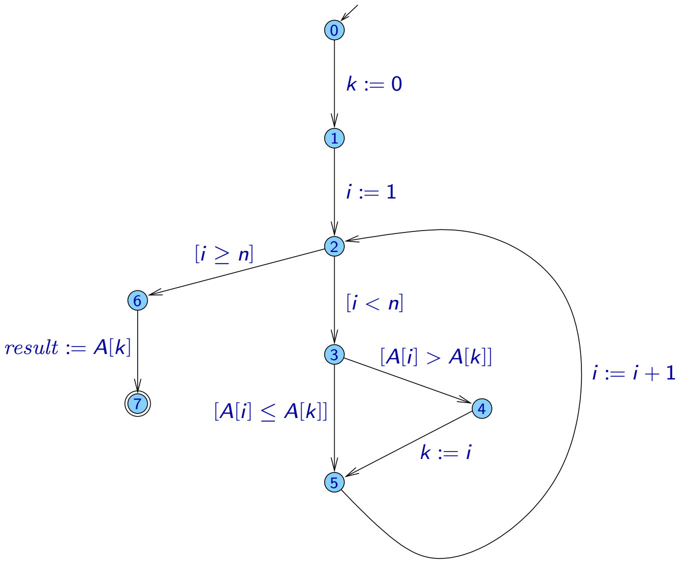
As a set W, we can use for example:

- 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

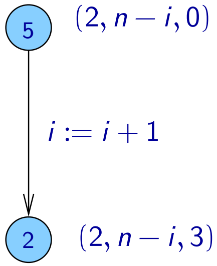


Finiteness of a Computation

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: $f(\alpha) = (0)$

Finiteness of a Computation



We must take into account that the value of variable i is modified by this instruction.

A transition from a configuration with assigned vector $(2, n-i, 0)$ to a configuration with assigned vector $(2, n-i', 3)$, where $i' = i+1$.

It is obvious that $n-i' < n-i$, since $n-(i+1) < n-i$.

Computational Complexity of Algorithms

Complexity of an Algorithm

- Computers work fast but not infinitely fast. Execution of each instruction takes some (very short) time.
- The same problem can be solved by several different algorithms. The time of a computation (determined mostly by the number of executed instructions) can be different for different algorithms.
- We would like to compare different algorithms and choose a better one.
- We can implement the algorithms and then measure the time of their computation. By this we find out how long the computation takes on particular data on which we test the algorithm.
- We would like to have a more precise idea how long the computation takes on all possible input data.

Complexity of an Algorithm

- A running time is affected by many factors, e.g.:
  - the algorithm that is used
  - the amount of input data
  - used hardware (e.g., the frequency at which a CPU is running can be important)
  - the used programming language — its implementation (compiler/interpreter)
  - ...
- If we need to solve problem for “small” input data, the running time is usually negligible.
- With increasing amount of input data (the size of input), the running time can grow, sometimes significantly.

Complexity of an Algorithm

- **Time complexity of an algorithm** — how the running time of the algorithm depends on the amount of input data
- **Space complexity of an algorithm** — how the amount of a memory used during a computation grows with respect to the size of input

Remark: The precise definitions of these notion will be given in a moment.

Remark:

- There are also other types of computational complexity, which we will not discuss here (e.g., communication complexity).

Complexity of an Algorithm

Consider some particular machine executing some algorithm —
e.g., a random-access machine, a Turing machine, . . .

We will assume that for the given machine $\mathcal{M}$ we have somehow defined for every input w from the set of all possible inputs In the following two functions:

- $time_{\mathcal{M}} : In \rightarrow \mathbb{N}$ — it expresses the running time of machine $\mathcal{M}$ on input w
- $space_{\mathcal{M}} : In \rightarrow \mathbb{N}$ — it expresses the amount of memory used by machine $\mathcal{M}$ in a computation on input w

Remark: We assume that a computation on an arbitraty input w will halt after some finite number of steps.

Complexity of an Algorithm

Example:

- One-tape Turing machine $\mathcal{M}$:
  - $time_{\mathcal{M}}(w)$ — the number of steps performed by during a computation on word w
  - $space_{\mathcal{M}}(w)$ — the number of cells on the tape visited during a computation on input w

- Random-access machine:
  - $time_{\mathcal{M}}(w)$ — the number of steps performed by the given RAM in a computation on input w
  - $space_{\mathcal{M}}(w)$ — the number of memory cells that were used during a computation on input w (in they were written to or if a value was read from them)

Size of Input

For different input data the program performs a different number of instructions.

If we want to analyze somehow the number of performed instructions, it is useful to introduce the notion of the **size of an input**.

Typically, the size of an input is a number specifying how “big” is the given instance (a bigger number means a bigger instance).

Remark: We can define the size of an input as we like depending on what is useful for our analysis.

The size of an input is not strictly determinable but there are usually some natural choices based on the nature of the problem.

Size of Input

Examples:

- For the problem "Sorting", where the input is a sequence of numbers $a_1, a_2, \ldots, a_n$ and the output the same sequence sorted, we can take n as the size of the input.

- For the problem "Primality" where the input is a natural number x and where the question is whether x is a prime, we can take the number of bits of the number x as the size of the input.

  (The other possibility is to take directly the value x as the size of the input.)

Size of Input

Sometimes it is useful to describe the size of an input with several numbers.

For example for problems where the input is a graph, we can define the size of the input as a pair of numbers n, m where:

- n – the number of nodes of the graph
- m – the number of edges of the graph

Remark: The other possibility is to define the size of the input as one number $n + m$.

Size of Input

In general, we can define the size of an input for an arbitrary problem as follows:

- When the input is a word over some alphabet Σ:
  the length of word w
- When the input as a sequence of bits (i.e., a word over $\{0, 1\}$):
  the number of bits in this sequence
- When the input is a natural number x:
  the number of bits in the binary representation of x

Time Complexity

We want to analyze a particular algorithm (its particular implementation).

We want to know how many steps the algorithm performs when it gets an input of size $0, 1, 2, 3, 4, \ldots$.

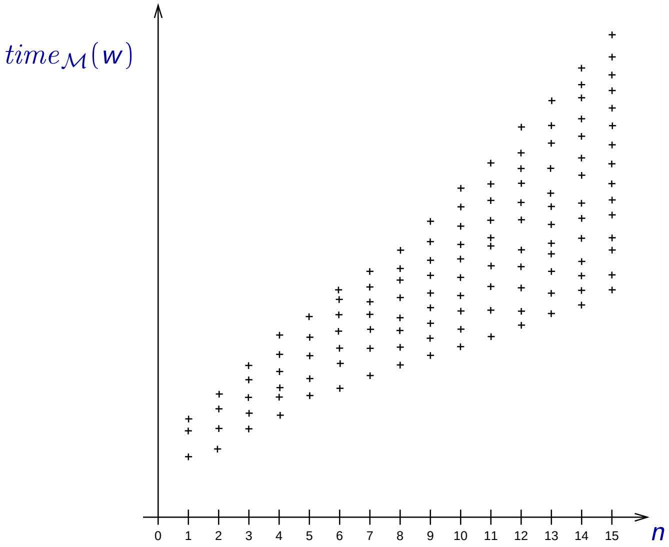
It is obvious that even for inputs of the same size the number of performed steps can be different.

Let us denote the size of input $w \in \mathit{In}$ as $\mathit{size}(w)$.

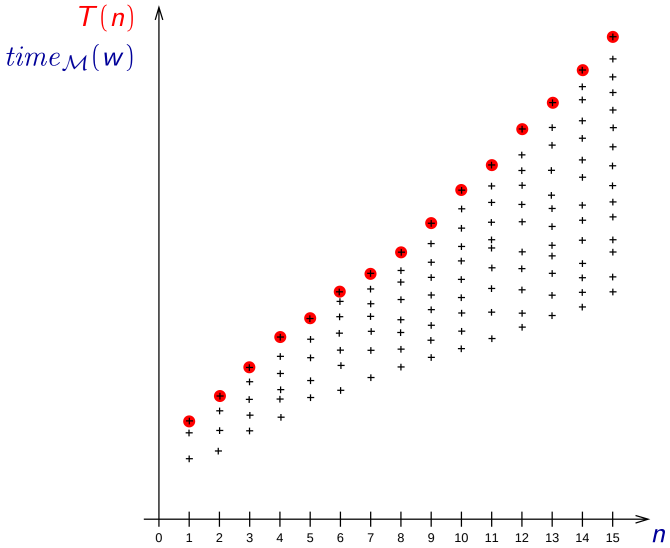
Now we define a function $T : \mathbb{N} \to \mathbb{N}$ such that for $n \in \mathbb{N}$ is

$$T(n) = \max\{\, \mathit{time}_{\mathcal{M}}(w) \mid w \in \mathit{In},\ \mathit{size}(w) = n \,\}$$

Time Complexity in the Worst Case



Time Complexity in the Worst Case



Time Complexity in the Worst Case

Such function $T(n)$ (i.e., a function that for the given algorithm and the given definition of the size of an input assignes to every natural number n the maximal number of instructions performed by the algorithm if it obtains an input of size n) is called the **time complexity of the algorithm in the worst case**.

$$T(n) = \max\{\, time_{\mathcal{M}}(w) \mid w \in In,\ size(w) = n \,\}$$

Analogously, we can define **space complexity** of the algorithm in the worst case as a function $S(n)$ where:

$$S(n) = \max\{\, space_{\mathcal{M}}(w) \mid w \in In,\ size(w) = n \,\}$$

Time Complexity in an Average Case

Sometimes it make sense to analyze the time complexity **in an average case**.

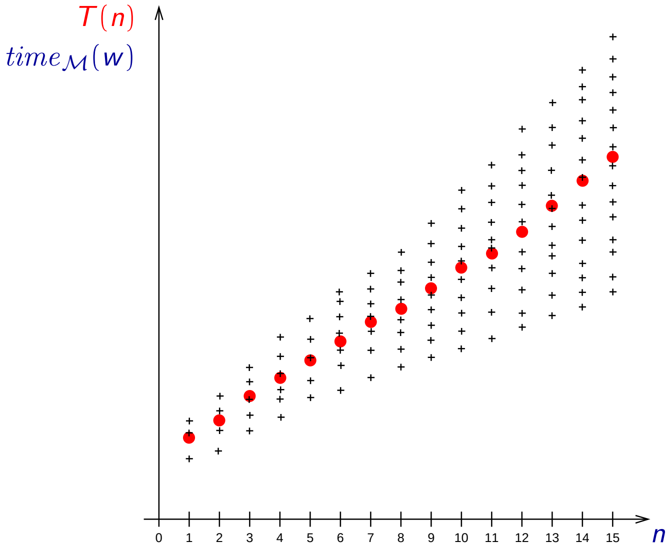
In this case, we do not define $T(n)$ as the maximum but as the arithmetic mean of the set

$$\{\, \mathit{time}_{\mathcal{M}}(w) \mid w \in \mathit{In},\ \mathit{size}(w) = n \,\}$$

- It is usually more difficult to determine the time complexity in an average case than to determine the time complexity in the worst case.
- Often, these two function are not very different but sometimes the difference is significant.

Remark: It usually makes no sense to analyze the time complexity in the best case.

Time Complexity in an Average Case



Computational Complexity of an Algorithm

It is obvious from this definition that the time complexity of an algorithm is a function whose precise values depend not only on the given algorithm Alg but also on the following things:

- on a machine $\mathcal{M}$, on which the algorithm Alg runs,
- on the precise definition of the running time $t(w)$ of algorithm Alg on machine $\mathcal{M}$ with input $w \in In$,
- on the precise definition of the size of an input (i.e., on the definition of function $size$).

Computational Complexity of an Algorithm

To determine the precise running time or the precise amount of used memory just by an analysis of an algorithm can be extremely difficult.

Usually the analysis of complexity of an algorithm involves many simplifications:

- It is usually not analysed how the running time or the amount of used memory depends precisely on particular input data but how they depend on the **size of the input**.
- Functions expressing how the running time or the amount of used memory grows depending on the size of the input are not computed precisely — instead **estimations** of these functions are computed.
- Estimations of these functions are usually expressed using **asymptotic notation** — e.g., it can be said that the running time of MergeSort is $O(n \log n)$, and that the running time of BubbleSort is $O(n^2)$.

Time Complexity of an Algorithm

An example of an analysis of the time complexity of algorithm **without** the use of asymptotic notation:

- Such precise analysis is almost never done in practice — it is too tedious and complicated.
- This illustrates what things are ignored in an analysis where asymptotic notation is used and how much the analysis is simplified by this.
- We will compute with constants $c_0, c_1, \ldots, c_k$, which specify the execution time of individual instructions — we won't compute with concrete numbers.

Running Time

Let us say that an algorithm is represented by a control-flow graph:

- To every instruction (i.e., to every edge) we assign a value specifying how long it takes to perform this instruction once.
- The execution time of different instructions can be different.
- For simplicity we assume that an execution of the same instruction takes always the same time — the value assigned to an instruction is a number from the set $\mathbb{R}_+$ (the set of nonnegative real numbers).

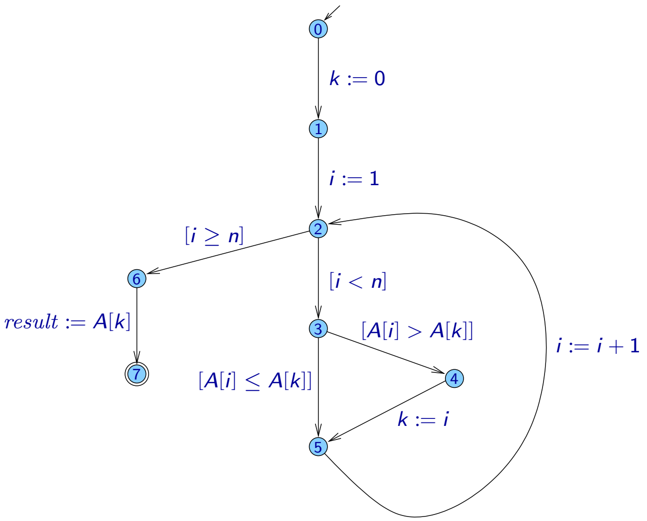
Running Time

Example:

Algorithm: 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 $A[k]$
```

Running Time



Instr.	time
$k := 0$	c_0
$i := 1$	c_1
$[i < n]$	c_2
$[i \geq n]$	c_3
$[A[i] \leq A[k]]$	c_4
$[A[i] > A[k]]$	c_5
$k := i$	c_6
$i := i + 1$	c_7
$result := A[k]$	c_8

Running Time

Example: The execution times of individual instructions could be for example:

Instr.	symbol	time
$k := 0$	c_0	4
$i := 1$	c_1	4
$[i < n]$	c_2	10
$[i \geq n]$	c_3	12
$[A[i] \leq A[k]]$	c_4	14
$[A[i] > A[k]]$	c_5	12
$k := i$	c_6	5
$i := i + 1$	c_7	6
$\mathit{result} := A[k]$	c_8	5

For a particular input w, e.g., for $w = ([3, 8, 4, 5, 2], 5)$, we could simulate the computation and determine the precise running time $t(w)$.

Time Complexity of an Algorithm

The inputs are of the form (A, n), where A is an array and n is the number of elements in this array (where $n \geq 1$).

We take n as the size of input (A, n).

Consider now some particular input $w = (A, n)$ of size n:

- The running time $t(w)$ on input w can be expressed as

$$t(w) = c_0 \cdot m_0(w) + c_1 \cdot m_1(w) + \cdots + c_8 \cdot m_8(w),$$

  where $m_0, m_1, \ldots, m_8$ are functions specifying how many times is each instruction performed in the computation on input w.

Time Complexity of an Algorithm

Instr.	time	occurences	value of $m_i(w)$
$k := 0$	c_0	$m_0(w)$	1
$i := 1$	c_1	$m_1(w)$	1
$[i < n]$	c_2	$m_2(w)$	$n-1$
$[i \geq n]$	c_3	$m_3(w)$	1
$[A[i] \leq A[k]]$	c_4	$m_4(w)$	$n-1-\ell$
$[A[i] > A[k]]$	c_5	$m_5(w)$	ℓ
$k := i$	c_6	$m_6(w)$	ℓ
$i := i+1$	c_7	$m_7(w)$	$n-1$
$\mathit{result} := A[k]$	c_8	$m_8(w)$	1

ℓ — the number of iterations of the cycle where $A[i] > A[k]$
(obviously $0 \leq \ell < n$)

Time Complexity of an Algorithm

By assigning values to

$$t(w) = c_0 \cdot m_0(w) + c_1 \cdot m_1(w) + \cdots + c_8 \cdot m_8(w),$$

we obtain

$$t(w) = d_1 + d_2 \cdot (n-1) + d_3 \cdot (n-1-\ell) + d_4 \cdot \ell,$$

where

$$\begin{aligned} d_1 &= c_0 + c_1 + c_3 + c_8 & \qquad d_3 &= c_4 \\ d_2 &= c_2 + c_7 & \qquad d_4 &= c_5 + c_6 \end{aligned}$$

After simplification we have

$$t(w) = (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot \ell + (d_1 - d_2 - d_3)$$

Remark: $t(w)$ is not the time complexity but the running time for a particular input w

Time Complexity of an Algorithm

For example, if the execution times of instructions will be:

Instr.	symb.	time
$k := 0$	c_0	4
$i := 1$	c_1	4
$[i < n]$	c_2	10
$[i \geq n]$	c_3	12
$[A[i] \leq A[k]]$	c_4	14
$[A[i] > A[k]]$	c_5	12
$k := i$	c_6	5
$i := i + 1$	c_7	6
$result := A[k]$	c_8	5

then $d_1 = 25$, $d_2 = 16$, $d_3 = 14$, and $d_4 = 17$.

In this case is $t(w) = 30n + 3\ell - 5$.

For the input $w = ([3, 8, 4, 5, 2], 5)$ is $n = 5$ and $\ell = 1$, therefore $t(w) = 30 \cdot 5 + 3 \cdot 1 - 5 = 148$.

Time Complexity of an Algorithm

It can depend on details of implementation and on the precise values of constants, for which inputs of size n the compution takes the longest time (i.e., which are the worst cases):

The running time of algorithm FIND-MAX for an input $w = (A, n)$ of size n:

$$t(w) \ = \ (d_2 + d_3) \cdot n \ + \ (d_4 - d_3) \cdot \ell \ + \ (d_1 - d_2 - d_3)$$

- If $d_3 \geq d_4$ — the worst cases are those where ℓ has the smallest value $\ell = 0$ — for example inputs of the form $[0, 0, \ldots, 0]$ or of the form $[n,\ n-1,\ n-2,\ \ldots,\ 2,\ 1]$
- If $d_3 \leq d_4$ — the worst are those cases where ℓ has the greatest value $\ell = n - 1$ — for example inputs of the form $[0, 1, \ldots, n-1]$

Time Complexity of an Algorithm

The time complexity $T(n)$ of algorithm Find-Max in the worst case is given as follows:

- If $d_3 \geq d_4$:

$$T(n) = (d_2 + d_3) \cdot n + (d_1 - d_2 - d_3)$$

- If $d_3 \leq d_4$:

$$\begin{aligned} T(n) &= (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot (n - 1) + (d_1 - d_2 - d_3) \\ &= (d_2 + d_4) \cdot n + (d_1 - d_2 - d_4) \end{aligned}$$

Example: For $d_1 = 25$, $d_2 = 16$, $d_3 = 14$, $d_4 = 17$ is

$$\begin{aligned} T(n) &= (16 + 17) \cdot n + (25 - 16 - 17) \\ &= 33n - 8 \end{aligned}$$

Time Complexity of an Algorithm

In both cases (when $d_3 \geq d_4$ or when $d_3 \leq d_4$), the time complexity of the algorithm Find-Max is a function

$$T(n) = an + b$$

where a and b are some constants whose precise values depend on the execution time of individual instructions.

Remark: These constants could be expressed as

$$a = d_2 + \max\{d_3, d_4\} \qquad b = d_1 - d_2 - \max\{d_3, d_4\}$$

For example

$$T(n) = 33n - 8$$

Time Complexity of an Algorithm

If it would be sufficient to find out that the time complexity of the algorithm Find-Max is some function of the form

$$T(n) = an + b,$$

where the precise values of constants a and b would not be important for us, the whole analysis could be considerably simpler.

- In fact, we usually do not want to know precisely how function $T(n)$ look (in general, it can be a very complicated function), and it would be sufficient to know that values of the function $T(n)$ "approximately" correspond to values of a function $S(n) = an + b$, where a and b are some constants.

Time Complexity of an Algorithm

For a given function $T(n)$ expressing the time or space complexity, it is usually sufficient to express it approximately — to have an **estimation** where

- we ignore the less important parts

  (e.g., in function $T(n) = 15n^2 + 40n - 5$ we can ignore $40n$ and -5, and to consider function $T(n) = 15n^2$ instead of the original function),

- we ignore multiplication constants

  (e.g., instead of function $T(n) = 15n^2$ we will consider function $T(n) = n^2$)

- we won't ignore constants in exponents — for example there is a big difference between functions $T_1(n) = n^2$ and $T_2(n) = n^3$.

- we will be interested how function $T(n)$ behaves for "big" values of n, we can ignore its behaviour on small values

Growth of Functions

A program works on an input of size n.
Let us assume that for an input of size n, the program performs $T(n)$ operations and that an execution of one operation takes $1\,\mu\mathrm{s}$ ($10^{-6}\,\mathrm{s}$).

	n							
$T(n)$	20	40	60	80	100	200	500	1000
n	20 µs	40 µs	60 µs	80 µs	0.1 ms	0.2 ms	0.5 ms	1 ms
$n \log n$	86 µs	0.213 ms	0.354 ms	0.506 ms	0.664 ms	1.528 ms	4.48 ms	9.96 ms
n^2	0.4 ms	1.6 ms	3.6 ms	6.4 ms	10 ms	40 ms	0.25 s	1 s
n^3	8 ms	64 ms	0.216 s	0.512 s	1 s	8 s	125 s	16.7 min.
n^4	0.16 s	2.56 s	12.96 s	42 s	100 s	26.6 min.	17.36 hours	11.57 days
2^n	1.05 s	12.75 days	36560 years	$38.3\cdot10^{9}$ years	$40.1\cdot10^{15}$ years	$50\cdot10^{45}$ years	$10.4\cdot10^{136}$ years	–
$n!$	77147 years	$2.59\cdot10^{34}$ years	$2.64\cdot10^{68}$ years	$2.27\cdot10^{105}$ years	$2.96\cdot10^{144}$ years	–	–	–

Growth of Functions

Let us consider 3 algorithms with complexities $T_1(n) = n$, $T_2(n) = n^3$, $T_3(n) = 2^n$. Our computer can do in a reasonable time (the time we are willing to wait) 10^{12} steps.

Complexity	Input size
$T_1(n) = n$	10^{12}
$T_2(n) = n^3$	10^4
$T_3(n) = 2^n$	40

Growth of Functions

Let us consider 3 algorithms with complexities $T_1(n) = n$, $T_2(n) = n^3$, $T_3(n) = 2^n$. Our computer can do in a reasonable time (the time we are willing to wait) 10^{12} steps.

Complexity	Input size
$T_1(n) = n$	10^{12}
$T_2(n) = n^3$	10^4
$T_3(n) = 2^n$	40

Now we speed up our computer 1000 times, meaning it can do 10^{15} steps.

Complexity	Input size	Growth
$T_1(n) = n$	10^{15}	$1000\times$
$T_2(n) = n^3$	10^5	$10\times$
$T_3(n) = 2^n$	50	$+10$

Asymptotic Notation

In the following, we will consider functions of the form $f : \mathbb{N} \to \mathbb{R}$, where:

- The values of $f(n)$ need not to be defined for all values of $n \in \mathbb{N}$ but there must exist some constant n_0 such that the value of $f(n)$ is defined for all $n \in \mathbb{N}$ such that $n \geq n_0$.

  Example: Function $f(n) = \log_2(n)$ is not defined for $n = 0$ but it is defined for all $n \geq 1$.

- There must exist a constant n_0 such that for all $n \in \mathbb{N}$, where $n \geq n_0$, is $f(n) \geq 0$.

  Example: It holds for function $f(n) = n^2 - 25$ that $f(n) \geq 0$ for all $n \geq 5$.

Asymptotic Notation

Let us take an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. Expressions $O(g)$, $\Omega(g)$, $\Theta(g)$, $o(g)$, and $\omega(g)$ denote **sets of functions** of the type $\mathbb{N} \to \mathbb{R}$, where:

- $O(g)$ – the set of all functions that grow at most as fast as g
- $\Omega(g)$ – the set of all functions that grow at least as fast as g
- $\Theta(g)$ – the set of all functions that grow as fast as g
- $o(g)$ – the set of all fuctions that grow slower than function g
- $\omega(g)$ – the set of all functions that grow faster than function g

Remark: These are not definitions! The definitions will follow on the next slides.

- O – big "O"
- Ω – uppercase Greek letter "omega"
- Θ – uppercase Greek letter "theta"
- o – small "o"
- ω – small "omega"

Asymptotic Notation – Symbol O

Informally:

$O(g)$ – the set of all functions that grow at most as fast as g

How to define formally when $f \in O(g)$ holds?

The first try:

- to compare the values of the functions

$$(\forall n \in \mathbb{N})(f(n) \leq g(n))$$

A problem: This does not allow to ignore the values of constants, e.g., it is not true that $(\forall n \in \mathbb{N})(3n^2 \leq 2n^2)$.

Asymptotic Notation – Symbol O

Informally:

$O(g)$ – the set of all functions that grow at most as fast as g

How to define formally when $f \in O(g)$ holds?

The second try:

- to multiply function g with some big enough constant c

$$(\exists c > 0)(\forall n \in \mathbb{N})(f(n) \leq c \cdot g(n))$$

A problem: The inequality need not hold for some small values of n even after multiplying g by some arbitralily big value.

For example, function $g(n) = n^2$ grows faster than function $f(n) = n + 5$. However, not matter how big constant c is chosen, it will never be true that $n + 5 \leq c \cdot n^2$ for $n = 0$.

Asymptotic Notation – Symbol O

Informally:

$O(g)$ – the set of all functions that grow at most as fast as g

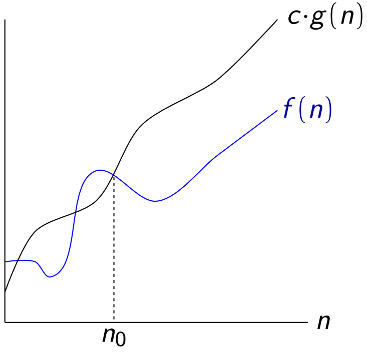
How to define formally when $f \in O(g)$ holds?

The third try:

- it is not required that the inequality holds for each n, it is sufficient when it holds for all values that are "big enough"

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \leq c \cdot g(n))$$

Asymptotic Notation – Symbol O



Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in O(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\big(f(n) \leq c \cdot g(n)\big).$$

Asymptotic Notation – Symbol O

Remarks:

- c is a posive real number (i.e., $c \in \mathbb{R}$ and $c > 0$)
- n_0 and n are natural numbers (i.e., $n_0 \in \mathbb{N}$ and $n \in \mathbb{N}$)

Asymptotic Notation – Symbol O

Example: Let us consider functions $f(n) = 2n^2 + 3n + 7$ and $g(n) = n^2$.

We want to show that $f \in O(g)$, i.e., $f \in O(n^2)$:

- **Approach 1:**

  Let us take for example $c = 3$.

  $$c \cdot g(n) = 3n^2 = 2n^2 + \frac{1}{2}n^2 + \frac{1}{2}n^2$$

  We need to find some n_0 such that for all $n \geq n_0$ it holds that

  $$2n^2 \geq 2n^2 \qquad \frac{1}{2}n^2 \geq 3n \qquad \frac{1}{2}n^2 \geq 7$$

  We can easily check that for example $n_0 = 6$ satisfies this.

  For each $n \geq 6$ we have $c \cdot g(n) \geq f(n)$:

  $$cg(n) = 3n^2 = 2n^2 + \frac{1}{2}n^2 + \frac{1}{2}n^2 \geq 2n^2 + 3n + 7 = f(n)$$

Asymptotic Notation – Symbol O

The example where $f(n) = 2n^2 + 3n + 7$ and $g(n) = n^2$:

- **Approach 2:**

  Let us take $c = 12$.

  $$c \cdot g(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2$$

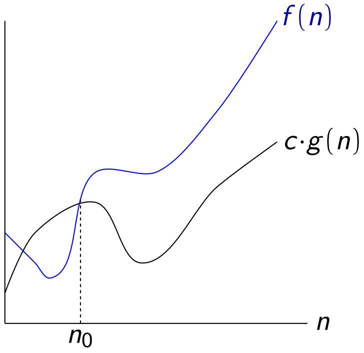
  We need to find some n_0 such that for all $n \geq n_0$ we have

  $$2n^2 \geq 2n^2 \qquad 3n^2 \geq 3n \qquad 7n^2 \geq 7$$

  These inequalities obviously hold for $n_0 = 1$, and so for each $n \geq 1$ we have $f(n) \leq c \cdot g(n)$:

  $$c \cdot g(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2 \geq 2n^2 + 3n + 7 = f(n)$$

Asymptotic Notation – Symbol Ω



Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in \Omega(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\big(c \cdot g(n) \leq f(n)\big).$$

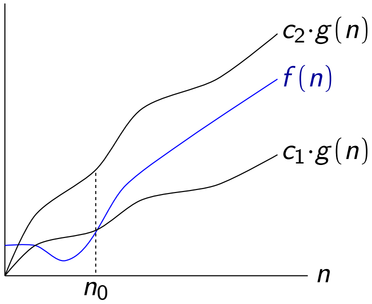
Asymptotic Notation – Symbol Ω

It is not difficult to prove the following proposition:

For arbitrary functions f and g we have:

$$f \in O(g) \qquad \text{iff} \qquad g \in \Omega(f)$$

Asymptotic Notation – Symbol Θ



Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in \Theta(g)$ iff

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\big(c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\big).$$

Asymptotic Notation – Symbol Θ

The following easily follows from the definition of Θ:

For arbitrary functions f and g we have:

$$
\begin{array}{lcl}
f \in \Theta(g) & \text{iff} & f \in O(g) \text{ and } f \in \Omega(g) \\
f \in \Theta(g) & \text{iff} & f \in O(g) \text{ and } g \in O(f) \\
f \in \Theta(g) & \text{iff} & g \in \Theta(f)
\end{array}
$$

Asymptotic Notation – Symbols o and ω

Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in o(g)$ iff

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$$

Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in \omega(g)$ iff

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$$

Asymptotic Notation

For arbitrary functions f and g we have the following propositions:

If there exists a constant $c \geq 0$ such that

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = c$$

then $f \in O(g)$.

If there exists a constant $c \geq 0$ such that

$$\lim_{n \to +\infty} \frac{g(n)}{f(n)} = c$$

then $f \in \Omega(g)$.

Asymptotic Notation

It is obvious that:

- If $f \in o(g)$ then $f \in O(g)$.
- If $f \in \omega(g)$ then $f \in \Omega(g)$.

Asymptotic Notation

The asymptotic notation can be viewed as a certain kind of comparison of a **rate of growth** functions:

$f \in O(g)$	—	rate of growth of f "$\leq$" rate of growth of g
$f \in \Omega(g)$	—	rate of growth of f "$\geq$" rate of growth of g
$f \in \Theta(g)$	—	rate of growth of f "$=$" rate of growth of g
$f \in o(g)$	—	rate of growth of f "$<$" rate of growth of g
$f \in \omega(g)$	—	rate of growth of f "$>$" rate of growth of g

Remark:

- There are pairs of functions f and g such that

  $$f \notin O(g) \qquad \text{and} \qquad g \notin O(f),$$

  for example

  $$f(n) = n^2 \qquad\qquad g(n) = \begin{cases} n & \text{if } n \mod 2 = 1 \\ n^3 & \text{otherwise} \end{cases}$$

Asymptotic Notation

- A function f is called:

  linear, if $f(n) \in \Theta(n)$
  quadratic, if $f(n) \in \Theta(n^2)$
  cubic, if $f(n) \in \Theta(n^3)$
  polynomial, if $f(n) \in O(n^k)$ for some $k > 0$
  exponential, if $f(n) \in O(c^{n^k})$ for some $c > 1$ and $k > 0$
  logarithmic, if $f(n) \in \Theta(\log n)$
  polylogarithmic, if $f(n) \in \Theta(\log^k n)$ for some $k > 0$

- $O(1)$ is the set of all **bounded** functions, i.e., functions whose function values can be bounded from above by a constant.

- Exponential functions are often written in the form $2^{O(n^k)}$ when the asymptotic notation is used, since then we do not need to consider different bases.

Asymptotic Notation

In general, it holds that:

- every polylogarithmic function grows slower than any polynomial function
- every polynomial function grows slower than any exponential function
- to compare polynomial functions n^k and n^ℓ it is sufficient to compare values k and ℓ
- to compare polylogarithmic functions $\log^k n$ and $\log^\ell n$ it is sufficient to compare values k and ℓ
- to compare exponential functions $2^{p(n)}$ and $2^{q(n)}$ it is sufficient to compare polynomials $p(n)$ and $q(n)$.

Asymptotic Notation

Proposition

Let us assume that a and b are constants such that $a > 0$ and $b > 0$, and k and ℓ are some arbitrary constants where $k \geq 0$, $\ell \geq 0$ and $k \leq \ell$.

Let us consider functions

$$f(n) = a \cdot n^k \qquad\qquad g(n) = b \cdot n^\ell$$

For each such functions f and g it holds that $f \in O(g)$:

Proof: Let us take $c = \frac{a}{b}$.

Because for $n \geq 1$ we obviously have $n^k \leq n^\ell$ (since $k \leq \ell$), for $n \geq 1$ we have

$$c \cdot g(n) = \tfrac{a}{b} \cdot g(n) = \tfrac{a}{b} \cdot b \cdot n^\ell = a \cdot n^\ell \geq a \cdot n^k = f(n)$$

Asymptotic Notation

Proposition

For any $a, b > 1$ and any $n > 0$ we have

$$\log_a n = \frac{\log_b n}{\log_b a}$$

Proof: From $n = a^{\log_a n}$ it follows that $\log_b n = \log_b(a^{\log_a n})$.
Since $\log_b(a^{\log_a n}) = \log_a n \cdot \log_b a$, we obtain $\log_b n = \log_a n \cdot \log_b a$, from which the above mentioned conclusion follows directly. $\square$

Due to this observation, the base of a logarithm is often omited in the asymptotic notation: for example, instead of $\Theta(n \log_2 n)$ we can write $\Theta(n \log n)$.

Asymptotic Notation

Examples:

$n \in O(n^2)$
$1000n \in O(n)$
$2^{\log_2 n} \in \Theta(n)$
$n^3 \notin O(n^2)$
$n^2 \notin O(n)$
$n^3 + 2^n \notin O(n^2)$

$n^3 \in O(n^4)$
$0.00001n^2 - 10^{10}n \in \Theta(10^{10}n^2)$
$n^3 - n^2 \log_2^3 n + 1000n - 10^{100} \in \Theta(n^3)$
$n^3 + 1000n - 10^{100} \in O(n^3)$
$n^3 + n^2 \notin \Theta(n^2)$
$n! \notin O(2^n)$

Asymptotic Notation

For arbitrary functions f, g, and h we have:

- if $f \in O(g)$ and $g \in O(h)$ then $f \in O(h)$
- if $f \in \Omega(g)$ and $g \in \Omega(h)$ then $f \in \Omega(h)$
- if $f \in \Theta(g)$ and $g \in \Theta(h)$ then $f \in \Theta(h)$

Asymptotic Notation

- For any function f a libovolnou konstantu $c > 0$ we have:
  - $c \cdot f \in \Theta(f)$
- For any pair of functions f, g we have:
  - $\max(f, g) \in \Theta(f + g)$
  - if $f \in O(g)$ then $f + g \in \Theta(g)$
- For any functions f_1, f_2, g_1, g_2 we have:
  - if $f_1 \in O(f_2)$ and $g_1 \in O(g_2)$ then $f_1 + g_1 \in O(f_2 + g_2)$ and $f_1 \cdot g_1 \in O(f_2 \cdot g_2)$
  - if $f_1 \in \Theta(f_2)$ and $g_1 \in \Theta(g_2)$ then $f_1 + g_1 \in \Theta(f_2 + g_2)$ and $f_1 \cdot g_1 \in \Theta(f_2 \cdot g_2)$

Asymptotic Notation

As mentioned before, expressions $O(g)$, $\Omega(g)$, $\Theta(g)$, $o(g)$, and $\omega(g)$ denote certain sets of functions.

In some texts, these expressions are sometimes used with a slightly different meaning:

- an expression $O(g)$, $\Omega(g)$, $\Theta(g)$, $o(g)$ or $\omega(g)$ does not represent the corresponding set of functions but **some** function from this set.

This convention is often used in equations and inequations.

Example: $3n^3 + 5n^2 - 11n + 2 = 3n^3 + O(n^2)$

When using this convention, we can for example write $f = O(g)$ instead of $f \in O(g)$.

Complexity of Algorithms

Let us say we would like to analyze the time complexity $T(n)$ of some algorithm consisting of instructions $I_1, I_2, \ldots, I_k$:

- Let us assume that how long it takes to execute each instruction is given by constants $c_1, c_2, \ldots, c_k$, i.e., the time it takes to execute instruction I_i once is specified by a constant c_i.

- Let us assume that In is the set of all possible inputes for the given algorithm.

  Let us define for each instruction I_i a corresponding function

  $$m_i : In \to \mathbb{N}$$

  specifying how many times instruction I_i will be executed during a computation over a given input, i.e., the value $m_i(w)$ specifies how many times instruction I_i will be executed during a computation over an input w.

Complexity of Algorithms

- Total running time of a computation over an input w:
$$t(w) = c_1 \cdot m_1(w) + c_2 \cdot m_2(w) + \cdots + c_k \cdot m_k(w).$$
- Let us racall that $T(n) = \max\{\, t(w) \mid \mathit{size}(w) = n \,\}$.
- For each of the functions $m_1, m_2, \ldots, m_k$ we can define a corresponding function $f_i : \mathbb{N} \to \mathbb{R}$, where
$$f_i(n) = \max\{\, m_i(w) \mid \mathit{size}(w) = n \,\}$$
is the maximum of numbers of executions of instruction I_i for all inputs of size n.
- It is obvious that $T \in O(f_1 + f_2 + \cdots + f_k)$.
- Let us recall that if $f_j \in O(f_i)$ then $c_i \cdot f_i + c_j \cdot f_j \in O(f_i)$.
- If there is a function f_i such that for all f_j, where $j \neq i$, we have $f_j \in O(f_i)$, then
$$T \in O(f_i).$$

Complexity of Algorithms

- Obviously, $T \in \Omega(f_i)$ for any function f_i.
- So in an analysis of a total running time $T(n)$, we can typically restrict our attention only to an analysis of how many times the most often executed instruction I_i is executed, i.e., on examination of a rate of groth of function $f_i(n)$ because

$$T \in \Theta(f_i).$$

- For other instructions I_j we just need to verify that

$$f_j \in O(f_i),$$

i.e., it is not necessary to determine precisely for them how fast they grow but rather it is sufficcient to determine for them that their rate of growth is not bigger than the rate of groth of f_i.

Complexity of Algorithms

Example:

Algorithm: 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 $A[k]$
```

Complexity of Algorithms

In the analysis of the complexity of the searching of a number in a sequence we obtained

$$f(n) = an + b\,.$$

If we would not like to do such a detailed analysis, we could deduce that the time complexity of the algorithm is $\Theta(n)$, because:

- The algorithm contains only one cycle, which is performed $(n-1)$ times for an input of size n, the number of iterations of the cycle is in $\Theta(n)$.
- Several instructions are performed in one iteration of the cycle. The number of these instructions is bounded from both above and below by some constant independent on the size of the input. So the time of execution of one iteration of the cycle is in $\Theta(1)$.
- Other instructions are executed just once. The time spent by their execution is in $\Theta(1)$.

Complexity of Algorithms

Let us try to analyze the time complexity of the following algorithm:

Algorithm: Insertion sort

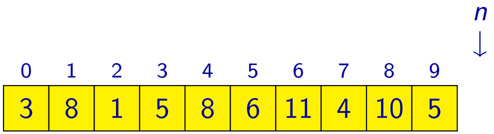
```
INSERTION-SORT ($A, n$):
    for $j := 1$ to $n - 1$ do
        $x := A[j]$
        $i := j - 1$
        while $i \geq 0$ and $A[i] > x$ do
            $A[i + 1] := A[i]$
            $i := i - 1$
        $A[i + 1] := x$
```

I.e., we want to find a function $T(n)$ such that the time complexity of the algorithm INSERTION-SORT in the worst case is in $\Theta(T(n))$.

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

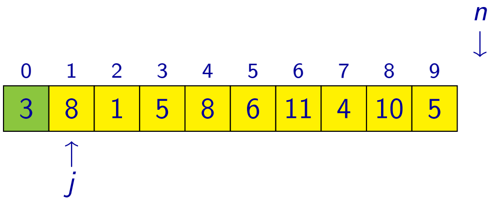


$x = ?$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

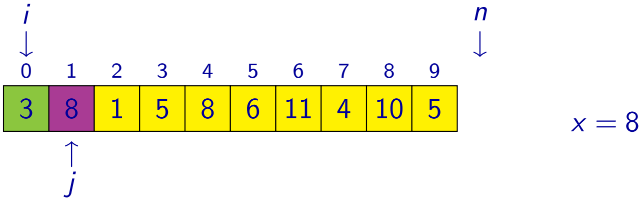


$x = ?$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

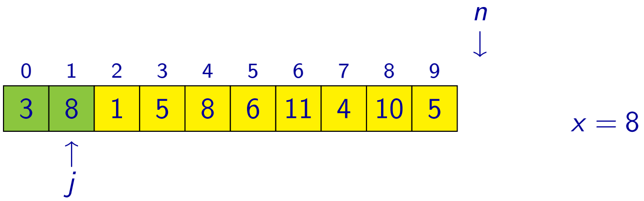
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\; n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

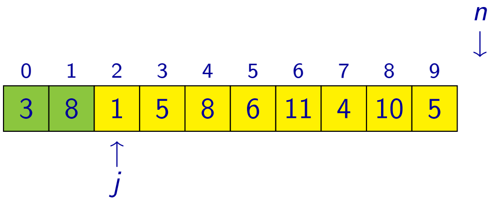
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

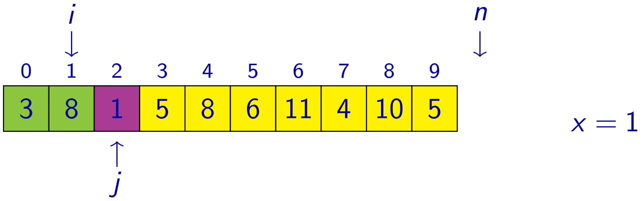


$x = 8$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

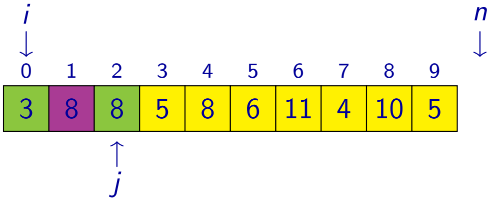
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Complexity of Algorithms

Example: A computation of Insertion-Sort on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

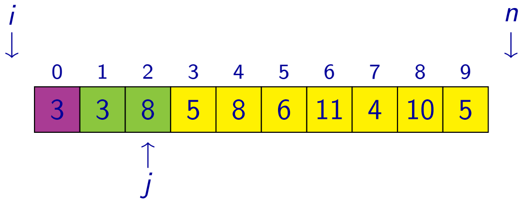


$x = 1$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

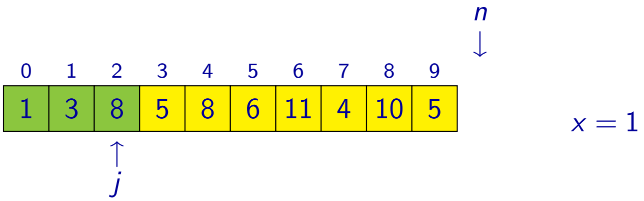


$x = 1$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

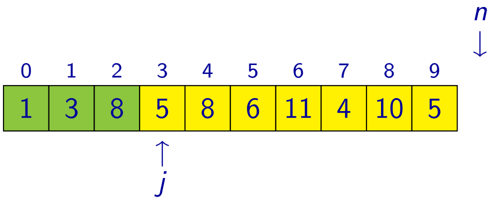
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

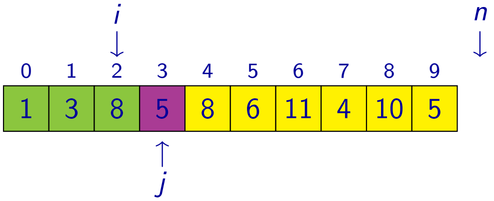


$x = 1$

Complexity of Algorithms

Example: A computation of Insertion-Sort on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\; n = 10.$$

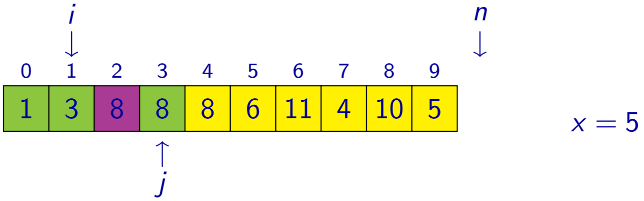


$x = 5$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

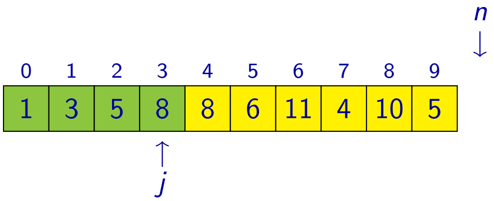
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

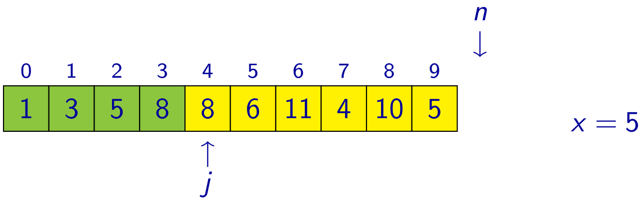


$x = 5$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

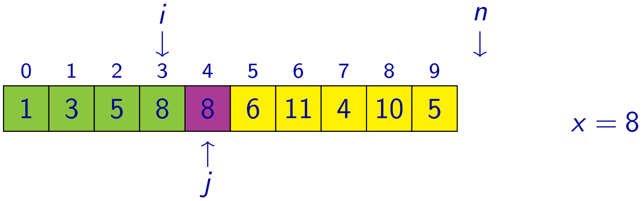
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

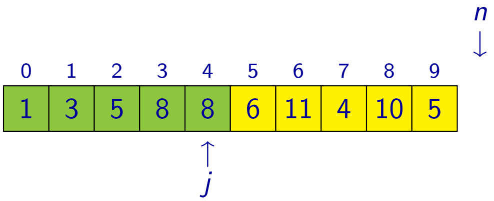
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$

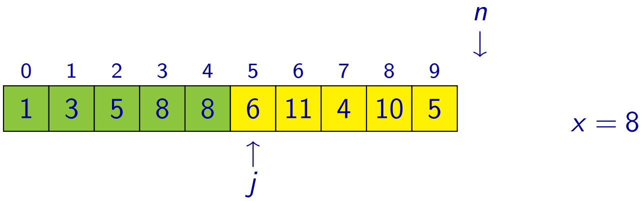


$x = 8$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

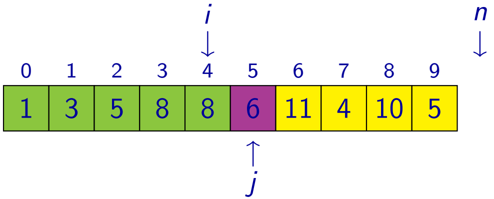
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$



Complexity of Algorithms

Example: A computation of Insertion-Sort on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

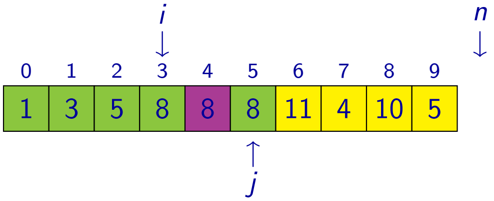


$x = 6$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

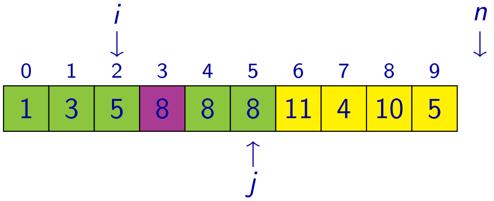


$x = 6$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

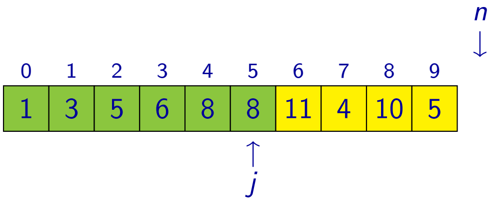


$x = 6$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$

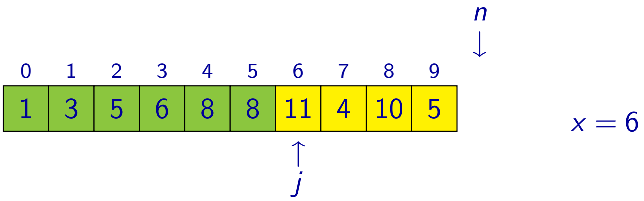


$x = 6$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

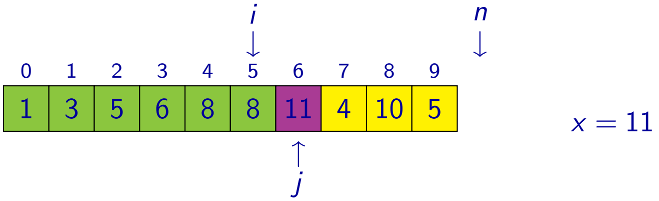
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of Insertion-Sort on input

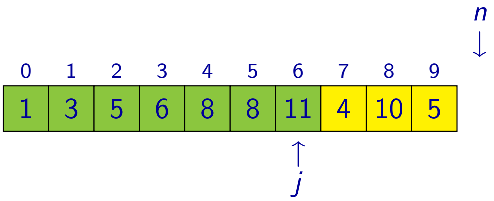
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

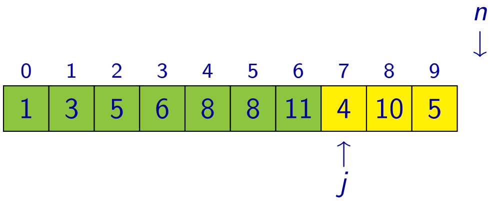


$x = 11$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

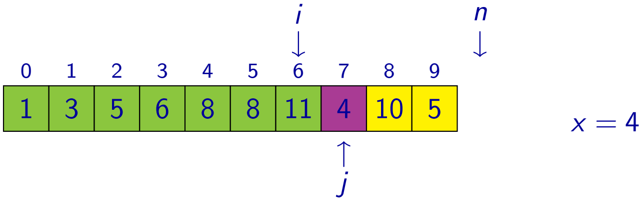


$x = 11$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

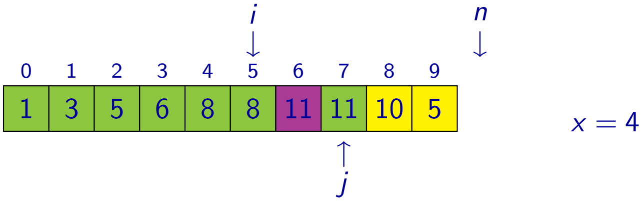
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

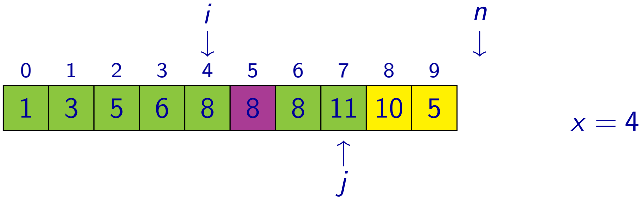
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of Insertion-Sort on input

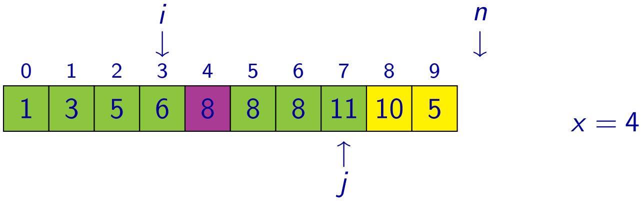
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

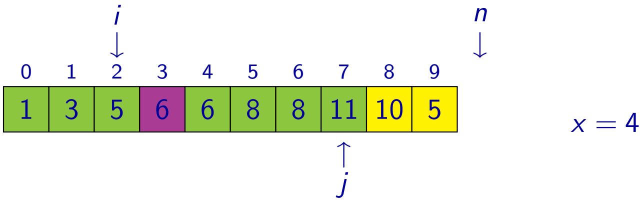
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

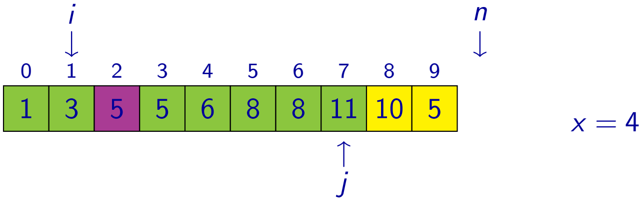
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

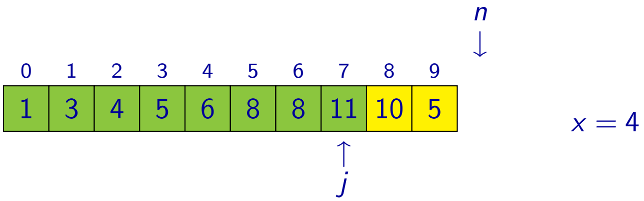
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

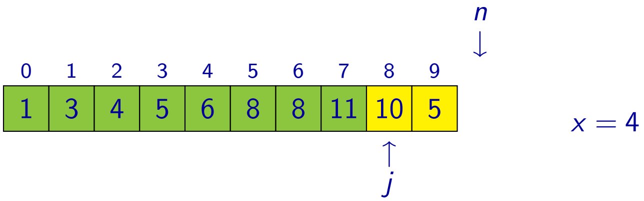
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

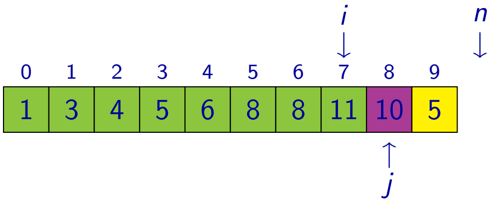
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

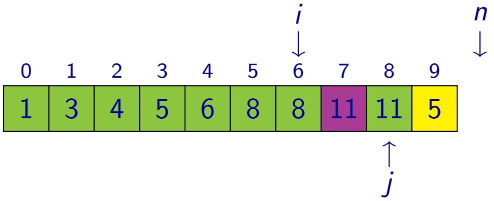


$x = 10$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

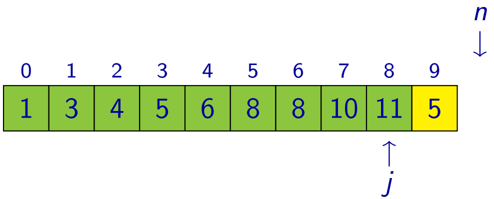


$x = 10$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

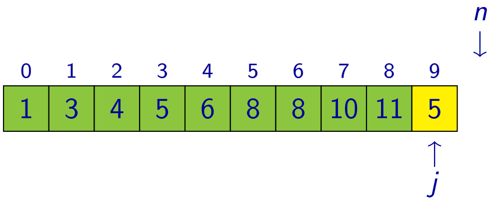


$x = 10$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$

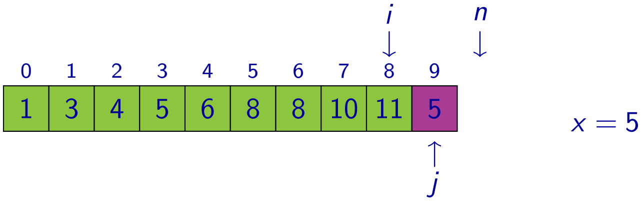


$x = 10$

Complexity of Algorithms

Example: A computation of Insertion-Sort on input

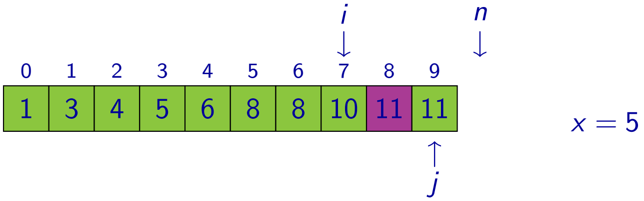
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

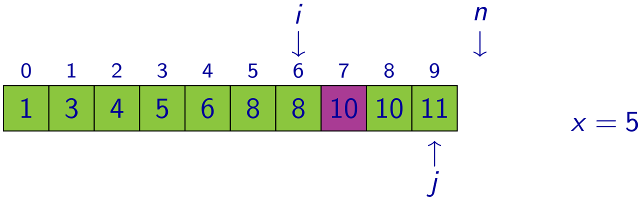
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

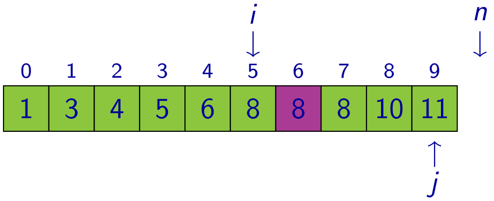
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$

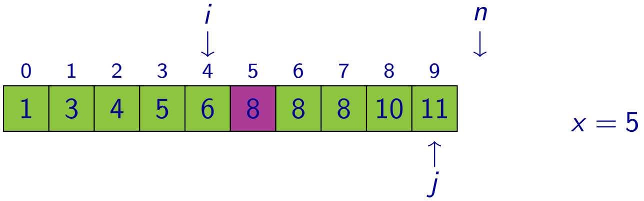


$x = 5$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

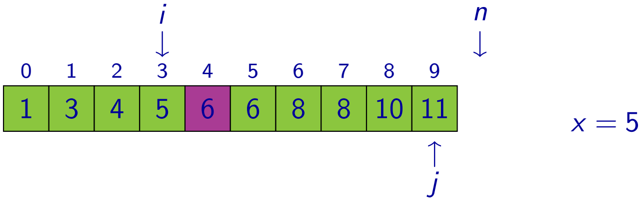
$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \; n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

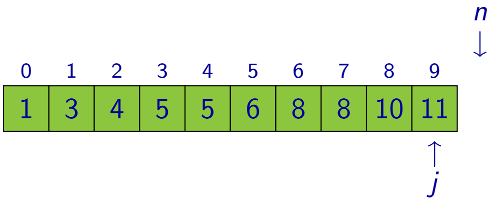
$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\ n = 10.$$



Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [3, 8, 1, 5, 8, 6, 11, 4, 10, 5], \ n = 10.$$

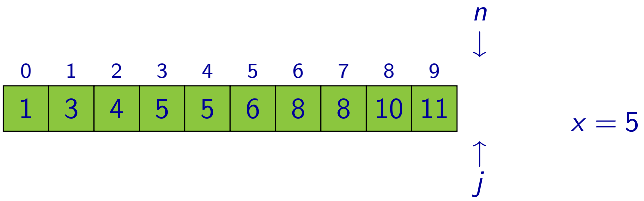


$x = 5$

Complexity of Algorithms

Example: A computation of INSERTION-SORT on input

$$A = [\,3, 8, 1, 5, 8, 6, 11, 4, 10, 5\,],\; n = 10.$$



Complexity of Algorithms

Algorithm: Insertion sort

```
INSERTION-SORT ($A, n$):
    for $j := 1$ to $n - 1$ do
        $x := A[j]$
        $i := j - 1$
        while $i \geq 0$ and $A[i] > x$ do
            $A[i + 1] := A[i]$
            $i := i - 1$
        $A[i + 1] := x$
```

Complexity of Algorithms

Let us consider inputs of size n:

- The outer cycle **for** is performed at most $n-1$ times.
  (Variable j takes values 1, 2, ..., $n-1$.)
- The inner cycle **while** is performed at most j times for a given value j.
  (Variable i takes values $j-1$, $j-2$, ..., 1, 0.)
- There are inputs such that the cycle **while** is performed exactly j times for each value j from 1 to $n-1$.
- So in the worst case, the cycle **while** is performed exactly m times, where

$$m = 1 + 2 + \cdots + (n-1) = (1 + (n-1)) \cdot \frac{n-1}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

- This means that the total running time of the algorithm $\textsc{Insertion-Sort}$ in the worst case is $\Theta(n^2)$.

Complexity of Algorithms

In the previous case, we have computed the total number of executions of the cycle **while** accurately.

This is not always possible in general, or it can be quite complicated. It is also not necessary, if we only want an asymptotic estimation.

Complexity of Algorithms

For example, if we were not able to compute the sum of the arithmetic progression, we could proceed as follows:

- The outer cycle **for** is not performed more than n times and the inner cycle **while** is performed at most n times in each iteration of the outer cycle.

  So we have $T \in O(n^2)$.

- For some inputs, the cycle **while** is performed at least $\lceil n/2 \rceil$ times in the last $\lfloor n/2 \rfloor$ iterations of the cycle **for**.

  So the cycle **while** is performed at least $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil$ times for some inputs.

  $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \geq (n/2 - 1) \cdot (n/2) = \frac{1}{4}n^2 - \frac{1}{2}n$

  This implies $T \in \Omega(n^2)$.

Complexity of Algorithms

- So far we considered that execution of a given instruction always takes the same time without regard the values, with which it works.
- So when asymptotic notation was used, the time how long it takes to execute an individual instruction played no role and it was only important how many times the given instruction is executed.
- For example, when RAMs are used as a model of computation, this corresponds to counting of instructions executed, i.e., an execution of one instruction takes 1 time unit.

  This is known as using the so called **uniform-cost measurement**.
- Estimations of the time complexity using the uniform-cost measurement correnspond to the running time on real computers under the assumption that operations, performed by the RAM, can be performed by a real computer in a constant time.

  This assumption holds, if numbers, the algorithm works with, are small (they can be stored, say, to 32 or 64 bits).

Complexity of Algorithms

- If the RAM works with "big" numbers (e.g., 1000 bit), the estimation using the uniform-cost measurement will be unrealistic in the sense that a computation on a real computer will take much more time.
- To analyse the time complexity of algorithms working with big numbers, we usually use so called **logarithmic-cost measurement**, where a duration of one instruction is not 1 but is proportional to the number of **bit operations**, which are necessary for an execution of this instruction.
- The duration of an exection of an instruction depends on the actual values of its operands.
- For example, a duration of an execution of instructions for addition and subtraction is equal to the sum of the numbers of bits of their operands.
- The duration of an execution of instructions multiplication and division is equal to the product of the numbers of bits of their operands.

Complexity of Algorithms

Remark: The notation $blen(x)$ denotes the number of bits in a binary representation of a natural number x.

It holds that

$$blen(x) = \max\left(1, \lceil \log_2(x+1) \rceil\right)$$

Space Complexity of Algorithms

- So far we have considered only the time necessary for a computation
- Sometimes the size of the memory necessary for the computation is more critical.

For RAMs and their use of memory, we can again distinguish between the use of uniform-cost and logarithmic-cost measurement:

Amount of memory of a RAM $\mathcal{M}$ used for an input w is the number of memory cells that are used by $\mathcal{M}$ during its computation on w.

Definition

A **space complexity** of a RAM $\mathcal{M}$ (in the worst case) is the function $S : \mathbb{N} \rightarrow \mathbb{N}$, where $S(n)$ is the maximal amount of memory used by $\mathcal{M}$ for inputs of size n.

Space Complexity of Algorithms

- There can be two algorithms for a particular problem such that one of them has a smaller time complexity and the other a smaller space complexity.

- If the time-complexity of an algorithm is in $O(f(n))$ then also the space complexity is in $O(f(n))$ (note that a RAM uses at most three cells in each step — at most two for reading and at most one for writing).

Examples of an Analysis of Complexity of Algorithms

Complexity of Algorithms

Some typical values of the size of an input n, for which an algorithm with the given time complexity usually computes the output on a "common PC" within a fraction of a second or at most in seconds.

(Of course, this depends on particular details. Moreover, it is assumed here that no big constants are hidden in the asymptotic notation)

$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$
1 000 000 – 100 000 000	100 000 – 1 000 000	1000 – 10 000	100 – 1000

$2^{O(n)}$	$O(n!)$
20 – 30	10 – 15

Complexity of Algorithms

When we use asymptotic estimations of the complexity of algorithms, we should be aware of some issues:

- Asymptotic estimations describe only how the running time grows with the growing size of input instance.
- They do not say anything about exact running time. Some big constants can be hidden in the asymptotic notation.
- An algorithm with better asymptotic complexity than some other algorithm can be in reality faster only for very big inputs.
- We usually analyze the time complexity in the worst case. For some algorithms, the running time in the worst case can be much higher than the running time on “typical” instances.

Complexity of Algorithms

- This can be illustrated on algorithms for sorting.

Algorithm	Worst-case	Average-case
Bubblesort	$\Theta(n^2)$	$\Theta(n^2)$
Heapsort	$\Theta(n \log n)$	$\Theta(n \log n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \log n)$

- Quicksort has a worse asymptotic complexity in the worst case than Heapsort and the same asymptotic complexity in an average case but it is usually faster in practice.

Complexity of Algorithms

Polynomial — an expression of the form

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n + a_0$$

where $a_0, a_1, \ldots, a_k$ are constants.

Examples of polynomials:

$$4n^3 - 2n^2 + 8n + 13 \qquad 2n + 1 \qquad n^{100}$$

Function f is called **polynomial** if it is bounded from above by some polynomial, i.e., if there exists a constant k such that $f \in O(n^k)$.

For example, the functions belonging to the following classes are polynomial:

$$O(n) \qquad O(n \log n) \qquad O(n^2) \qquad O(n^5) \qquad O(\sqrt{n}) \qquad O(n^{100})$$

Complexity of Algorithms

Function such as 2^n or $n!$ are not polynomial — for arbitrarily big constant k we have

$$2^n \in \Omega(n^k) \qquad\qquad n! \in \Omega(n^k)$$

Polynomial algorithm — an algorithm whose time complexity is polynomial (i.e., bounded from above by some polynomial)

Roughly we can say that:

- polynomial algorithms are effiecient algorithms that can be used in practice for inputs of considerable size
- algorithms, which are not polynomial, can be used in practice only for rather small inputs

Complexity of Algorithms

The division of algorithms on polynomial and non-polynomial is very rough — we cannot claim that polynomial algorithms are always efficient and non-polynomial algorithms are not:

- an algorithm with the time complexity $\Theta(n^{100})$ is probably not very useful in practice,
- some algorithms, which are non-polynomial, can still work very efficiently for majority of inputs, and can have a time complexity bigger than polynomial only due to some problematic inputs, on which the computation takes long time.

Remark: Polynomial algorithms where the constant in the exponent is some big number (e.g., algorithms with complexity $\Theta(n^{100})$) almost never occur in practice as solutions of usual algorithmic problems.

Complexity of Algorithms

For most of common algorithmic problems, one of the following three possibilities happens:

- A polynomial algorithm with time complexity $O(n^k)$ is known, where k is some very small number (e.g., 5 or more often 3 or less).
- No polynomial algorithm is known and the best known algorithms have complexities such as $2^{\Theta(n)}$, $\Theta(n!)$, or some even bigger.

  In some cases, a proof is known that there does not exist a polynomial algorithm for the given problem (it cannot be constructed).
- No algorithm solving the given problem is known (and it is possibly proved that there does not exist such algorithm)

Complexity of Algorithms

A typical example of polynomial algorithm — matrix multiplication with time complexity $\Theta(n^3)$ and space complexity $\Theta(n^2)$:

Algorithm: Matrix multiplication

```
MATRIX-MULT $(A, B, C, n)$:
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            $x := 0$
            for $k := 1$ to $n$ do
                $x := x + A[i][k] * B[k][j]$
            $C[i][j] := x$
```

Complexity of Algorithms

- For a rough estimation of complexity, it is often sufficient to count the number of nested loops — this number then gives the degree of the polynomial

  Example: Three nested loops in the matrix multiplication — the time complexity of the algorithm is $O(n^3)$.

- If it is not the case that all the loops go from 0 to n but the number of iterations of inner loops are different for different iterations of an outer loops, a more precise analysis can be more complicated.

  It is often the case, that the sum of some sequence (e.g., the sum of arithmetic or geometric progression) is then computed in the analysis.

  The results of such more detailed analysis often does not differ from the results of a rough analysis but in many cases the time complexity resulting from a more detailed analysis can be considerably smaller than the time complexity following from the rough analysis.

Arithmetic Progression

Arithmetic progression — a sequence of numbers $a_0, a_1, \ldots, a_{n-1}$, where

$$a_i = a_0 + i \cdot d,$$

where d is some constant independent on i.

So in an arithmetic progression, we have $a_{i+1} = a_i + d$ for each i.

Example: The arithmetic progression where $a_0 = 1$, $d = 1$, and $n = 100$:

$$1, 2, 3, 4, 5, 6, \ \ldots\ , 96, 97, 98, 99, 100$$

The sum of an arithmetic progression:

$$\sum_{i=0}^{n-1} a_i \ = \ a_0 + a_1 + \cdots + a_{n-1} \ = \ \frac{1}{2} n \left(a_0 + a_{n-1}\right)$$

Arithmetic Progression

Example:

$$1 + 2 + \cdots + n \; = \; \frac{1}{2}n(n+1) \; = \; \frac{1}{2}n^2 + \frac{1}{2}n \; = \; \Theta(n^2)$$

For example, for $n = 100$ we have

$$1 + 2 + \cdots + 100 \; = \; 50 \cdot 101 \; = \; 5050.$$

Arithmetic Progression

Proof: Let us denote

$$s = \sum_{i=0}^{n-1} a_i = a_0 + a_1 + \cdots + a_{n-1}$$

$$\begin{aligned}
2s &= s + s \\
&= (a_0 + a_1 + \cdots + a_{n-1}) + (a_0 + a_1 + \cdots + a_{n-1}) \\
&= (a_0 + a_1 + \cdots + a_{n-1}) + (a_{n-1} + a_{n-2} + \cdots + a_0) \\
&= (a_0 + a_{n-1}) + (a_1 + a_{n-2}) + \cdots + (a_{n-1} + a_0) \\
&= ((a_0 + 0{\cdot}d) + (a_0 + (n-1){\cdot}d)) + ((a_0 + 1{\cdot}d) + (a_0 + (n-2){\cdot}d)) + \\
&\quad \cdots + ((a_0 + (n-1){\cdot}d) + (a_0 + 0{\cdot}d)) \\
&= n \cdot (a_0 + a_0 + (n-1){\cdot}d) \\
&= n \cdot (a_0 + a_{n-1})
\end{aligned}$$

Aritmetická posloupnost

Example: $s = 1 + 2 + 3 + \cdots + 99 + 100$

$$\begin{aligned} 2s &= s + s \\ &= (1 + 2 + \cdots + 100) + (1 + 2 + \cdots + 100) \\ &= (1 + 2 + \cdots + 100) + (100 + 99 + \cdots + 1) \\ &= (1 + 100) + (2 + 99) + (3 + 98) + \cdots + (99 + 2) + (100 + 1) \\ &= 100 \cdot (1 + 100) = 10100 \end{aligned}$$

So

$$s = \frac{1}{2} \cdot 10100 = 5050$$

Geometric Progression

Geometric progression — a sequence of numbers $a_0, a_1, \ldots, a_n$, where

$$a_i = a_0 \cdot q^i,$$

where q is some constant independent on i.

So in a geometric progression we have $a_{i+1} = a_i \cdot q$ for each i.

Example: The geometric progression where $a_0 = 1$, $q = 2$, and $n = 14$:

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384

The sum of a geometic progression (where $q \neq 1$):

$$\sum_{i=0}^{n} a_i = a_0 + a_1 + \cdots + a_n \;=\; a_0 \frac{q^{n+1} - 1}{q - 1}$$

Geometrická posloupnost

Example:

$$1 + q + q^2 + \cdots + q^n \ = \ \frac{q^{n+1} - 1}{q - 1}$$

In particular, for $q = 2$:

$$1 + 2^1 + 2^2 + 2^3 + \cdots + 2^n \ = \ \frac{2^{n+1} - 1}{2 - 1} \ = \ 2 \cdot 2^n - 1 \ = \ \Theta(2^n)$$

Geometric Progression

Proof: Let us denote

$$s = \sum_{i=0}^{n} a_i = a_0 + a_1 + \cdots + a_n$$

$$\begin{aligned}
s &= a_0 \cdot q^0 + a_0 \cdot q^1 + \cdots + a_0 \cdot q^n \\
s \cdot q &= (a_0 \cdot q^0 + a_0 \cdot q^1 + \cdots + a_0 \cdot q^n) \cdot q \\
&= a_0 \cdot q^1 + a_0 \cdot q^2 + \cdots + a_0 \cdot q^{n+1} \\
s \cdot q - s &= a_0 \cdot q^{n+1} - a_0 \cdot q^0 \\
s \cdot (q-1) &= a_0 \cdot (q^{n+1} - 1) \\
s &= a_0 \cdot \frac{q^{n+1} - 1}{q - 1}
\end{aligned}$$

Complexity of Algorithms

An **exponential** function: a function of the form c^n, where c is a constant — e.g., function 2^n

Logarithm — the inverse function to an exponential function: for a given n,

$$\log_c n$$

is the value x such that $c^x = n$.

Complexity of Algorithms

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576

n	$\lceil \log_2 n \rceil$
0	—
1	0
2	1
3	2
4	2
5	3
6	3
7	3
8	3
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	4
17	5
18	5
19	5
20	5

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
131072	17
262144	18
524288	19
1048576	20

Complexity of Algorithms

Examples where exponential functions and logarithms can appear in an analysis of algorithms:

- Some value is repeatedly decreased to one half or is repeatedly doubled.

  For example, in the **binary search**, the size of an interval halves in every iteration of the loop.

  Let us assume that an array has size n.

  What is the minimal size of an array n, for which the algorithm performs at least k iterations?

  The answer: 2^k

  So we have $k = \log_2(n)$. The time complexity of the algorithm is then $\Theta(\log n)$.

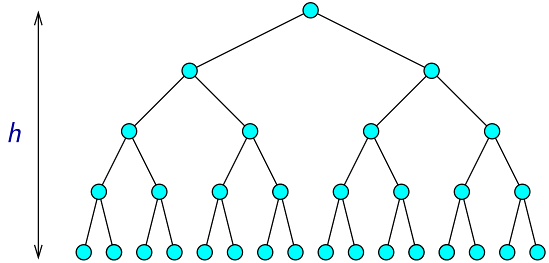
Complexity of Algorithms

- Using n bits we can represent numbers from 0 to $2^n - 1$.
- The minimal numbers of bits, which are sufficient for representing a natural number x in binary is
$$\lceil \log_2(x+1) \rceil .$$
- A perfectly balanced tree of height h has $2^{h+1} - 1$ nodes, and 2^h of these nodes are leaves.
- The height of a perfectly balanced binary tree with n nodes is $\log_2 n$.

  An illustrating example: If we would draw a balanced tree with $n = 1\,000\,000$ nodes in such a way that the distance between neighbouring nodes would be 1 cm and the height of each layer of nodes would be also 1 cm, the width of the tree would be 10 km and its height would be approximately 20 cm.

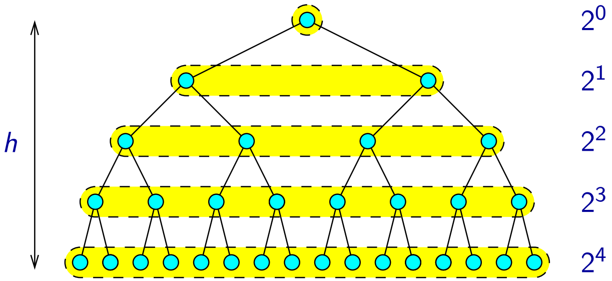
Complexity of Algorithms

A perfectly balanced binary tree of height h:



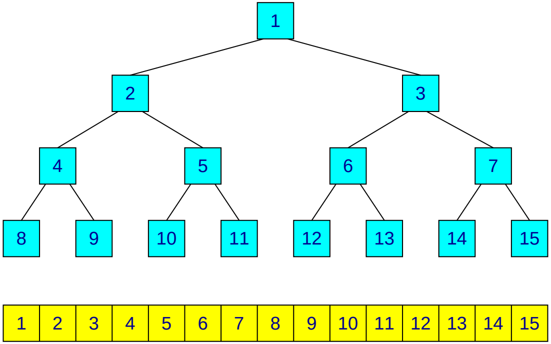
Complexity of Algorithms

A perfectly balanced binary tree of height h:



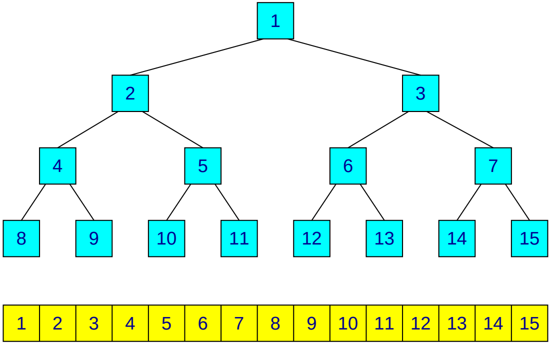
Complexity of Algorithms

An efficient way to store a complete binary tree in an array:



Complexity of Algorithms

An efficient way to store a complete binary tree in an array:



Children of a node with index i have indexes $2i$ and $2i+1$.
The parent of a node with index i has index $\lfloor i/2 \rfloor$.

Complexity of Algorithms

Heap — a complete binary tree stored in an array A in way described on the previous slide, where moreover the following invariant holds for each $i = 1, 2, \ldots, n$:

- if $2i \leq n$ then $A[i] \leq A[2i]$
- if $2i + 1 \leq n$ then $A[i] \leq A[2i + 1]$

Examples of a usage of a heap:

- sorting algorithm **HeapSort**
- an efficient implementation of a **priority queue** — this allows to perform most operations on this queue with time complexity in $O(\log n)$ where n is the number of elements currently in the queue

Complexity of Algorithms

Algorithm: Construction of a heap from an unsorted array

```
CREATE-HEAP $(A, n)$:
    $i := \lfloor n/2 \rfloor$
    while $i \geq 1$ do
        $j := i$
        $x := A[j]$
        while $2 * j \leq n$ do
            $k := 2 * j$
            if $k + 1 \leq n$ and $A[k+1] < A[k]$ then
                $k := k + 1$
            if $x \leq A[k]$ then break
            $A[j] := A[k]$
            $j := k$
        $A[j] := x$
        $i := i - 1$
```

Complexity of Algorithms

Time complexity of CREATE-HEAP:

- By a quick and rough analysis, we can easily determine that this complexity is in $O(n \log n)$ and in $\Omega(n)$:
    - The outer cycle is executed always $\lfloor n/2 \rfloor$ times — so the number of its iterations is in $\Theta(n)$.
    - The number of iterations of the inner cycle (in one iteration of the outer cycle) is obviously in $O(\log n)$.
- It is much less obvious that the total number of iterations of the inner cycle (i.e., over all iterations of the outer cycle) is in fact in $O(n)$.

So together we obtain:

The time complexity of CREATE-HEAP is in $\Theta(n)$.

Complexity of Algorithms

Justification that the total number of iterations of the inner cycle is in $O(n)$:

Let us assume for simplicity that all branches of the tree are of the same length and that their length is h — so we have $n = 2^{h+1} - 1$.

Let C_i, where $0 \leq i < h$, be the total number of iterations of the inner cycle where at the beginning of the cycle the node with index j is in i-th layer of the tree (the layers are numbered top to bottom as $0, 1, 2, \ldots$).

It is obvious that the total number of iterations s is

$$s \ = \ C_{h-1} + C_{h-2} + \cdots + C_0 \ = \sum_{i=0}^{h-1} C_i$$

The value of C_i can be computed as the total number of nodes in the layers $0, 1, \ldots, i$:

$$C_i \ = \ 2^0 + 2^1 + \cdots + 2^i \ = \sum_{k=0}^{i} 2^k \ = \ \frac{2^{i+1} - 1}{2 - 1} \ = \ 2^{i+1} - 1$$

Complexity of Algorithms

The total sum then can be computed as follows:

$$s = \sum_{i=0}^{h-1} C_i = \sum_{i=0}^{h-1} (2^{i+1} - 1) = 2 \cdot \Big(\sum_{i=0}^{h-1} 2^i \Big) - \Big(\sum_{i=0}^{h-1} 1 \Big)$$

$$= 2 \cdot \frac{2^h - 1}{2 - 1} - h = 2^{h+1} - 2 - h = n - 1 - h = O(n)$$

Undecidable Problems

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 Unsolvable Problems

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

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

Surprisingly, there are many (exactly defined) problems, for which it was proved that they are not algorithmically solvable.

Halting Problem

Let us consider some general programming language $\mathcal{L}$.

Futhermore, let us assume that programs in language $\mathcal{L}$ run on some idealized machine where a (potentially) unbounded amount of memory is available — i.e., the allocation of memory never fails.

Example: The following problem called the **Halting problem** is undecidable:

Halting problem

Input: A source code of a $\mathcal{L}$ program P, input data x.

Question: Does the computation of P on the input x halt after some finite number of steps?

Halting Problem

Let us assume that there is a program that can decide the Halting problem.

So we could construct a subroutine H, declared as

Bool H(String code, String input)

where $H(P, x)$ returns:

- true if the program P halts on the input x,
- false if the program P does not halt on the input x.

Remark: Let us say that subroutine $H(P, x)$ returns false if P is not a syntactically correct program.

Halting Problem

Using the subroutine H we can construct a program D that performs the following steps:

- It reads its input into a variable x of type String.
- It calls the subroutine $H(x, x)$.
- If subroutine H returns true, program D jumps into an infinite loop

  ```
  loop: goto loop
  ```

  In case that H returns false, program D halts.

What does the program D do if it gets its own code as an input?

Halting Problem

If D gets its own code as an input, it either halts or not.

- If D halts then $H(D, D)$ returns true and D jumps into the infinite loop. A contradiction!
- If D does not halt then $H(D, D)$ returns false and D halts. A contradiction!

In both case we obtain a contradiction and there is no other possibility. So the assumption that H solves the Halting problem must be wrong.

Semidecidable Problems

A problem is **semidecidable** if there is an algorithm such that:

- If it gets as an input an instance for which the answer is YES, then it halts after some finite number of steps and writes `"YES"` on the output.
- If it gets as an input an instance for which the answer is NO, then it either halts and writes `"NO"` on the input, or does not halt and runs forever.

It is obvious that for example HP (Halting Problem) is semidecidable.

Some problems are not even semidecidable.

Post's Theorem

The **complement** problem for a given decision problem P is a problem where inputs are the same as for the problem P and the question is negation of the question from the problem P.

Post's Theorem

If a problem P and its complement problem are semidecidable then the problem P is decidable.

Reduction between Problems

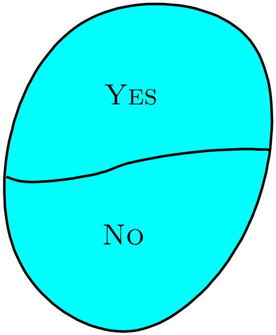
If we have already proved a (decision) problem to be undecidable, we can prove undecidability of other problems by reductions.

Problem P_1 can be **reduced** to problem P_2 if there is an algorithm Alg such that:

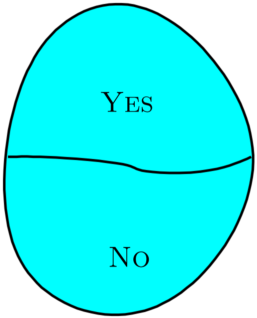
- It can get an arbitrary instance of problem P_1 as an input.
- For an instance of a problem P_1 obtained as an input (let us denote it as w) it produces an instance of a problem P_2 as an output.
- It holds i.e., the answer for the input w of problem P_1 is $\textsc{Yes}$ iff the answer for the input $Alg(w)$ of problem P_2 is $\textsc{Yes}$.

Reductions between Problems

Inputs of problem P_1

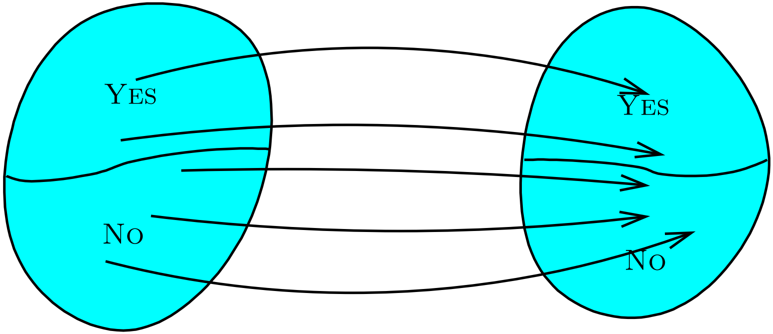


Inputs of problem P_2



Reductions between Problems

Inputs of problem P_1 Inputs of problem P_2



Alg

Reductions between Problems

Let us say there is some reduction Alg from problem P_1 to problem P_2.

If problem P_2 is decidable then problem P_1 is also decidable.

Solution of problem P_1 for an input x:

- Call Alg with x as an input, it returns a value $Alg(x)$.
- Call the algorithm solving problem P_2 with input $Alg(x)$.
- Write the returned value to the output as the result.

It is obvious that if P_1 is undecidable then P_2 cannot be decidable.

Other Undecidable Problems

By reductions from the Halting problem we can show undecidability of many other problems dealing with a behaviour of programs:

- Is for some input the output of a given program Yes?
- Does a given program halt for an arbitrary input?
- Do two given programs produce the same outputs for the same inputs?
- ...

Halting Problem

For purposes of proofs, the following version of Halting problem is often used:

Halting problem

Input: A description of a Turing machine $\mathcal{M}$ and a word w.

Question: Does the computation of the machine $\mathcal{M}$ on the word w halt after some finite number of steps?

Other Undecidable Problems

We have already seen the following example of an undecidable problem:

Problem

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

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

respectively

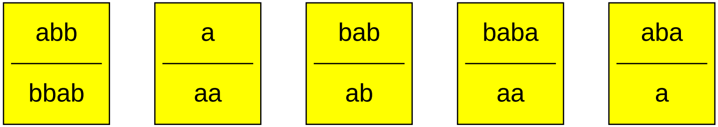
Problem

Input: A context-free grammar generating a language over an alphabet Σ.

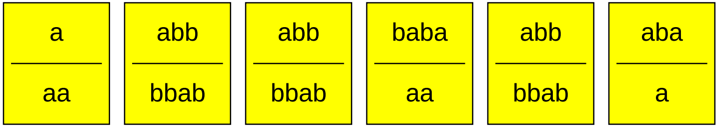
Question: Is $\mathcal{L}(\mathcal{G}) = \Sigma^*$?

Other Undecidable Problems

An input is a set of types of cards, such as:



The question is whether it is possible to construct from the given types of cards a non-empty finite sequence such that the concatenations of the words in the upper row and in the lower row are the same. Every type of a card can be used repeatedly.



In the upper and in the lower row we obtained the word `aabbabbbabaabbaba`.

Other Undecidable Problems

Undecidability of several other problems dealing with context-free grammars can be proved by reductions from the previous problem:

Problem

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

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

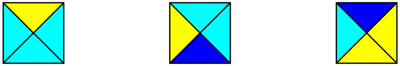
Problem

Input: A context-free grammar $\mathcal{G}$.

Question: Is $\mathcal{G}$ ambiguous?

Other Undecidable Problems

An input is a set of types of tiles, such as:

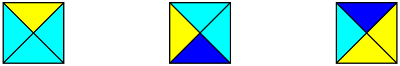


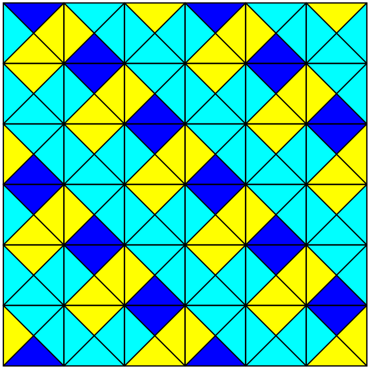
The question is whether it is possible to cover every finite area of an arbitrary size using the given types of tiles in such a way that the colors of neighboring tiles agree.

Remark: We can assume that we have an infinite number of tiles of all types.

The tiles cannot be rotated.

Other Undecidable Problems





Other Undecidable Problems

Problem

Input: A closed formula of the first order predicate logic where the only predicate symbols are $=$ and $<$, the only function symbols are $+$ and $*$, and the only constant symbols are 0 and 1.

Question: Is the given formula true in the domain of natural numbers (using the natural interpretation of all function and predicate symbols)?

An example of an input:

$$\forall x \exists y \forall z ((x * y = z) \wedge (y + 1 = x))$$

Remark: There is a close connection with Gödel's incompleteness theorem.

Other Undecidable Problems

It is interesting that an analogous problem, where real numbers are considered instead of natural numbers, is decidable (but the algorithm for it and the proof of its correctness are quite nontrivial).

Also when we consider natural numbers or integers and the same formulas as in the previous case but with the restriction that it is not allowed to use the multiplication function symbol $*$, the problem is algorithmically decidable.

Other Undecidable Problems

If the function symbol $*$ can be used then even the very restricted case is undecidable:

Hilbert's tenth problem

Input: A polynomial $f(x_1, x_2, \ldots, x_n)$ constructed from variables $x_1, x_2, \ldots, x_n$ and integer constants.

Question: Are there some natural numbers $x_1, x_2, \ldots, x_n$ such that $f(x_1, x_2, \ldots, x_n) = 0$?

An example of an input: $5x^2y - 8yz + 3z^2 - 15$

I.e., the question is whether

$$\exists x \exists y \exists z (5 * x * x * y + (-8) * y * z + 3 * z * z + (-15) = 0)$$

holds in the domain of natural numbers.

Other Undecidable Problems

Also the following problem is algorithmically undecidable:

Problem

Input: A closed formula φ of the first-order predicate logic.

Question: Is $\models \varphi$?

Remark: Notation $\models \varphi$ denotes that formula φ is logically valid, i.e., it is true in all interpretations.

Complexity Classes

Complexity of Problems

- It seems that different (algorithmic) problems are of different difficulty.
- More difficult are those problems that require more time and space to be solved.
- We would like to analyze somehow the difficultness of problems
  - absolutely – how much time and space do we need for their solution,
  - relatively – by how much is their solution harder or simpler with respect to other problems.
- Why do we not succeed in finding efficient algorithms for some problems?
  Can there exist an efficient algorithm for a given problem?
- What are practical boundaries of what can be achieved?

Complexity of Problems

It is necessary to distinguish between a **complexity of an algorithm** and a **complexity of a problem**.

If we for exaple study the time complexity in the worst case, informally we could say:

- **complexity of an algorithm** — a function expressing maximal running time of the given algorithm on inputs of size n
- **complexity of a problem** — what is the time complexity of the "most efficient" algorithm for the given problem

A formal definition of a notion "complexity of a problem" in the above sense leads to some technical difficulties. So the notion "complexity of a problem" is not defined as such but it is bypassed by a definition of **complexity classes**.

Complexity Classes

Complexity classes are subsets of the set of all (algorithmic) **problems**.

A certain particular complexity class is always characterized by a property that is shared by all the problems belonging to the class.

A typical example of such a property is a property that for the given problem there exists some algorithm with some restrictions (e.g., on its time or space complexity):

- Only a problem for which such algorithm exists belongs to the given class.
- A problem for which such algorithm does not exist does not belong to the class.

Remark: In the following discussion, we will concentrate almost exclusively on classes of **decision** problems.

Complexity Classes

Definition

For every function $f : \mathbb{N} \to \mathbb{N}$ we define $\mathcal{T}(f(n))$ as the class containing exactly those decision problems for which there exists an algorithm with time complexity $O(f(n))$.

Example:

- $\mathcal{T}(n)$ – the class of all decision problems for which there exists an algorithm with time complexity $O(n)$
- $\mathcal{T}(n^2)$ – the class of all decision problems for which there exists an algorithm with time complexity $O(n^2)$
- $\mathcal{T}(n \log n)$ – the class of all decision problems for which there exists an algorithm with time complexity $O(n \log n)$

Complexity Classes

Definition

For every function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define $\mathcal{S}(f(n))$ as the class containing exactly those decision problems for which there exists an algorithm with space complexity $O(f(n))$.

Example:

- $\mathcal{S}(n)$ – the class of all decision problems for which there exists an algorithm with space complexity $O(n)$
- $\mathcal{S}(n^2)$ – the class of all decision problems for which there exists an algorithm with space complexity $O(n^2)$
- $\mathcal{S}(n \log n)$ – the class of all decision problems for which there exists an algorithm with space complexity $O(n \log n)$

Complexity Classes

Remark:

Note that for classed $\mathcal{T}(f)$ and $\mathcal{S}(f)$ it depends which problems belong to the class on the used computational model (if it is a RAM, a one-tape Turing machine, a multitape Turing machine, . . .).

Complexity Classes

Using classes $\mathcal{T}(f(n))$ and $\mathcal{S}(f(n))$ we can define classes PTIME and PSPACE as

$$\text{PTIME} = \bigcup_{k \geq 0} \mathcal{T}(n^k) \qquad \text{PSPACE} = \bigcup_{k \geq 0} \mathcal{S}(n^k)$$

- PTIME is the class of all decision problems for which there exists an algorithm with polynomial time complexity, i.e., with time complexity $O(n^k)$ where k is a constant.
- PSPACE is the class of all decision problems for which there exists an algorithm with polynomial space complexity, i.e., with space complexity $O(n^k)$ where k is a constant.

Complexity Classes

Remark: Since all (reasonable) computational models are able to simulate each other in such a way that in this simulation the number of steps does not increase more than polynomially, the definitions of classes PTIME and PSPACE are not dependent on the used computational model.
For their definition we can use any computational model.

We say that these classes are **robust** – their definitions do not depend on the used computational model.

Complexity Classes

Other classes are introduced analogously:

EXPTIME – the set of all decision problems for which there exists an algorithm with time complexity $2^{O(n^k)}$ where k is a constant

EXPSPACE – the set of all decision problems for which there exists an algorithm with space complexity $2^{O(n^k)}$ where k is a constant

LOGSPACE – the set of all decision problems for which there exists an algorithm with space complexity $O(\log n)$

Remark: Instead of $2^{O(n^k)}$ we can also write $O(c^{n^k})$ where c and k are constants.

Complexity Classes

For definition of LOGSPACE class we specify more exacly what we consider as a space complexity of an algorithm.

For example, let us consider a Turing machine with three tapes:

- An **input tape** on which the input is written at the beginning.
- A **working tape** which is empty at the start of the computation. It is possible to read from this tape and to write on it.
- An **output tape** which is also empty at the start of the computation. It is only possible to write on it.

The amount of used space is then defined as the number of cells used on the working tape.

Complexity Classes

Other examples of complexity classes:

2-EXPTIME – the set of all problems for which there exists an algorithm with time complexity $2^{2^{O(n^k)}}$ where k is a constant

2-EXPSPACE – the set of all problems for which there exists an algorithm with space complexity $2^{2^{O(n^k)}}$ where k is a constant

ELEMENTARY – the set of all problems for which there exists an algorithm with time (or space) complexity

$$2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2^{O(n^k)}}}}}}}}$$

where k is a constant and the number of exponents is bounded by a constant.

Relationships between Complexity Classes

If a Turing machine performs m steps then it visits at most m cells on the tape.

This means that if there exists an algorithm for some problem with time complexity $O(f(n))$, the space complexity of this algorithm is (at most) $O(f(n))$.

So it is obvious that the following relationship holds.

Observation

For every function $f : \mathbb{N} \to \mathbb{N}$ is $\mathcal{T}(f(n)) \subseteq \mathcal{S}(f(n))$.

Remark: We can analogously reason in the case of a RAM.

Relationships between Complexity Classes

Based on the previous, we see that:

$$
\begin{aligned}
\text{PTIME} &\subseteq \text{PSPACE} \\
\text{EXPTIME} &\subseteq \text{EXPSPACE} \\
\text{2-EXPTIME} &\subseteq \text{2-EXPSPACE} \\
&\vdots
\end{aligned}
$$

Since polynomial functions grow more slowly than exponential and logarithmic more slowly than polynomial, we obviously have:

$$\text{PTIME} \subseteq \text{EXPTIME} \subseteq \text{2-EXPTIME} \subseteq \cdots$$

$$\text{LOGSPACE} \subseteq \text{PSPACE} \subseteq \text{EXPSPACE} \subseteq \text{2-EXPSPACE} \subseteq \cdots$$

Relationships between Complexity Classes

- For every pair of real numbers ϵ_1 a ϵ_2 taková, že $0 \leq \epsilon_1 < \epsilon_2$, is

$$\mathcal{S}(n^{\epsilon_1}) \subsetneq \mathcal{S}(n^{\epsilon_2})$$

- LOGSPACE $\subsetneq$ PSPACE
- PSPACE $\subsetneq$ EXPSPACE
- For every pair of real numbers ϵ_1 a ϵ_2 taková, že $0 \leq \epsilon_1 < \epsilon_2$, is

$$\mathcal{T}(n^{\epsilon_1}) \subsetneq \mathcal{T}(n^{\epsilon_2})$$

- PTIME $\subsetneq$ EXPTIME
- EXPTIME $\subsetneq$ 2-EXPTIME

Relationships between Complexity Classes

For analyzing relationships between complexity classes it is useful to consider **configurations**.

A configuration is a global state of a machine during one step of a computation.

- For a Turing machine, a configuration is given by the state of its control unit, the content of the tape (resp. tapes), and the position of the head (resp. heads).
- For a RAM, a configuration is given by the content of the memory, by the content of all registers (including IP), by the content of the input and output tapes, and by positions of their heads.

Relationships between Complexity Classes

It should be clear that configurations (or rather their descriptions) can be written as words over some alphabet.

Moreover, we can write configurations in such a way that the length of the corresponding words will be approximately the same as the amount of memory used by the algorithm (i.e., the number of cells on the tape used by a Turing machine, the number of number of bits of memory used by a RAM, etc.).

Remark: If we have an alphabet Σ where $|\Sigma| = c$ then:

- The number of words of length n is c^n, i.e., $2^{\Theta(n)}$.
- The number of words of length at most n is

$$\sum_{i=0}^{n} c^n = \frac{c^{n+1} - 1}{c - 1}$$

  i.e., also $2^{\Theta(n)}$.

Relationships between Complexity Classes

It is clear that during a computation of an algorithm there is no configuration repeated, since otherwise the computation would loop.

Therefore, if we know that the space complexity of an algorithm is $O(f(n))$, it means that the number of different configurations that are reachable during a computation is $2^{O(f(n))}$.

Since configurations do not repeat during a computation, also the time complexity of the algorithm is at most $2^{O(f(n))}$.

Observation

For every function $f : \mathbb{N} \to \mathbb{N}$ it holds that pokud je nějaký problém P řešený algoritmem s prostorovou složitostí $O(f(n))$, pak časová složitost tohoto algoritmu je v $2^{O(f(n))}$.

Pokud je tedy problém P ve třídě $\mathcal{S}(f(n))$, pak je i ve třídě $\mathcal{T}(2^{c \cdot f(n)})$ pro nějaké $c > 0$.

Relationships between Complexity Classes

The following results can be drawn from the previous discussion:

$$\begin{aligned} \text{LOGSPACE} &\subseteq \text{PTIME} \\ \text{PSPACE} &\subseteq \text{EXPTIME} \\ \text{EXPSPACE} &\subseteq \text{2-EXPTIME} \\ &\vdots \end{aligned}$$

Relationships between Complexity Classes

Summary:

$$\begin{aligned}\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE} \subseteq \\ \subseteq \text{2-EXPTIME} \subseteq \text{2-EXPSPACE} \subseteq \cdots \subseteq \text{ELEMENTARY}\end{aligned}$$

- $\text{PTIME} \subsetneq \text{EXPTIME} \subsetneq \text{2-EXPTIME} \subsetneq \cdots$
- $\text{LOGSPACE} \subsetneq \text{PSPACE} \subsetneq \text{EXPSPACE} \subsetneq \text{2-EXPSPACE}, \subsetneq \cdots$

Upper and Lower Bounds on Complexity of Problems

An **upper bound** on a complexity of a problem means that the complexity of the problem is not greater than some specified complexity.

Usually it is formulated so that the problem belongs to a particular complexity class.

Examples of propositions dealing with upper bounds on the complexity:

- The problem of reachability in a graph is in PTIME.
- The problem of equivalence of two regular expressions is in EXPSPACE.

If we want to find some upper bound on the complexity of a problem it is sufficient to show that there is an algorithm with a given complexity.

Upper and Lower Bounds on Complexity of Problems

A **lower bound** on a complexity of a problem means that the complexity of the problem is at least as big as some specified complexity.

In general, proving of (nontrivial) lower bounds is more difficult than proving of upper bounds.

To derive a lower bound we must prove that **every** algorithm solving the given problem has the given complexity.

Upper and Lower Bounds on Complexity of Problems

Problem "Sorting"

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

Output: Elements $a_1, a_2, \ldots, a_n$ sorted from the smallest to the greatest.

It can be proven that every algorithm, that solves the problem "Sorting" and that has the property that the only operation applied on elements of a sorted sequence is a comparison (i.e., it does not examine the content of these elements), has the time complexity in the worst case $\Omega(n \log n)$ (i.e., for every such algorithm there exist constants $c > 0$ and $n \geq n_0$ such that for every $n \geq n_0$ there is an input of size n, for which the algorithm performs at least $cn \log n$ operations.)

Nodeterministic Algorithms and Complexity Classes

Nondeterminism

Nondeterministic RAM:

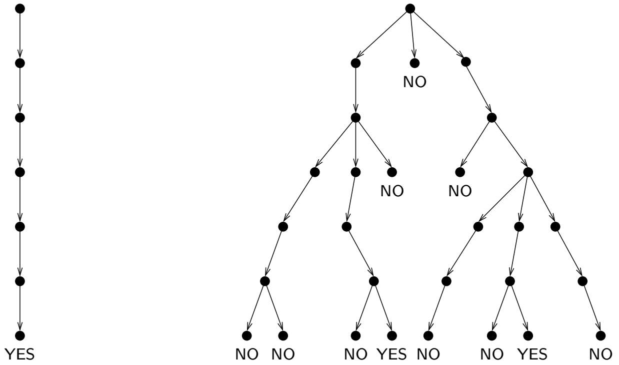
- Its definition is very similar to that of a deterministic RAM.
- Moreover, it has an instruction

  $$\textbf{nd\_goto } \ell_1, \ell_2$$

  that allows it to choose the next instruction from two possibilities.
- If at least one of computations of such a machine on a given input ends with the answer $\textsc{Yes}$, then the answer is $\textsc{Yes}$.
- If all computations end with the answer $\textsc{No}$ then the answer is $\textsc{No}$.

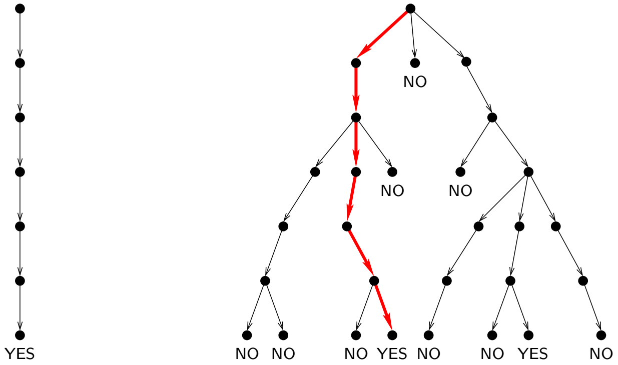
Nondeterministic versions of other computational models (such as nondeterministic Turing machines) are defined similarly.

Nondeterminism



- The time required for a computation of a nondeterministic RAM (or other nondeterministic machine) on a given input is defined as the length of the longest computation on the input.

Nondeterminism



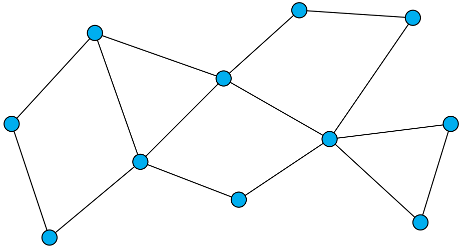
- The time required for a computation of a nondeterministic RAM (or other nondeterministic machine) on a given input is defined as the length of the longest computation on the input.

Nondeterminism

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?



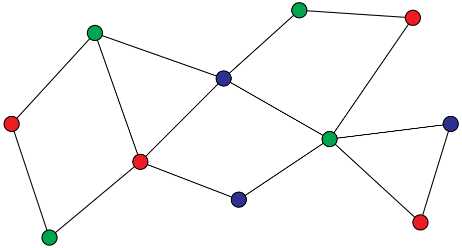
$k = 3$

Nondeterminism

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?



$k = 3$

Nondeterminism

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?

A nondeterministic algorithm works as follows:

1. It assignes nondeterministically to every node of G one of k colors.
2. It goes through all edges of G and for each of them verifies that its endpoints are colored with different colors. If this is not the case, it halts with the answer No.
3. If it has verified for all edges that their endpoints are colored with different colors, it halts with the answer Yes.

Nondeterminism

Problem "Graph isomorphism"

Input: Undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

Question: Are graphs G_1 and G_2 isomorphic?

Remark: Graphs G_1 and G_2 are isomorphic if there exists some bijection $f : V_1 \to V_2$ such that for every pair of nodes $u, v \in V_1$ is $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$.

A nondeterministic algorithm works as follows:

1. It nondeterministically chooses values of the function f for every $v \in V_1$.
2. It (deterministically) verifies that f is a bijection and that the above mentioned condition is satisfied for all pairs of nodes.
3. If some of the conditions is violated, it halts with the answer $\textsc{No}$. Otherwise it halts with the answer $\textsc{Yes}$.

Nondeterminism

- For decidability of problems, the nondeterministic algorithms are not more powerful than deterministic ones:
  If a problem can be solved by a nondeterministic RAM or TM, it can be also solved by a deterministic RAM or TM that successively tries all possible computations of the nondeterministic machine on a given input.
- Nondeterminism is useful primarily in the study of a complexity of problems.

Nondeterminism

- In the straightforward simulation of a nondeterministic algorithm by a deterministic, described above, where the deterministic algorithm systematically tries all possible computations, the time complexity of the deterministic algorithm is exponentially bigger than in the nondeterministic algorithm.
- For many problems, it is clear that there exists a nondeterministic algorithm with a polynomial time complexity solving the given problem but it is not clear at all whether there also exists a deterministic algorithm solving the same problem with a polynomial time complexity.

Nondeterminism

Nondeterminism can be viewed in two different ways:

1. When a machine should nondeterministically choose between several possibilities, it "guesses" which of these possibilities will lead to the answer YES (if there is such a possibility).

2. When a machine should choose between several possibilities, it splits itself into several copies, each corresponding to one of the possibilities. These copies continue in the computation in parallel.

   The answer is YES iff at least one of these copies halts with the answer YES.

None of these possibilities is something that could be efficiently realistically implemented.

Nondeterminism

Other possible view of the nondeterminism:

- A kind of an algorithm that does not solve the given problem but using an additional information — called **witness** — can **verify** that the answer for the given instance is $\textsc{Yes}$.

  Let us assume that in the original problem the input is some x from the set of instances In and the question is whether this x has some specified property P.

  For the given input x, there is a corresponding set $W(x)$ of **potential witnesses** with the property that x has the property P iff there exists an **actual witness** $y \in W(w)$ of the fact that x really has property P.

  There is a **deterministic** algorithm Alg that expects as input a pair (x, y) (where $y \in W(x)$) and that checks that y is a witness of the fact that x has property P.

Nondeterminism

Example: The problem "Graph Colouring with k colours":

- *Input*: An undirected graph $G = (V, E)$ and number k.
- *Potential witnesses*: All possible colourings of nodes of graph G with k colours, i.e., all functions c of the form $c : V \rightarrow \{1, \dots, k\}$.
- *Actual witnesses*: Those colourings c where for each edge $(u, v) \in E$ holds that $c(u) \neq c(v)$.

Nondeterminism

- For each **deterministic** algorithm Alg that can verify for a given pair (x, y) that y is a witness of the fact that x has property P, we can easily construct a corresponding **nondeterministic** algorithm that solves the original problem:
    - For a given $x \in In$ it generates nondeterministically a potential witness $y \in W(x)$.
    - Then it uses the (deterministic) algorithm Alg as a subroutine to check that y is an actual witness.

Nondeterminism

- On the contrary, for every **nondeterministic** algorithm, we can also easily construct a **deterministic** algorithm for checking witnesses:
  - A potential witness will be a sequence specifying for each nondeterministic step of the original algorithm, which possibility should be chosen in the given step.
  - The deterministic algorithm then simulates one particular computation (one branch of the tree) of the original algorithm where in those steps where several choices are possible, it does not guess but continues according to the sequence given as a witness.

Nondeterminism

We will concentrate particularly to those cases where the time complexity of the algorithm for checking a witness is polynomial with respect to the size of input x.

This also means that a given witness y, witnessing that the answer for x is $\textsc{Yes}$, must be of a polynomial size.

So by a nondeterministic algorithm with a polynomial time complexity we can solve those decision problems where:

- for a given input x there exists a corresponding (polynomially big) witness iff the answer for x is $\textsc{Yes}$,
- it is possible to check using a deterministic algorithm in polynomial time that a given potential witness is really a witness.

Nondeterminism

In many cases, the existence of such polynomially big witnesses and deterministic algorithms checking them is obvious and it is trivial to show that they exist — e.g., for problems like “Graph Colouring with k Colours”, “Graph Isomorphism”, or the following problem:

Testing that a number is composite

Input: A natural number x.

Question: Is the number x composite?

Remark: Number x is **composite** if there exist natural numbers a and b such that $a > 1$, $b > 1$, and $x = a \cdot b$.

For example, number 15 is composite because $15 = 3 \cdot 5$.

So the number $x \in \mathbb{N}$ is composite iff $x > 1$ and x is not a prime.

Existence of such polynomially big witnesses of course does not automatically mean that it is easy to find them.

Nondeterminism

For some problems, a proof of existence of such polynomially bounded witnesses, which can be checked deterministically in a polynomial time, rather nontrivial result.

An example can be the following problem:

Primality Testing

Input: A natural number x.

Question: Is number x a prime?

Using some nontrivial results from number theory, there can be shown existence of such witnesses even for this problem — those witnesses here are rather complicated recursively defined data structures.

Remark: This result was shown by V. Pratt in 1975.

Much later it was shown that "Primality Testing" is in PTIME (Agrawal–Kayal–Saxena, 2002).

Nondeterministic Complexity Classes

Definition

For a function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define the **time complexity class** $\mathcal{NT}(f)$ as the set of all problems that are solved by nondeterministic RAMs with a time complexity in $O(f(n))$.

Definition

For a function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define the **space complexity class** $\mathcal{NS}(f)$ as the set of all problems that are solved by nondeterministic RAMs with a space complexity in $O(f(n))$.

Remark: Of course, the definitions given above can also use Turing machines or some other model of computation instead of RAMs.

Class NPTIME

Definition

$$\mathsf{NPTIME} = \bigcup_{k=0}^{\infty} \mathcal{NT}(n^k)$$

- NPTIME (sometimes we write just NP) is the class of all problems, for which there exists a nondeterministic algorithm with polynomial time complexity.
- The class NPTIME contains those problems for which it is possible to verify in polynomial time that the answer is Yes if somebody, who wants to convince us that this is really the case, provides additional information.

Classes NPSPACE, NEXPTIME, NEXPSPACE, ...

Other classes can be defined similarly:

NPSPACE – množina všech rozhodovacích problémů, pro které existuje nedeterministický algoritmus s polynomiání prostorovou složitostí

NEXPTIME – the set of all decision problems for which there exists an algorithm with time complexity $2^{O(n^k)}$ where k is a constant

NEXPSPACE – the set of all decision problems for which there exists an algorithm with space complexity $2^{O(n^k)}$ where k is a constant

NLOGSPACE – the set of all decision problems for which there exists an algorithm with space complexity $O(\log n)$

Relationships between Complexity Classes

It is clear that deterministic algorithms can be viewed as a special case of nondeterministic algorithms.

Therefore it obviously holds that:

$$\begin{aligned} \mathsf{LOGSPACE} &\subseteq \mathsf{NLOGSPACE} \\ \mathsf{PTIME} &\subseteq \mathsf{NPTIME} \\ \mathsf{PSPACE} &\subseteq \mathsf{NPSPACE} \\ \mathsf{EXPTIME} &\subseteq \mathsf{NEXPTIME} \\ \mathsf{EXPSPACE} &\subseteq \mathsf{NEXPSPACE} \\ &\vdots \end{aligned}$$

Relationships between Complexity Classes

It is also obvious that for both deterministic and nondeterministic algorithms, an algorithm can not use considerably bigger number of memory cells than what is the number of steps executed by the algorithm.

A space complexity of an algorithm is therefore always at most as big as its time complexity.

From this follows that:

$$\begin{aligned} \mathrm{PTIME} &\subseteq \mathrm{PSPACE} \\ \mathrm{NPTIME} &\subseteq \mathrm{NPSPACE} \\ \mathrm{EXPTIME} &\subseteq \mathrm{EXPSPACE} \\ \mathrm{NEXPTIME} &\subseteq \mathrm{NEXPSPACE} \\ &\vdots \end{aligned}$$

Relationships between Complexity Classes

Consider a **nondeterministic** algorithm with **time** complexity $O(f(n))$.

A **deterministic** algorithm that will simulate its behaviour by systematically trying all its possible computations (by going through the tree of these computations in a depth-first manner) will need only the following memory:

- a memory to store a current configuration of the simulated machine — its size is $O(f(n))$ (since if this simulated machine performs at most $O(f(n))$ steps then its configurations will use at most $O(f(n))$ memory cells)

- a memory to store a stack that will be used to allow returning to previous configurations
  — to allow to go back to a previous configuration α from a following configuration α', it is sufficient to store a constant amount of information — only those things that were changed in the transition from α to α'

Relationships between Complexity Classes

- Since the length of branches is $O(f(n))$, the amount of memory needed for the stack is $O(f(n))$.
- So in total, the deterministic algorithm uses in this simulation an amount of memory, which is at most $O(f(n))$.

It follows from this that:

$$\mathrm{NPTIME} \subseteq \mathrm{PSPACE}$$
$$\mathrm{NEXPTIME} \subseteq \mathrm{EXPSPACE}$$
$$\vdots$$

Relationships between Complexity Classes

Consider a **nondeterministic** algorithm with a **space** complexity $O(f(n))$:

- Let us recall that the total number of configurations of size at most $O(f(n))$ is $O(c^{f(n)})$, where c is a constant, so this can be written as $2^{O(f(n))}$.
- So the number of steps of the nondeterministic algorithm in one branch of computation could be at most $2^{O(f(n))}$.

  (Remark: No configuration can be repeated during a computation since otherwise computations could be infinite.)
- So the simulation done this way would have time complexity $2^{2^{O(f(n))}}$.

Relationships between Complexity Classes

In a simulation we can proceed in a more clever way — consider a directed graph where:

- **nodes** — all configurations of the simulated machine whose size is at most $O(f(n))$
  — the number of such configurations is $2^{O(f(n))}$
- **edges** — there is an edge between nodes representing configurations α and α' iff the simulated machine can go in one step from configuration α to configuration α'
  — the number of edges going out from each node is bounded from above by some constant — so the number of edges is also $2^{O(f(n))}$

It is sufficient to be able to find out whether there is a path in this graph from the node corresponding to the initial configuration (for the given input x) to some node corresponding to a final configuration where the machine gives answer $\textsc{Yes}$.

Relationships between Complexity Classes

Existence of such a path can be tested using an arbitrary algorithm for searching a graph — breadth-first search, depth-first search, . . . :

- This algorithm needs to store and mark, which configurations have been already visited.
  It also needs a memory to store a queue or a stack, etc.
- The time and space complexity of such algorithm is linear with respect to the size of the graph, i.e., $2^{O(f(n))}$.

Relationships between Complexity Classes

So we obtain the following:

The behaviour of a nondeterministic algorithm whose space complexity is $O(f(n))$can be simulated by a deterministic algorithm with time complexity $2^{O(f(n))}$.

It follows from this that:

$$\begin{aligned} \mathrm{NLOGSPACE} &\subseteq \mathrm{PTIME} \\ \mathrm{NPSPACE} &\subseteq \mathrm{EXPTIME} \\ \mathrm{NEXPSPACE} &\subseteq \text{2-EXPTIME} \\ &\vdots \end{aligned}$$

Relationships between Complexity Classes

Consider once again a **nondeterministic** algorithm with **space** complexity $O(f(n))$. Now we would like to have the **space** complexity of the simulating deterministic algorithm as small as possible.

Theorem (Savitch, 1970)

The behaviour of a nondeterministic algorithm with space complexity $O(f(n))$ can be simulated by a deterministic algorithm with space complexity $O(f(n)^2)$.

Proof idea:

- Consider once again the graph of configurations with $2^{O(f(n))}$ nodes (and edges).
- The algorithm will try to find out whether there exists a path from the initial configuration to some accepting configuration.

Relationships between Complexity Classes

The most important part is a recursive function $F(\alpha, \alpha', i)$ that for arbitrary configurations α and α' and number $i \in \mathbb{N}$ finds out whether the given graph contains a path from α to α' of length at most 2^i:

- For $i = 0$ it finds out whether there is a path from α to α' of length at most 1:
    - it is either a path of length 0, i.e., $\alpha = \alpha'$,
    - or it is a path of length 1, i.e., it is possible to go from α to α' in one step
- For $i > 0$, it will systematically try all configurations α'' and check whether:
    - there is a path of length at most $2^i/2$ from α to α'' — it calls $F(\alpha, \alpha'', i-1)$ recursively
    - there is a path of length at most $2^i/2$ from α'' to α' — it calls $F(\alpha'', \alpha', i-1)$ recursively

  If both returns $\textsc{True}$, it returns $\textsc{True}$, otherwise it continues with trying the next α''.

Relationships between Complexity Classes

The analysis of the space complexity of the algorithm:

- in one recursive call of the function F, the algorithm needs to store:
  - three configurations α, α', α'' — all of them of size $O(f(n))$
  - the value of the number i, which is approximately $O(f(n))$ — so to store this number, $O(\log F(n))$ bits are sufficient
  - other auxiliary variables whose sizes are negligible compared to the sizes of the values described above

- So the amount of memory needed for one recursive call is $O(f(n))$.

- The depth of the recursion is also $O(f(n))$.

- So the total space complexity of the algorithm is $O(f(n)^2)$.

Relationships between Complexity Classes

It follows from this theorem that:

$$\mathrm{NPSPACE} \subseteq \mathrm{PSPACE}$$
$$\mathrm{NEXPSPACE} \subseteq \mathrm{EXPSPACE}$$
$$\vdots$$

Together with the trivial facts that $\mathrm{PSPACE} \subseteq \mathrm{NPSPACE}$, $\mathrm{EXPSPACE} \subseteq \mathrm{NEXPSPACE}$, ... this implies:

$$\mathrm{PSPACE} = \mathrm{NPSPACE}$$
$$\mathrm{EXPSPACE} = \mathrm{NEXPSPACE}$$
$$\vdots$$

Remark: Note that it **does not follow** from this that $\mathrm{LOGSPACE} = \mathrm{NLOGSPACE}$.

Relationships between Complexity Classes

Putting all this together, we obtain the following **hierarchy of complexity classes**:

$$\begin{gathered}\mathsf{LOGSPACE} \subseteq \mathsf{NLOGSPACE} \subseteq \\ \subseteq \mathsf{PTIME} \subseteq \mathsf{NPTIME} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \\ \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME} \subseteq \mathsf{EXPSPACE} = \mathsf{NEXPSPACE} \subseteq \\ \vdots\end{gathered}$$

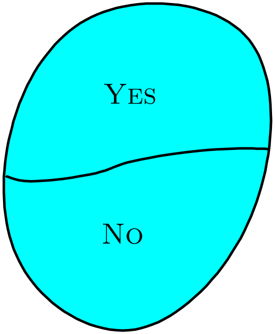
NP-Complete Problems

Polynomial Reductions between Problems

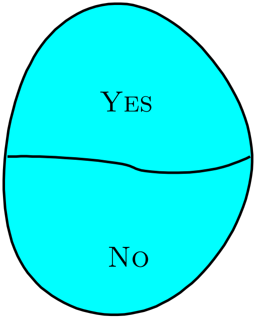
There is a **polynomial reduction** of problem P_1 to problem P_2 if there exists an algorithm Alg with a polynomial time complexity that reduces problem P_1 to problem P_2.

Polynomial Reductions between Problems

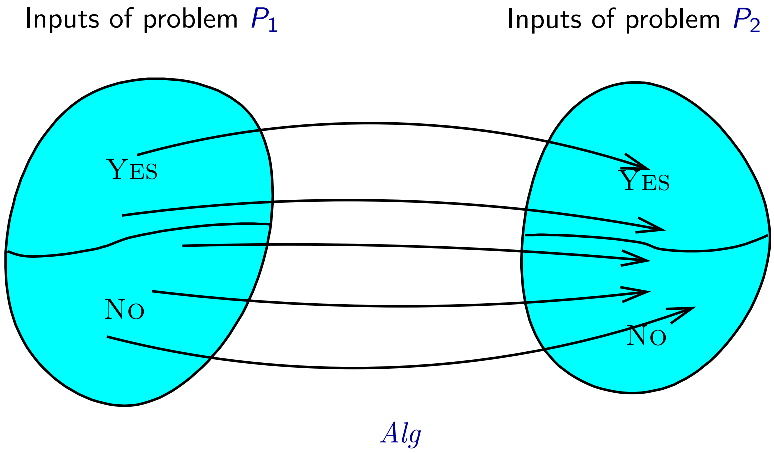
Inputs of problem P_1



Inputs of problem P_2



Polynomial Reductions between Problems



Polynomial Reductions between Problems

Let us say that problem A can be reduced in polynomial time to problem B, i.e., there is a polynomial algorithm P realizing this reduction.

If problem B is in the class PTIME then problem A is also in the class PTIME.

A solution of problem A for an input x:

- Call P with input x and obtain a returned value $P(x)$.
- Call a polynomial time algorithm solving problem B with the input $P(x)$.
  Write the returned value as the answer for A.

That means:

If A is not in PTIME then also B can not be in PTIME.

Polynomial Reductions between Problems

There is a big class of algorithmic problems called **NP-complete** problems such that:

- patří do třídy NPTIME, tj. jsou řešitelné v polynomiálním čase **nedeterministickým** algoritmem
- these problems can be solved by exponential time algorithms
- no polynomial time algorithm is known for any of these problems
- on the other hand, for any of these problems it is not proved that there cannot exist a polynomial time algorithm for the given problem
- every NP-complete problem can be polynomially reduced to any other NP-complete problem

Remark: This is not a definition of NP-complete problems. The precise definition will be described later.

Problem SAT

A typical example of an NP-complete problem is the SAT problem:

SAT (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 v where $v(x_1) = 1$, $v(x_2) = 0$, $v(x_3) = 1$, the formula φ_1 is true.

Formula $\varphi_2 = (x_1 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge x_2)$ is not satisfiable:
it is false for every valuation v.

Problem 3-SAT

3-SAT is a variant of the SAT problem where the possible inputs are restricted to formulas of a certain special form:

3-SAT

Input: Formula φ is a conjunctive normal form where every clause contains exactly 3 literals.

Question: Is φ satisfiable?

Problem 3-SAT

Recalling some notions:

- A **literal** is a formula of the form x or $\neg x$ where x is an atomic proposition.
- A **clause** is a disjuction of literals.

  Examples: $x_1 \vee \neg x_2$ $\qquad \neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}$ $\qquad x_6$
- A formula is in a **conjuctive normal form (CNF)** if it is a conjuction of clauses.

  Example: $(x_1 \vee \neg x_2) \ \wedge\ (\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}) \ \wedge\ x_6$

So in the 3-SAT problem we require that a formula φ is in a CNF and moreover that every clause of φ contains exactly three literals.

Example:

$$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

Problem 3-SAT

The following formula is satisfiable:

$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$

It is true for example for valuation v where

$$\begin{aligned} v(x_1) &= 0 \\ v(x_2) &= 1 \\ v(x_3) &= 0 \\ v(x_4) &= 1 \end{aligned}$$

On the other hand, the following formula is not satisfiable:

$$(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$$

Polynomial Reductions between Problems

As an example, a polynomial time reduction from the 3-SAT problem to the independent set problem (IS) will be described.

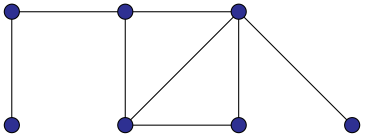
Remark: Both 3-SAT and IS are examples of NP-complete problems.

Independent Set (IS) Problem

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?



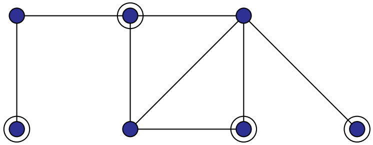
$k = 4$

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

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?

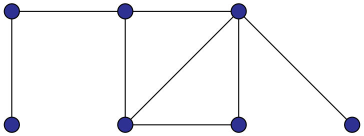


$k = 4$

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.

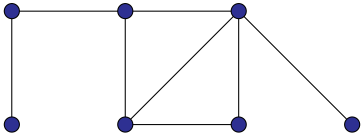
Independent Set (IS) Problem

An example of an instance where the answer is YES:



$k = 4$

An example of an instance where the answer is NO:



$k = 5$

A Reduction from 3-SAT to IS

We describe a (polynomial-time) algorithm with the following properties:

- **Input:** An arbitrary instance of 3-SAT, i.e., a formula φ in a conjunctive normal form where every clause contains exactly three literals.
- **Output:** An instance of IS, i.e., an undirected graph G and a number k.
- Moreover, the following will be ensured for an arbitrary input (i.e., for an arbitrary formula φ in the above mentioned form):

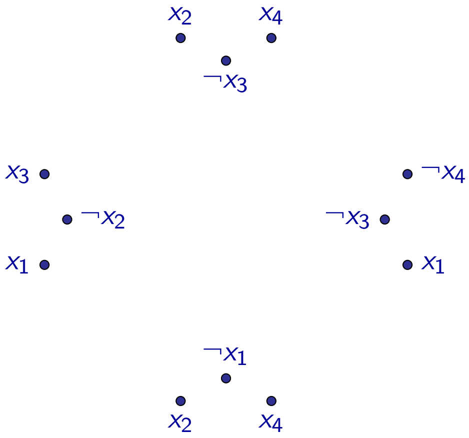
  There will be an independent set of size k in graph G iff formula φ will be satisfiable.

A Reduction from 3-SAT to IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

A Reduction from 3-SAT to IS

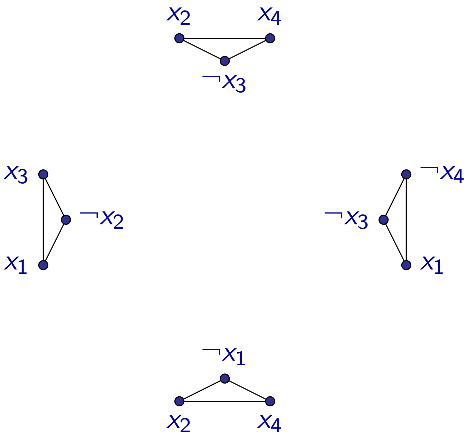
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



For each occurrence of a literal we add a node to the graph.

A Reduction from 3-SAT to IS

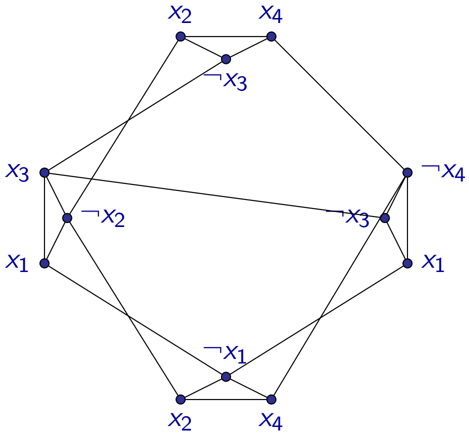
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



We connect with edges the nodes corresponding to occurrences of literals belonging to the same clause.

A Reduction from 3-SAT to IS

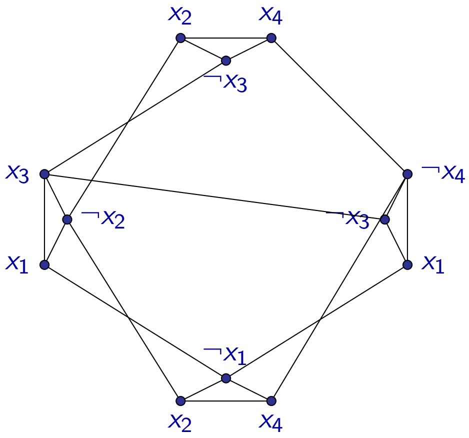
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



For each pair of nodes corresponding to literals x_i and $\neg x_i$ we add an edge between them.

A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

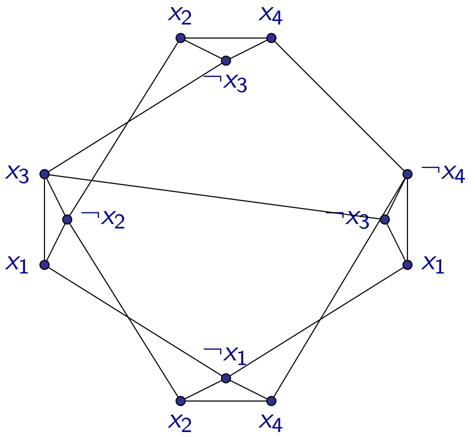


$k = 4$

We put k to be equal to the number of clauses.

A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



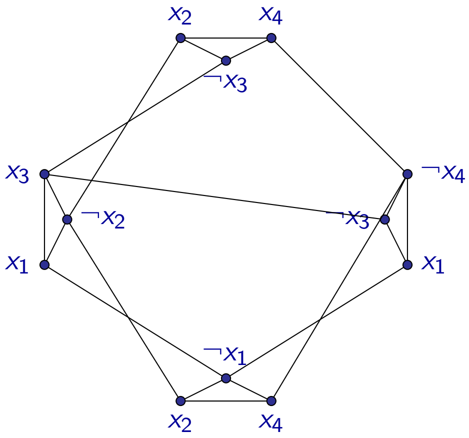
$k = 4$

The constructed graph and number k are the output of the algorithm.

A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

$v(x_1) = 1$
$v(x_2) = 1$
$v(x_3) = 0$
$v(x_4) = 1$



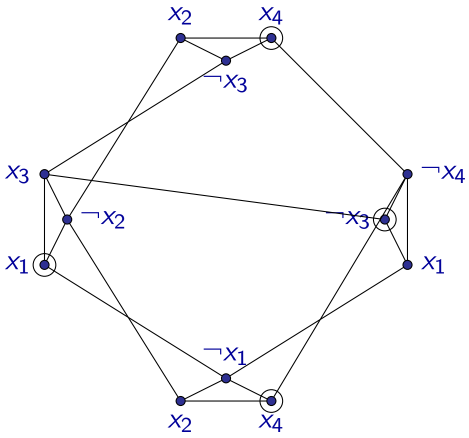
$k = 4$

If the formula φ is satisfiable then there is a valuation v where every clause contains at least one literal with value 1.

A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

$v(x_1) = 1$
$v(x_2) = 1$
$v(x_3) = 0$
$v(x_4) = 1$



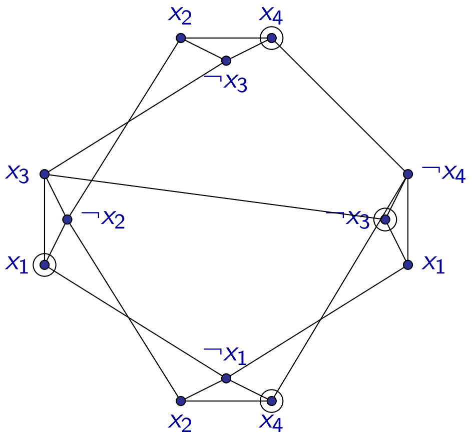
$k = 4$

We select one literal that has a value 1 in the valuation v, and we put the corresponding node into the independent set.

A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

$v(x_1) = 1$
$v(x_2) = 1$
$v(x_3) = 0$
$v(x_4) = 1$



$k = 4$

We can easily verify that the selected nodes form an independent set.

A Reduction from 3-SAT to IS

The selected nodes form an independent set because:

- One node has been selected from each triple of nodes corresponding to one clause.
- Nodes denoted x_i and $\neg x_i$ could not be selected together.
  (Exactly of them has the value 1 in the given valuation v.)

A Reduction from 3-SAT to IS

On the other hand, if there is an independent set of size k in graph G, then it surely has the following properties:

- At most one node is selected from each triple of nodes corresponding to one clause.

  But because there are k clauses and k nodes are selected, exactly one node must be selected from each triple.

- Nodes denoted x_i and $\neg x_i$ cannot be selected together.

We can choose a valuation according to the selected nodes, since it follows from the previous discussion that it must exist.
(Arbitrary values can be assigned to the remaining variables.)

For the given valuation, the formula φ has surely the value 1, since in each clause there is at least one literal with value 1.

A Reduction from 3-SAT to IS

It is obvious that the running time of the described algorithm polynomial:

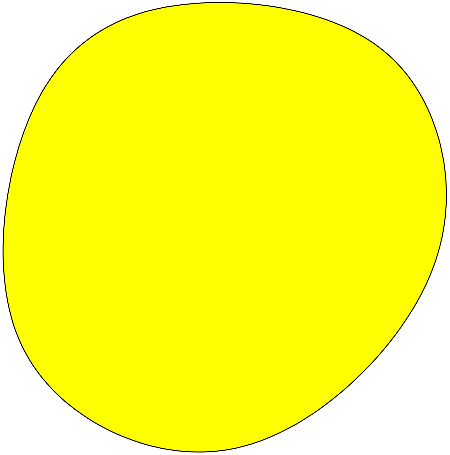
Graph G and number k can be constructed for a formula φ in time $O(n^2)$, where n is the size of formula φ.

We have also seen that there is an independent set of size k in the constructed graph G iff the formula φ is satisfiable.

The described algorithm shows that 3-SAT can be reduced in polynomial time to IS.

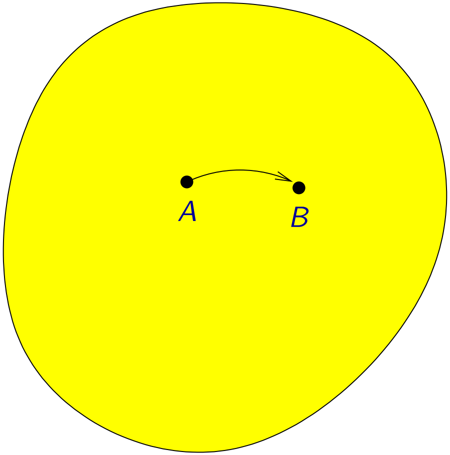
NP-Complete Problems

Let us consider a set of all decision problems.



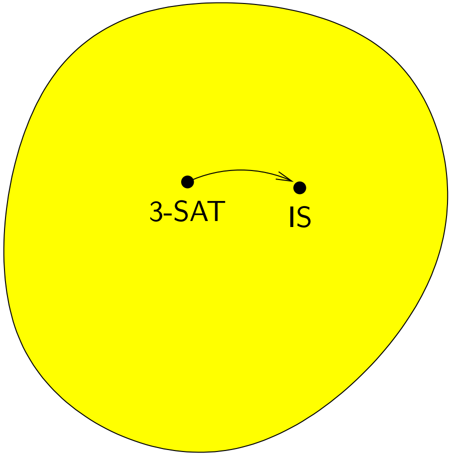
NP-Complete Problems

By an arrow we denote that a problem A can be reduced in polynomial time to a problem B.



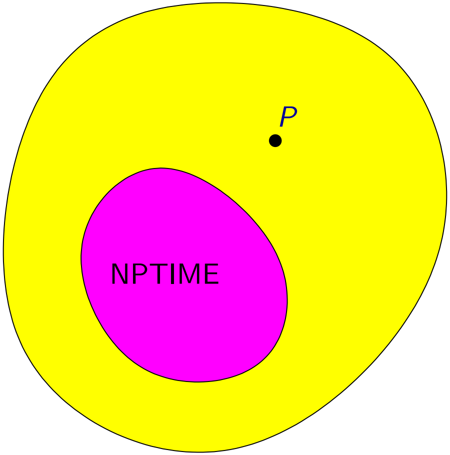
NP-Complete Problems

For example 3-SAT can be reduced in polynomial time to IS.



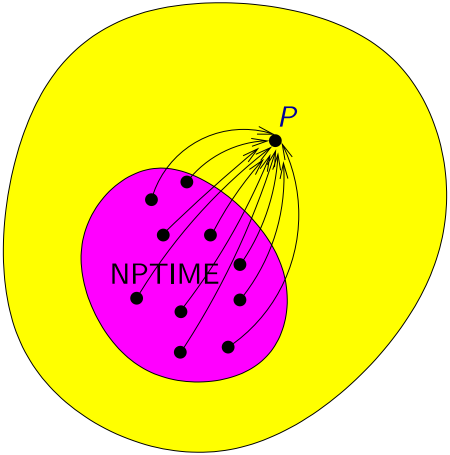
NP-Complete Problems

Let us consider now the class NPTIME and a problem P.



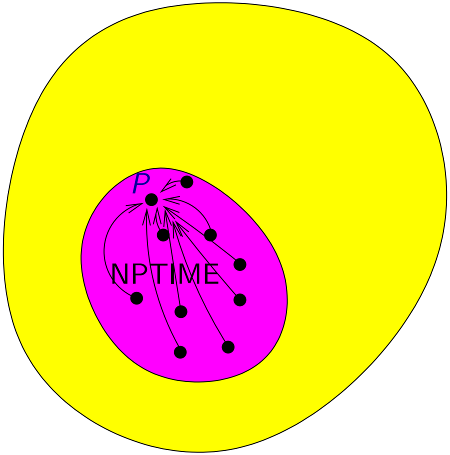
NP-Complete Problems

A problem P is NP-**hard** if every problem from NPTIME can be reduced in polynomial time to P.



NP-Complete Problems

A problem P is **NP-complete** if it is NP-hard and it belongs to the class NPTIME.



NP-Complete Problems

If we have found a polynomial time algorithm for some NP-hard problem P, then we would have polynomial time algorithms for all problems P' from NPTIME:

- At first we would apply an algorithm for the reduction from P' to P on an input of a problem P'.
- Then we would use a polynomial algorithm for P on the constructed instance of P and returned its result as the answer for the original instance of P'.

Is such case, PTIME $=$ NPTIME would hold, since for every problem from NPTIME there would be a polynomial-time (deterministic) algorithm.

NP-Complete Problems

On the other hand, if there is at least one problem from NPTIME for which a polynomial-time algorithm does not exist, then it means that for none of NP-hard problems there is a polynomial-time algorithm.

It is an open question whether the first or the second possibility holds.

NP-Complete Problems

It is not difficult to see that:

If a problem A can be reduced in a polynomial time to a problem B and problem B can be reduced in a polynomial time to a problem C, then problem A can be reduced in a polynomial time to problem C.

So if we know about some problem P that it is NP-hard and that P can be reduced in a polynomial time to a problem P', then we know that the problem P' is also NP-hard.

NP-Complete Problems

Věta (Cook, 1971)

Problem SAT is NP-complete.

It can be shown that SAT can be reduced in a polynomial time to 3-SAT and we have seen that 3-SAT can be reduced in a polynomial time to IS.

This means that problems 3-SAT and IS are NP-hard.

It is not difficult to show that 3-SAT and IS belong to the class NPTIME.

Problems 3-SAT and IS are NP-complete.

NP-Complete Problems

By a polynomial reductions from problems that are already known to be NP-complete, NP-completeness of many other problems can be shown:

$$\begin{tikzcd}
 & & & \text{VC} & & \\
 & & & \text{CLIQUE} & & \\
 & & \text{IS} \arrow[uur] \arrow[ur] \arrow[dr] & & & \\
\text{SAT} \arrow[r] & \text{3-SAT} \arrow[ur] \arrow[dr] \arrow[ddr] \arrow[dddr] & & \text{HC} \arrow[r] & \text{HK} \arrow[r] & \text{TSP} \\
 & & \text{3-CG} & & & \\
 & & \text{SUBSET-SUM} & & & \\
 & & \text{ILP} & & &
\end{tikzcd}$$

Examples of Some NP-Complete Problems

The following previously mentioned problems are NP-complete:

- SAT (boolean satisfiability problem)
- 3-SAT
- IS — independent set problem

On the following slides, examples of some other NP-complete problems are described:

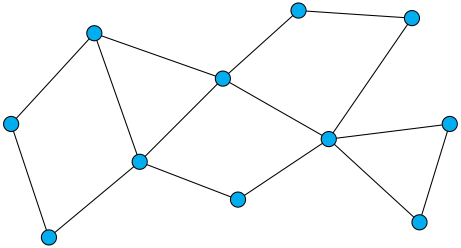
- CG — graph coloring (remark: it is NP-complete even in the special case where we have 3 colors)
- VC — vertex cover
- CLIQUE — clique problem
- HC — Hamiltonian cycle
- HK — Hamiltonian circuit
- TSP — traveling salesman problem
- SUBSET-SUM
- ILP — integer linear programming

Graph Coloring

Graph coloring

Input:	An undirected graph G, a natural number k.
Question:	Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



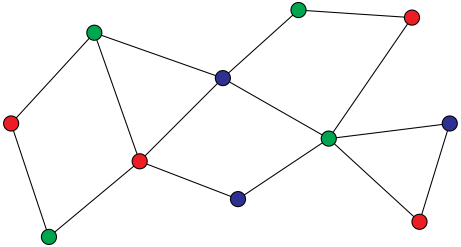
Graph Coloring

Graph coloring

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

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



Answer: YES

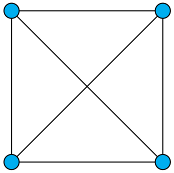
Graph Coloring

Graph coloring

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

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



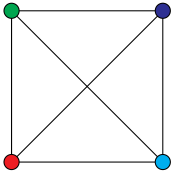
Graph Coloring

Graph coloring

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

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



Answer: No

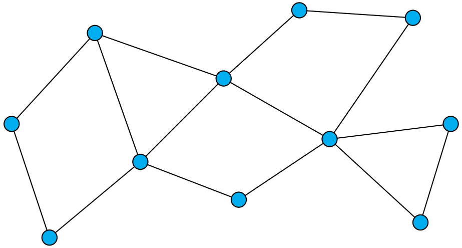
VC – Vertex Cover

VC – vertex cover

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

Question: Is there some subset of nodes of G of size k such that every edge has at least one of its nodes in this subset?

Example: $k = 6$



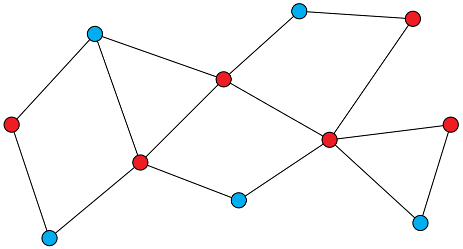
VC – Vertex Cover

VC – vertex cover

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

Question: Is there some subset of nodes of G of size k such that every edge has at least one of its nodes in this subset?

Example: $k = 6$



Answer: YES

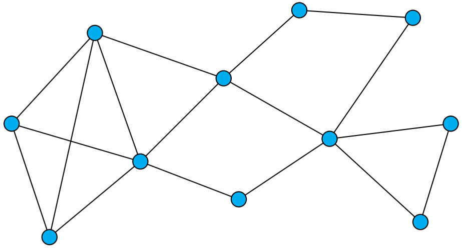
CLIQUE

CLIQUE

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

Question: Is there some subset of nodes of G of size k such that every two nodes from this subset are connected by an edge?

Example: $k = 4$



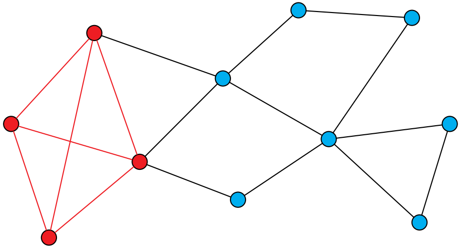
CLIQUE

CLIQUE

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

Question: Is there some subset of nodes of G of size k such that every two nodes from this subset are connected by an edge?

Example: $k = 4$



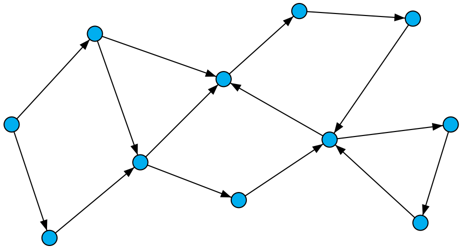
Answer: YES

Hamiltonian Cycle

HC – Hamiltonian cycle

Input:	A directed graph G.
Question:	Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



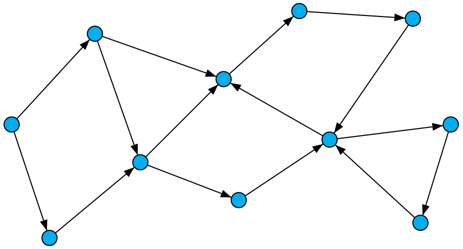
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



Answer: No

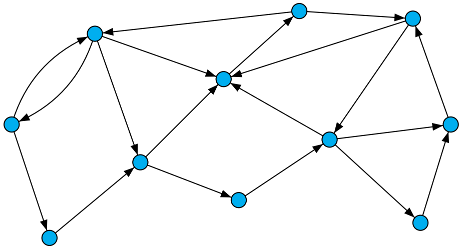
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



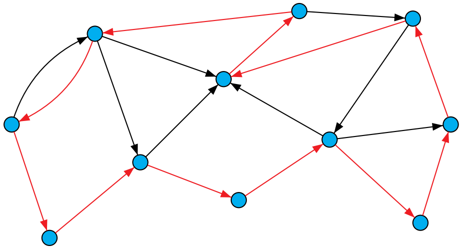
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



Answer: YES

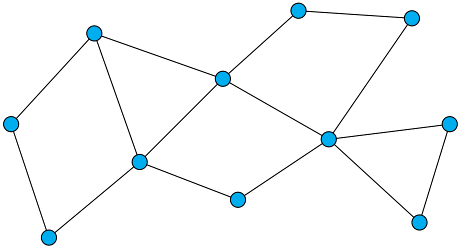
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G.

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



Answer: No

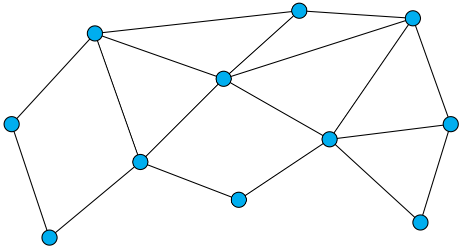
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G.

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



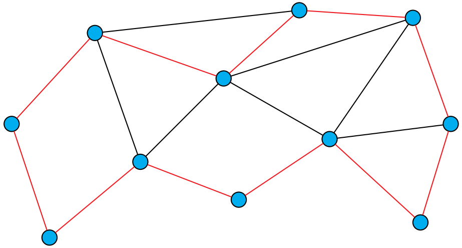
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G.

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



Answer: Yes

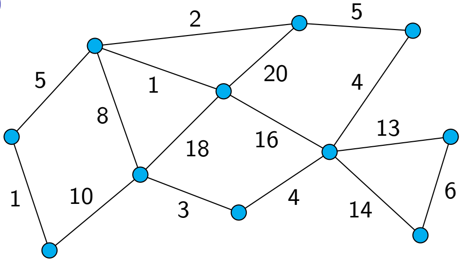
Traveling Salesman Problem

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers and a number k.

Question: Is there a closed tour going through all nodes of the graph G such that the sum of labels of edges on this tour is at most k?

Example: $k = 70$



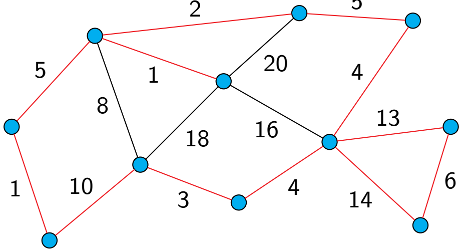
Traveling Salesman Problem

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers and a number k.

Question: Is there a closed tour going through all nodes of the graph G such that the sum of labels of edges on this tour is at most k?

Example: $k = 70$



Answer: YES, since there is a tour with the sum 69.

SUBSET-SUM

Problem SUBSET-SUM

Input: A sequence $a_1, a_2, \ldots, a_n$ of natural numbers and a natural number s.

Question: Is there a set $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i = s$?

In other words, the question is whether it is possible to select a subset with sum s of a given (multi)set of numbers.

Example: For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 15$ the answer is $\textsc{Yes}$, since $3 + 5 + 7 = 15$.

For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 16$ the answer is $\textsc{No}$, since no subset of these numbers has sum 16.

SUBSET-SUM

Remark:
The order of numbers $a_1, a_2, \ldots, a_n$ in an input is not important.

Note that this is not exactly the same as if we have formulated the problem so that the input is a set $\{a_1, a_2, \ldots, a_n\}$ and a number s — numbers cannot occur multiple times in a set but they can in a sequence.

SUBSET-SUM

Problem SUBSET-SUM is a special case of a **knapsack problem**:

Knapsack problem

Input: Sequence of pairs of natural numbers $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ and two natural numbers s and t.

Question: Is there a set $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i \leq s$ and $\sum_{i \in I} b_i \geq t$?

SUBSET-SUM

Informally, the knapsack problem can be formulated as follows:

We have n objects, where the i-th object weights a_i grams and its price is b_i dollars.

The question is whether there is a subset of these objects with total weight at most s grams (s is the capacity of the knapsack) and with total price at least t dollars.

Remark:

Here we have formulated this problem as a decision problem.

This problem is usually formulated as an optimization problem where the aim is to find such a set $I \subseteq \{1, 2, \ldots, n\}$, where the value $\sum_{i \in I} b_i$ is maximal and where the condition $\sum_{i \in I} a_i \leq s$ is satisfied, i.e., where the capacity of the knapsack is not exceeded.

SUBSET-SUM

That SUBSET-SUM is a special case of the Knapsack problem can be seen from the following simple construction:

Let us say that $a_1, a_2, \ldots, a_n$, s_1 is an instance of SUBSET-SUM.
It is obvious that for the instance of the knapsack problem where we have the sequence (a_1, a_1), $(a_2, a_2), \ldots,$ (a_n, a_n), $s = s_1$ and $t = s_1$, the answer is the same as for the original instance of SUBSET-SUM.

SUBSET-SUM

If we want to study the complexity of problems such as SUBSET-SUM or the knapsack problem, we must clarify what we consider as the size of an instance.

Probably the most natural it is to define the size of an instance as the total number of bits needed for its representation.

We must specify how natural numbers in the input are represented – if in binary (resp. in some other numeral system with a base at least 2 (e.g., decimal or hexadecimal) or in unary.

- If we consider the total number of bits when numbers are written in **binary** as the size of an input, no polynomial time algorithm is known for SUBSET-SUM.
- If we consider the total number of bits when numbers are written in **unary** as the size of an input, SUBSET-SUM can be solved by an algorithm whose time complexity is polynomial.

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:

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

ILP – Integer Linear Programming

One of solutions of the system

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

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

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

because

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

So the answer for this instance is $\textsc{Yes}$.

ILP – Integer Linear Programming

Remark: A similar problem where the question for a given system of linear inequations is whether it has a solution in the set of **real** numbers, can be solved in a polynomial time.

PSPACE-Complete and EXPTIME-Complete Problems

- A problem P is PSPACE-**hard** if for every problem P' from PSPACE there is a polynomial time reduction of P' to P.
- A problem P is PSPACE-**complete** if it is PSPACE-hard and belongs to PSPACE.

- A problem P is EXPTIME-**hard** if for every problem P' from EXPTIME there is a polynomial time reduction of P' to P.
- A problem P is EXPTIME-**complete** if it is EXPTIME-hard and belongs to EXPTIME.

$\vdots$

PSPACE-Complete and EXPTIME-Complete Problems

Generally, for arbitrary complexity class $\mathcal{C}$ we can introduce classes of $\mathcal{C}$-**hard** and $\mathcal{C}$-**complete** problems:

Definition

- A problem P is $\mathcal{C}$-**hard** if for every problem P' from the class $\mathcal{C}$ there is a polynomial time reduction of P' to P.
- A problem P is $\mathcal{C}$-**complete** if it is $\mathcal{C}$-hard and belongs to the class $\mathcal{C}$.

So in addition to NP-complete problems, we have PSPACE-complete problems, EXPTIME-complete problems, EXPSPACE-complete problems, 2-EXPTIME-complete problems, . . .

Generally speaking, $\mathcal{C}$-complete problems are always the hardest problems in the given class $\mathcal{C}$.

PTIME-Complete Problems, NL-Complete Problems, ...

Remark: The notions of $\mathcal{C}$-hard and $\mathcal{C}$-complete problems defined as above, where a polynomial time reductions are used, do not make much sense for the class PTIME and other classes, which are subsets of this class (such as NLOGSPACE).

For such classes, the notions of $\mathcal{C}$-hard and $\mathcal{C}$-complete problems are defined in a similar way as before but instead of polynomial time reductions they use so called **logspace reductions**:

- an algorithm performing the given reduction must be deterministic and to have a logarithmic space complexity

For example, the following classes are defined this way:

- PTIME-complete and PTIME-hard problems
- NLOGSPACE-complete and NLOGSPACE-hard problems (they are usually denoted with a shorter name as NL-complete and NL-hard)

An Example of NL-Complete Problem

A typical example of NL-complete problem:

Reachability in a Graph

Input: A directed graph G and two of its nodes s and t.

Question: Is there a path from the node s to the node t in the graph G ?

An Example of a PTIME-Complete Problem

A typical example of PTIME-complete problem:

Circuit Value Problem

Input:	An acyclic boolean circuit C consisting of gates and wires, and boolean values $x_1, x_2, \ldots, x_n$ on inputs of this circuit.
Question:	Is the value on the output of the circuit C equal to 1 for the given values of inputs?

Examples of PSPACE-Complete Problems

A typical example of a PSPACE-complete problem is the problem of **quantified boolean formulas** (**QBF**):

QBF

Input: A quantified boolean formula of the form

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots \exists x_{n-1} \forall x_n : \varphi,$$

where φ is a (standard) boolean formula containing variables $x_1, x_2, \ldots, x_n$.

Question: Is the given formula true?

Examples of PSPACE-Complete Problems

EqNFA

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

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

Universality of NFA

Input: A nondeterministic finite automaton $\mathcal{A}$.

Question: Is $\mathcal{L}(\mathcal{A}) = \Sigma^*$?

Examples of PSPACE-Complete Problems

EqRE

Input: Regular expressions α_1 and α_2.

Question: Is $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$?

Universality of RE

Input: A regular expression α.

Question: Is $\mathcal{L}(\alpha) = \Sigma^*$?

Examples of PSPACE-Complete Problems

Consider the following game played by two players on a directed graph G:

- Players alternately move a pebble on the nodes of graph G.
- In moves they mark those nodes that have been already visited.
- A play starts on the specified node v_0.
- Let us say that the pebble is currently on a node v. The player who is on turn choses a node v' such that there is an edge from v to v' and v' has not been visited yet.
- A player that does not have any possible move, loses, and his/her opponent wins.

Generalized Geografy

Input: A directed graph G with a denoted initial node v_0.

Question: Does the player that plays first have a winning strategy in the game played on the graph G with the initial node v_0 ?

Examples of EXPTIME-Complete Problems

A typical example of EXPTIME-complete problem:

Input: A Turing machine $\mathcal{M}$, a word w, and number k written in binary.

Question: Does the computation of the machine $\mathcal{M}$ on the word w halt in k steps?
(I.e., does the machine $\mathcal{M}$ perform at most k steps in the computation on the word w?)

Examples of EXPTIME-Complete Problems

Other examples of EXPTIME-complete problems are for example generalized variants of games such chess, checkers, or Go, played on a board of an arbitrary size (e.g., on a chessboard of size $n \times n$):

- the input is a position in the given game (e.g., in chess, a particular placement of pieces on a chessboard and information whose player is on turn)
- the question is if the player who is currently on turn, has a winning strategy in the given position

Examples of EXPSPACE-Complete Problems

Regular expressions with squaring are defined similarly as standard regular expressions but in addition to the standard operators $+$, $\cdot$, and *, they can contain unary operator ^{2} with the following meaning:

- α^2 is a shorthand for $\alpha \cdot \alpha$.

The following two problems are EXPSPACE-complete:

Input:	Regular expressions with squaring α_1 and α_2.
Question:	Is $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$?

Input:	A regular expression with squaring α.
Question:	Is $\mathcal{L}(\alpha) = \Sigma^*$?

Presburger Arithmetic

An example of a problem that is decidable but its computational complexity is very high:

Problem

Input: A closed formula of the first order predicate logic where the only predicate symbols are $=$ and $<$, the only function symbol is $+$, and the only constant symbols are 0 and 1.

Question: Is the given formula true in the domain of natural numbers (using the natural interpretation of all function and predicate symbols)?

A deterministic algorithm with time complexity $2^{2^{2^{O(n)}}}$ is known for this problem, and it is also known that every nondeterministic algorithm solving this problem must have a time complexity at least $2^{2^{\Omega(n)}}$.